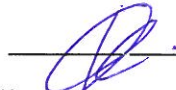


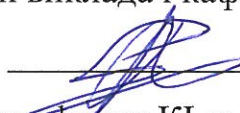
Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи бакалавра
на тему: «Проект інформаційної системи управління
розумним будинком»

Виконав студент 4-го курсу, групи КІ-3
спеціальності 123 «Комп'ютерна інженерія»

 В. О. Юрченко
Керівник старший викладач кафедри КН

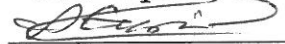
 Р. В. Петросян
Рецензент доцент кафедри КІ та КБ, к.т.н., доцент

 І. В. Пулеко

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА КІБЕРБЕЗПЕКИ
СПЕЦІАЛЬНІСТЬ 123 «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ»

ЗАТВЕРДЖУЮ

Зав. кафедри комп'ютерної
інженерії та кібербезпеки

 А. А. Єфіменко
« 12 » 04 20 21 р.

ЗАВДАННЯ

на випускню кваліфікаційну роботу

Студента Юрченка Владислава Олеговича

Тема роботи: Проект інформаційної системи управління розумним будинком

Затверджена Наказом університету від « 12 » квітня 2021 р. № 133/с

Термін здачі студентом закінченої роботи 29 червня 2021 р.

Вихідні дані роботи (зазначається предмет і об'єкт дослідження) _____
використання доступних мікроконтролерів та модулів для побудови системи управління розумним будинком; алгоритми та методи побудови системи управління розумним будинком.

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

Керівник  Р. В. Петросян

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	<i>Постановка задачі</i>	3.05.21	виконано
2	<i>Пошук, огляд та аналіз аналогічних розробок</i>	4.05.21 – 8.05.21	виконано
3	<i>Розробка плану роботи</i>	9.05.21 – 15.05.21	виконано
4	<i>Опрацювання літературних джерел</i>	16.05.21 – 27.05.21	виконано
5	<i>Проектування структурної схеми системи</i>	28.05.21 – 2.06.21	виконано
6	<i>Розробка програмного забезпечення</i>	3.06.21 – 10.06.21	виконано
7	<i>Тестування</i>	11.06.21 – 22.06.21	виконано
8	<i>Захист</i>	29.06.21	

Студент  В. О. Юрченко

Керівник  Р. В. Петросян

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається інформаційної системи управління розумним будинком та пояснювальної записки. Пояснювальна записка до випускної кваліфікаційної роботи містить 60 сторінок, 82 рисунки та 1 таблицю.

Метою роботи є проектування та розробка інформаційної системи управління розумним будинком.

В роботі визначено основні завдання для розробки системи, проаналізовано сучасні наявні аналоги. Розроблено структурну схему системи, для якої обрано та обґрунтовано мікроконтролер чи модуль для кожного окремого вузла системи. Досліджено деякі протоколи для зв'язку вузлів системи між собою, та розроблено один власний. Написано програмний код для реалізації роботи системи. Також, наводяться результати тестування системи.

Об'єктом дослідження є процес проектування та побудови інформаційної системи управління розумним будинком. Предметом дослідження є алгоритми та методи побудови системи управління розумним будинком.

КЛЮЧОВІ СЛОВА: РОЗУМНИЙ БУДИНОК, МІКРОКОНТРОЛЕР, ПРОТОКОЛ, ХАБ, ПРИСТРІЙ АВТОМАТИЗАЦІЇ.

					КІК.КР.Б. – 123 – 22 – ПЗ			
Зм.	Арк.	№ докум.	Підпис	Дата	Проект інформаційної системи управління розумним будинком	Літ.	Арк.	Аркушів
Розроб.		В. О. Юрченко		29.06.21			4	
Керівник		Р. В. Петросян		29.06.21	Пояснювальна записка	Житомирська політехніка, група КІ-3		
Рецензент.		І. В. Пулеко		29.06.21				
Зав. каф.		А. А. Єфіменко		29.06.21				

ABSTRACT

The final qualifying work of the bachelor consists of an information system for smart home management and an explanatory note. The explanatory note to the final qualifying work contains 60 pages, 82 figures and 1 table.

The purpose of the work is to design and develop a smart home management information system.

The main tasks for system development are defined in the work, modern available analogues are analyzed. The structural scheme of the system is developed, for which the microcontroller or the module for each separate node of the system is selected and substantiated. Some protocols for communication of system nodes have been researched, and one own has been developed. The program code for realization of work of system is written. Also, the test results of the system are given.

The object of research is the process of designing and building a smart home management information system. The subject of research is algorithms and methods of building a smart home management system.

KEYWORDS: SMART HOME, MICROCONTROLLER, PROTOCOL, HUB, AUTOMATION DEVICE.

ЗМІСТ

РЕФЕРАТ	4
ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	7
ВСТУП.....	8
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ УПРАВЛІННЯ РОЗУМНИМ БУДИНКОМ.....	11
1.1. Постановка задачі	11
1.2. Аналіз аналогів інформаційних систем управління розумним будинком	12
1.3. Проектування узагальненої структурної схеми системи управління розумним будинком	16
1.4. Обґрунтування вибору мікроконтролерів та модулів для реалізації системи	17
1.5. Обґрунтування вибору інструментальних засобів	20
Висновки до розділу 1	24
РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ УПРАВЛІННЯ РОЗУМНИМ БУДИНКОМ.....	25
2.1. Визначення варіантів використання та побудова структурної схеми системи	25
2.2. Опис алгоритмів взаємодії окремих модулів системи	26
2.3. Конфігурування та комутація мікроконтролерів та модулів системи.....	31
2.4. Розробка програмного забезпечення системи управління розумним будинком....	38
Висновки до розділу 2	48
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ УПРАВЛІННЯ	49
3.1. Порядок підключення та налаштування системи. Структура інтерфейсу	49
3.2. Можливості роботи з системою за допомогою стороннього програмного забезпечення.....	50
3.3. Тестування роботи системи	53
Висновки до розділу 3	55
ВИСНОВКИ.....	56
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ.....	58
ДОДАТКИ.....	61

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

МК – Мікроконтролер.

ПА – Пристрій автоматизації.

ПК – Персональний комп'ютер.

IDE (Integrated development environment) – Інтегроване середовище розробки.

HAL (Hardware Abstraction Layer) – бібліотека для створення програм для STM32.

STM32 – Скорочена назва мікроконтролера STM32F103C8.

IoT (Internet of Things) – Інтернет речей.

ВСТУП

Під терміном «розумний дім» розуміється місце проживання, в якому є побутова та кліматична техніка, освітлення, розважальні та охоронні системи та інші прилади, що можуть взаємодіяти між собою та керуватися локально вдома, або віддалено, за допомогою смартфона чи комп'ютера та Інтернету. Розумний дім дозволяє об'єднати техніку, пристрої та комунікації будинку для ефективного управління ними.

З динамічним розвитком інформаційно-комунікаційних технологій, IoT (Інтернет речей), великих даних та штучного інтелекту, системи управління розумним будинком дедалі, то все більше набувають популярності [1]. Це і не дивно, адже основна мета розумного будинку – забезпечити комфорт та безпеку житла людини. Також, важливою перевагою такої системи є можливість оптимізації та контролю ресурсоспоживання.

Говорячи про «розумний будинок» - зараз варто виділяти не інноваційність ідеї, а якість реалізації. Важливою задачею систем управління розумним будинком є організація тісної взаємодії пристроїв та датчиків таким чином, щоб виконувати різноманітні патерни управління домом. Користувачеві не потрібно турбуватися про прості речі – розумний будинок ввімкне опалення за деякий час до його приходу, вимкне світло або електроприлади, якщо ними не користуються; зробить все для комфорту людини. Також «розумний будинок» може забезпечувати захист – управляти сигналізацією, контролювати, чи не прорвало кран, тощо.

Концепція «розумний дім» створена дуже давно. Вважається, що вона поширилась серед експертів галузі після 1898 року, коли Нікола Тесла винайшов пульти дистанційного керування [2]. Першою системою розумного дому, що задовольняє сучасні концепції, прийнято вважати «Echo IV», що була розроблена в 1966 році. За допомогою цього пристрою можна контролювати побутову техніку та температуру в домі. Проте, через надвисоку ціну, пристрій не мав належного розповсюдження та хоч якоїсь комерційної вигоди.

Сьогодні нікого не здивуєш здатністю управляти приладами зі смартфона, чи то кондиціонер, чи то освітлення, чи то просто розетка. Практично будь-який пристрій можна синхронізувати з розумним будинком – через віддалені сервіси, або за допомогою підключення до хабу.

Хаб розумного будинку – це центральний керуючий пристрій, за допомогою якого з'єднуються та синхронізуються прилади розумного будинку. В ньому поєднуються різні протоколи для зв'язку з пристроями, дротові та бездротові. Також, зазвичай, хаб виступає в ролі шлюзу – з'єднуючи низку пристроїв з глобальною або локальною мережею.

Загалом, сучасна система розумного будинку ґрунтується на концепції IoT. Під цим поняттям мається на увазі модель, що дозволяє спілкуватися пристроям та датчикам за допомогою Інтернету, а також взаємодію між пристроями та людиною. IoT – технологія, що стрімко розвивається, так як надає інноваційні рішення багатьох питань та проблем, що виникають в державних та приватних галузях бізнесу по всьому світу [3].

Метою випускної кваліфікаційної роботи є проектування та розробка інформаційної системи управління розумним будинком.

Встановлена мета обумовлює наступні завдання:

- проведення аналізу задач, які має виконувати система управління;
- вибору та проектування архітектури системи, її узагальненої схеми;
- обґрунтування вибору мікроконтролерів та модулів для реалізації системи;
- проектування способів взаємодії окремих модулів системи;
- конфігурування та комутації мікроконтролерів та модулів системи;
- реалізація програмної частини системи.

Об'єктом дослідження є процес проектування та побудови інформаційної системи управління розумним будинком.

Предметом дослідження є алгоритми та методи побудови системи управління розумним будинком.

В роботі було використано методи дослідження протоколів комутації

мікроконтролерів, вивчення їх теорії, конфігурації та комутації мікроконтролерів та модулів, об'єктно-орієнтованого проектування та об'єктно-орієнтованого програмування мікроконтролерів та настільних додатків.

Практичне значення роботи полягає в тому, що розроблена інформаційна система буде використана як базис для створення та впровадження повноцінної системи управління розумним будинком.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ УПРАВЛІННЯ РОЗУМНИМ БУДИНКОМ

1.1. Постановка задачі

Основною метою даної роботи є проектування системи управління розумним будинком є: ознайомлення з принципами та концепціями розумного будинку, методами реалізації системи управління розумним будинком. Система буде здатна контролювати датчики, підключені дротовими або бездротовими протоколами, та управляти простими пристроями.

Результатом роботи буде реалізована інформаційна система, яка демонструватиме можливості управління розумним будинком. Для її реалізації, потрібно виконати такі етапи:

1. Розробити структурну схему системи та дослідити методи її побудови.

1.1. Створити узагальнену схему, що складатиметься з:

- хаба;
- пристрою для налаштування системи;
- пристрою для керування системою;
- пристрою автоматизації, підключеного через радіозв'язок;
- пристрою автоматизації, підключеного через мережу Інтернет;
- пристрою автоматизації, підключеного дротовим способом.

1.2. Для кожного окремого блока системи дослідити варіанти та обрати мікроконтролер або модуль для його реалізації.

1.3. Обрати інструментальні засоби для конфігурування та програмування мікроконтролерів та модулів.

1.4. Розробити повну структурну схему системи, включаючи.

2. Реалізувати апаратну частину системи.

2.1. Дослідити протоколи та інтерфейси схеми, такі як:

- SPI;

- UART;
- I²C;
- ESB;
- MQTT.

2.2. Розробити власний протокол для з'єднання модуля Wi-Fi та центрального мікроконтролера хабу, що працюватиме поверх протоколів UART та MQTT.

2.2.1. Спроектувати пакет для передачі даних з перевіркою цілісності за допомогою контрольної суми.

2.2.2. Розробити систему для передачі команд для обміну даними та службовими командами.

2.3. Провести комутацію всіх модулів системи.

3. Реалізувати програмну частину системи.

3.1. Дослідити методи та реалізувати конфігурації мікроконтролерів.

3.2. Дослідити методи та реалізувати програмування мікроконтролерів.

3.2.1. Написати програмний код для комутації мікроконтролерів;

3.2.2. Написати програмний код для роботи пристроїв автоматизації.

3.3. Створити настільний додаток для налаштування хабу

1.2. Аналіз аналогів інформаційних систем управління розумним будинком

Перш, ніж почати аналізування систем управління розумним будинком, виділимо деякі критерії їх оцінювання:

- спосіб управління хабом (за допомогою смартфона, комп'ютера, окремої панелі)
- канали зв'язку для управління хабом (дротовий, Інтернет, радіоканал);
- протоколи зв'язку пристроїв автоматизації з хабом та між собою (дротовий, бездротовий);
- пристрої, що входять в комплект;
- можливість масштабування;

– вартість.

Загалом, існують 3 основних екосистеми для реалізації роботи розумного будинку: «Amazon Alexa», «Google Assistant», «Apple HomeKit» [4].

Розглянемо систему управління розумним домом від виробника «Xiaomi», зокрема хаб «Xiaomi Mi Smart Home Hub» (рис. 1.1).



Рисунок 1.1 – Хаб «Xiaomi Mi Smart Home Hub»

«Xiaomi» можна вважати бюджетним представником виробника пристроїв для розумного дому. Але, розглянувши детальніше хаб цього виробника, критичних мінусів, що б показували його в негативному світлі знайти важко. Серед переваг хабу та інших пристроїв виробника варто зазначити:

- можливість повної автономності датчиків;
- великий вибір пристроїв для масштабування;
- можливість під'єднання пристроїв інших виробників, завдяки бездротовому протоколу ZigBee;
- управління смартфоном через Wi-Fi;
- можливість підключення до екосистеми «Apple HomeKit»;
- низька вартість стартового комплекту (близько 90\$).

Якщо розглядати мінуси, слід виділити:

- невелика зона дії сигналу, не більше 10 м;
- відсутність українського інтерфейсу;
- відсутність резервного живлення хабу.

Далі розглянемо українського виробника систем для управління розумним будинком «Ајах» на прикладі хабу «Ајах Hub 2 Plus» (рис. 1.2). Основною відмінністю цієї системи є її надійне шифрування та захищеність радіозв'язку.



Рисунок 1.2 – Хаб «Ajax Hub 2 Plus»

Розглядаючи переваги хабу та інших пристроїв виробника, варто приділити увагу таким:

- велика зона дії сигналу, до 2 км;
- резервне автономне живлення, до 15 годин;
- пропрієтарний, захищений та зашифрований канал радіозв'язку між пристроями розумного дому;
- Ethernet, Wi-Fi та LTE канали зв'язку;
- багато варіантів сповіщення користувача: SMS-повідомлення, повідомлення в додатку Android та дзвінок;
- стійкий до саботажу;
- середня вартість за стартовий комплект (від 180\$)
- можливість підключення до 200 пристроїв

Серед мінусів варто зазначити:

- відсутність автономних датчиків;
- власна екосистема;
- можна під'єднати сумісні датчики лише декількох виробників;
- управління лише за допомогою смартфона.

Також, варто розглянути виробника систем та пристроїв для управління розумним будинком «BroadLink», зокрема хаб «BroadLink RM-Pro Wi-Fi» (рис. 1.3).



Рисунок 1.3 – Хаб «BroadLink RM-Pro Wi-Fi»

Серед переваг даного хабу, варто зазначити такі:

- швидкий монтаж та легке налаштування;
- широкий асортимент датчиків цього виробника;
- легке масштабування та управління пристроями;
- канали зв'язку – Wi-Fi, інфрачервоний зв'язок та радіозв'язок;
- в комплекті є пульт для керування хабом;
- підтримує синхронізацію з «Google Assistant» та «Amazon Alexa»;
- середня вартість за стартовий комплект (близько 200\$).

Мінусами можна зазначити:

- середню зону дії сигналу, до 50 м;
- відсутність резервного живлення хабу.

Переглянувши та проаналізувавши системи управління розумним будинком від різних виробників, варто підсумувати, що, загалом, кожен окремий продукт не сильно відрізняється від своїх конкурентів. Здебільшого, різниця в каналі зв'язку, котрим з'єднані датчики системи.

Враховуючи плюси та мінуси проаналізованих пристроїв, можна виділити особливості, за котрими створимо власну систему управління розумним

будинком:

- можливість вибору каналу зв'язку, за котрим датчики підключатимуться до хабу (Wi-Fi або радіозв'язок або дротовий зв'язок);
- при підключенні датчиків за допомогою Wi-Fi, цей датчик матиме змогу працювати автономно;
- невисока ціна.

1.3. Проектування узагальненої структурної схеми системи управління розумним будинком

Для розробки системи управління розумним будинком, використаємо схему, в котрій центральним є хаб (рис. 1.4), який, для налаштування, може бути підключеним до комп'ютера.

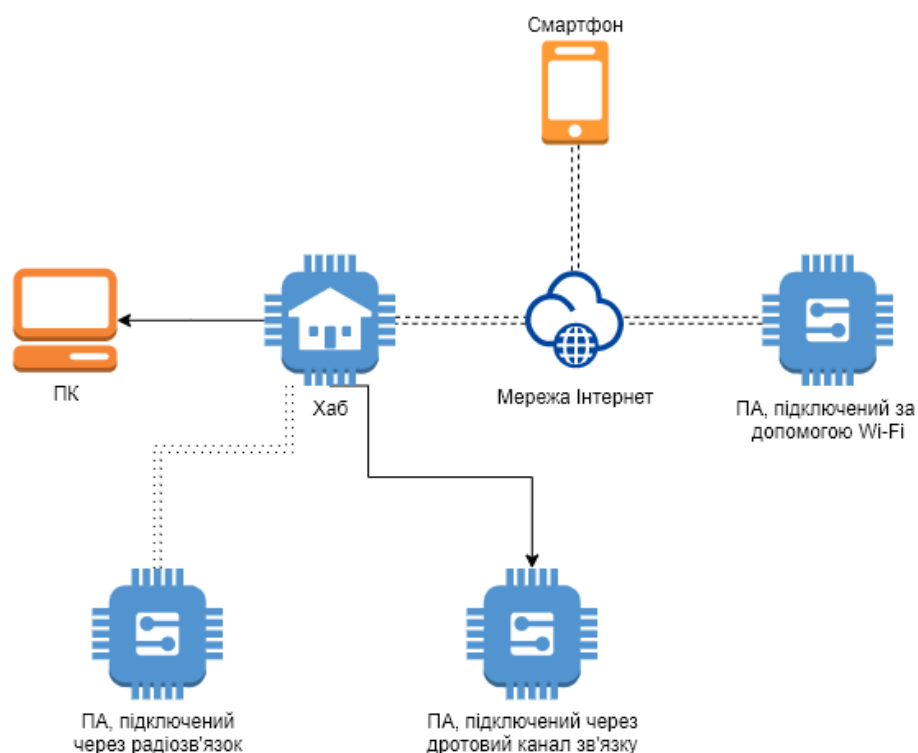


Рисунок 1.4 – Узагальнена структурна схема системи управління розумним будинком

В цій схемі, хаб – це центральний керуючий пристрій, який синхронізує датчики між собою, а також виступає в ролі мережевого шлюзу.

Датчики та ПА можуть підключатись до хабу як через дротовий зв'язок, так і через радіозв'язок. Додатково, ПА можуть працювати в автономному режимі при підключенні за допомогою Wi-Fi.

Керується система за допомогою смартфона, що підключений до мережі Інтернет.

1.4. Обґрунтування вибору мікроконтролерів та модулів для реалізації системи

Мікроконтролер «STM32F103C8» є частиною модульної плати, відомої як «Blue pill» (рис. 1.5) [5]. МК побудований на архітектурі ARM 32-bit Cortex-M3.

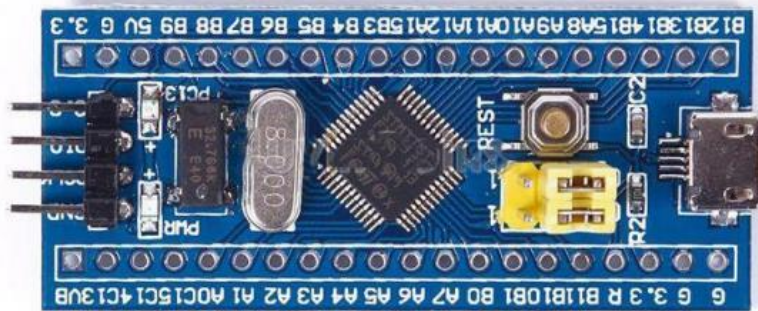


Рисунок 1.5 – Зовнішній вигляд STM32F103C8

Мікроконтролер був обраним в якості центрального для хаба через низку переваг, котрі необхідні для побудови мережі:

- вбудована підтримка до 3-ьох інтерфейсів USART;
- вбудована підтримка до 2-ох інтерфейсів SPI;
- вбудована підтримка до 2-ох інтерфейсів I²C;
- можливість покрокового відлагодження «з коробки»;

Інші характеристики мікроконтролера:

- частота - до 72 МГц (через мультиплікатор, кристал - 8 МГц);
- 7 таймерів, до 9 інтерфейсів комунікації (I2C, SPI, UART, USB, CAN);
- програмна пам'ять - 64Кб;
- оперативна пам'ять - 20Кб;
- живлення логіки - 2-3.6В, толерантне до 5В.

- для кінцевих пристроїв автоматизації не потрібна висока продуктивність;
- вбудований USB-UART перетворювач.



Інші характеристики АТmega328Р:

- Частота: 16 МГц;
- Інтерфейси: SPI, I2C, UART;
- Програмна пам'ять: 32 кБ;
- Оперативна пам'ять: 2 кБ.

Також, завдяки вбудованому USB-UART перетворювачі «Arduino Nano V3» зручно використовувати як модуль UART-TTL для забезпечення зв'язку між хабом та комп'ютером для налаштування. Для цього потрібно лише замкнути вихід мікроконтролера «RST» на землю.

В якості радіомодуля було використано плату nRF24L01+ (рис. 1.7) [7].

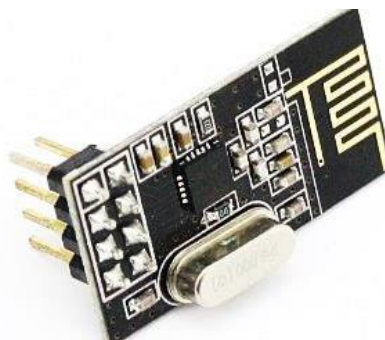


Рисунок 1.7 – Зовнішній вигляд радіомодуля nRF24L01+

Основні характеристики радіомодуля:

- напруга живлення – 1,9 – 3,6 В;
- частота роботи – 2.4 ГГц;
- кількість частотних каналів – 126 з кроком 1 МГц
- швидкості роботи – 256 кбіт/с, або 1 Мбіт/с, або 2 Мбіт/с.

В якості Wi-Fi-модуля був обраний ESP8266. Це досить поширений та доступний модуль. Для використання в хабі, була обрана його варіація – плата «ESP-01S» (рис. 1.8) [8], завдяки зручно виведеним контактам для підключення по UART.

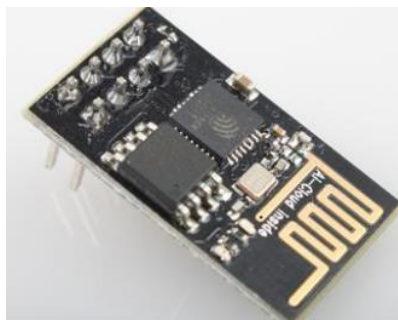


Рисунок 1.8 – Зовнішній вигляд плати ESP-01S

Для пристрою автоматизації була обрана плата «NodeMCU 0.9» (рис. 1.9) [9], так як в ній є повноцінні виведені виходи мікроконтролера, реалізовано живлення через USB та вбудований USB-UART перетворювач, за допомогою якого можна буде налаштувати SSID та пароль Wi-Fi мережі.

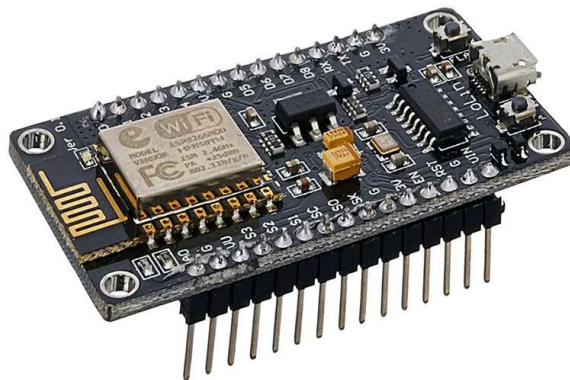


Рисунок 1.9 – Зовнішній вигляд NodeMCU 0.9

Основні характеристики ESP8266:

- 80 MHz 32-bit процесор Tensilica Xtensa L106 з можливістю розгону до 160 МГц;

- IEEE 802.11 b/g/n Wi-Fi. Підтримується WEP та WPA/WPA2;
- 14 портів вводу-виводу (з них можливо використовувати 11), SPI, I²C, I²S, UART, 10-bit АЦП;
- живлення 2,2...3,6 В.

1.5. Обґрунтування вибору інструментальних засобів

Для конфігурації мікроконтролера STM32F103C8 використовуватиметься «CubeMX» (рис. 1.10). Це зручний інструмент для налаштування портів вводу/виводу, таймерів, інтерфейсів комунікації, тощо. Також ним можна конфігурувати тактування. На виході отримуємо пустий проект з обраними налаштуваннями МК.



Рисунок 1.10 – Зовнішній програми CubeMX

Для розробки прошивки та програмування мікроконтролера STM32F103C8 використовуватиметься STM32CubeIDE (рис. 1.11). Це кросплатформне програмне забезпечення щоб розробляти проекти для мікроконтролерів сімейства STM32 з можливістю відлагодження програми за допомогою утиліти «OpenOCD». Мова програмування – C/C++.

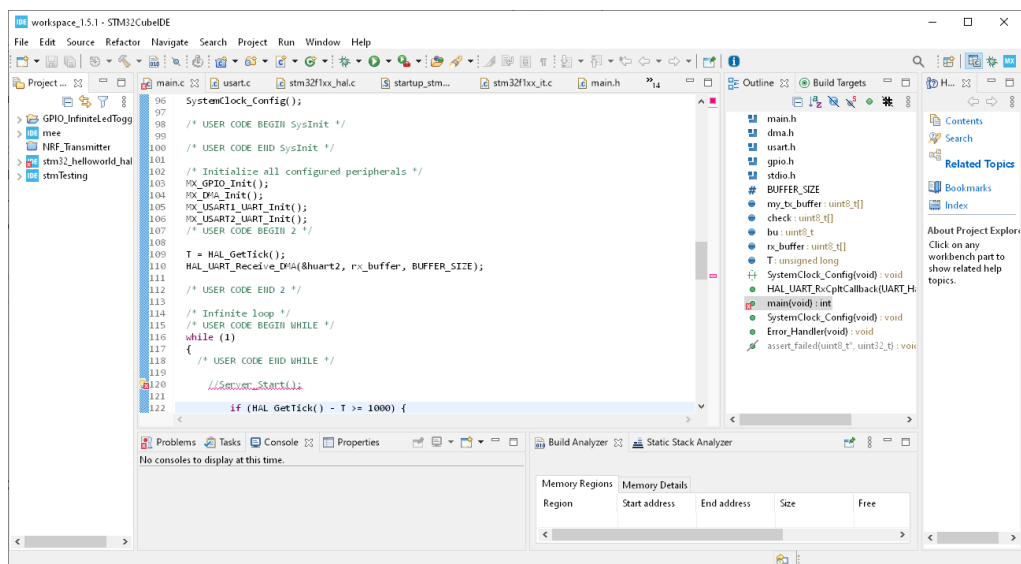


Рисунок 1.11 – Зовнішній вигляд програми STM32CubeIDE

Для програмування мікроконтролерів ATmega328 та ESP8266 використаємо «Visual Studio Code» зі встановленим розширенням «PlatformIO» [10]. На відміну від «Arduino IDE», яку зазвичай обираються для даних плат, «Visual Studio Code» має ряд основних переваг:

- технологія «IntelliSense» та підсвічування синтаксису;
- навігація по коду;
- інтегрований менеджер бібліотек;
- гнучке середовище розробки та розширюваність;
- не обмежена можливість одночасно розробляти декілька проектів;
- вбудований менеджер бібліотек та прикладів коду.

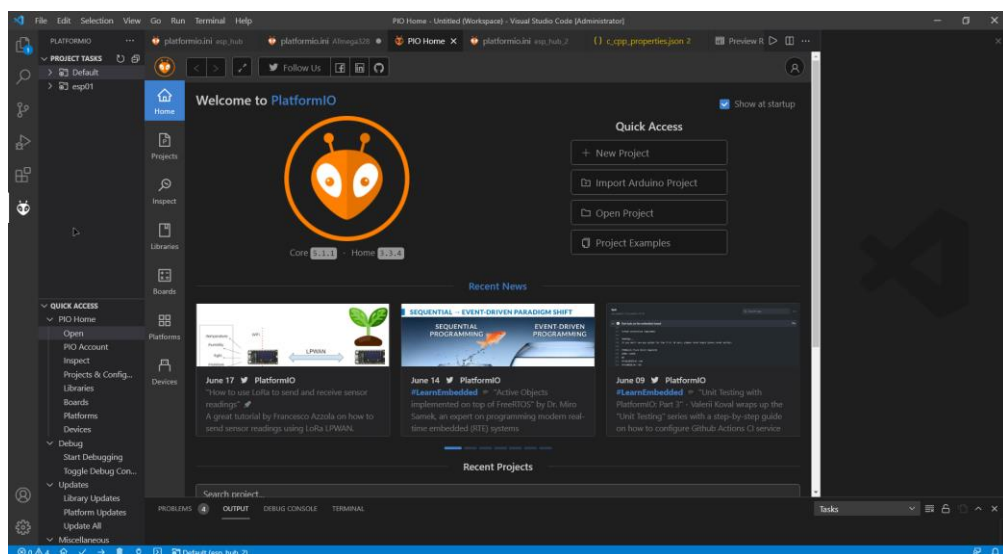


Рисунок 1.12 – Зовнішній вигляд Visual Studio Code

Для початку роботи з мікроконтролерами ATmega328 та ESP8266 – потрібно завантажити відповідні платформи (рис. 1.13).

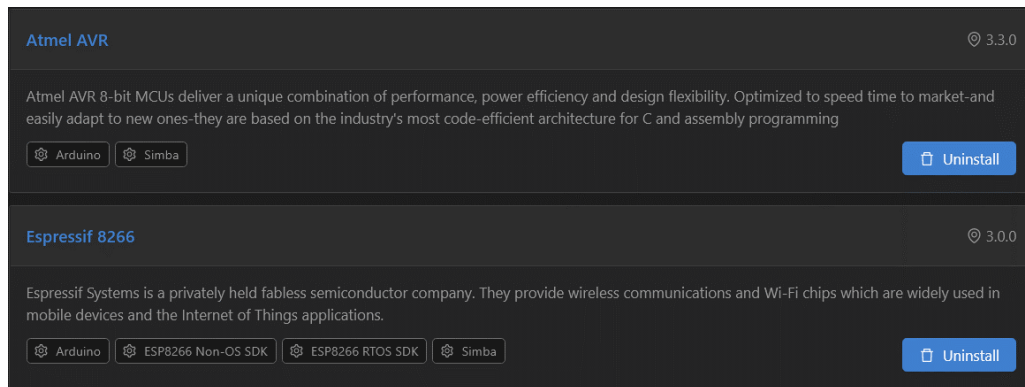


Рисунок 1.13 – Завантажені платформи для ATmega328 та ESP8266

Потім, налаштувати конфігураційний файл «platformio.ini» для окремо для кожного мікроконтролера (рис. 1.14, 1.15).

```
[env:nanoatmega328]
platform = atmelavr
board = nanoatmega328
framework = arduino
upload_port = COM5
upload_speed = 9600
lib_extra_dirs = C:\Users\Meow\Documents\Arduino\libraries
```

Рисунок 1.14 – Вміст файлу «platformio.ini» для роботи з ATmega328

```
[env:esp01_1m]
platform = espressif8266
board = esp01_1m
framework = arduino
upload_port = COM3
upload_speed = 115200
lib_extra_dirs = C:\Users\Meow\Documents\Arduino\libraries
```

Рисунок 1.15 – Вміст файлу «platformio.ini» для роботи з ESP8266

Для роботи з COM-портом, використаємо програму «Advanced Serial Port Terminal» (рис. 1.16). На відміну від аналогів (наприклад, вбудованого в «Arduino IDE» інструмента «Serial monitor»), дана програма має низку переваг:

- можливість відключатись від порту, не закриваючи вікно моніторингу;
- можливість переглядати дані в такому ж форматі, як їх відправив мікроконтролер, разом з даними, форматованими в текст;
- можливість відправляти дані в різних форматах;
- можливість налаштування циклічної відправки даних.

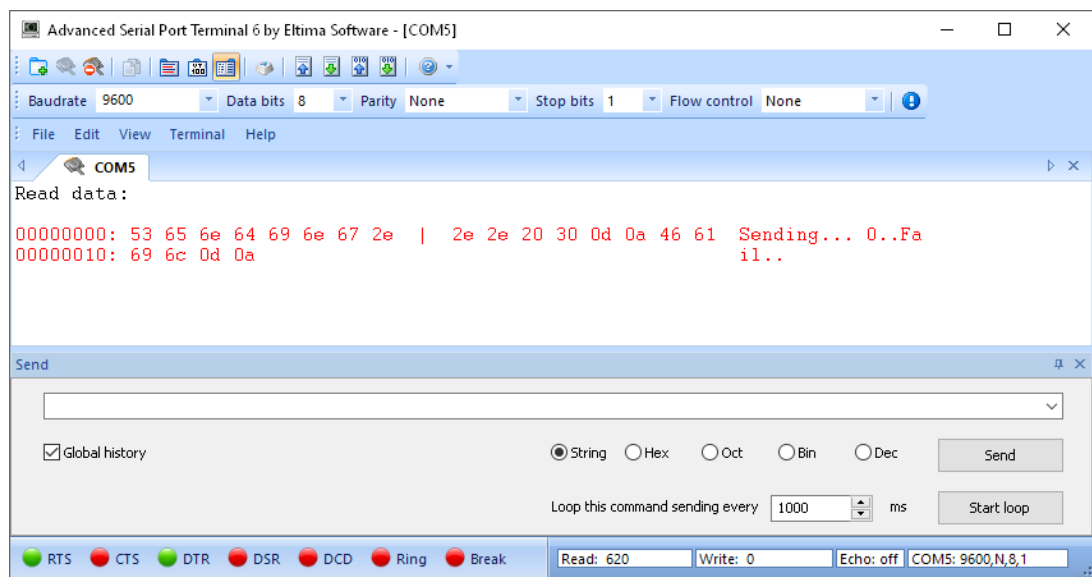


Рисунок 1.16 – Зовнішній вигляд програми Advanced Serial Port Terminal

Для розробки настільного додатку для комп'ютера, використаємо середовище розробки «Visual Studio» (рис. 1.17), платформу «.NET Framework», мову програмування «C#» та технологію «WPF».

C# – це об'єктно-орієнтована мова програмування зі строгою типізацією та вбудованим збиральником сміття, що дозволяє розробляти безпечні та надійні додатки, не хвилюючись проблемами очищення стеку та витоку пам'яті.

Для технології «WPF», Visual Studio надає зручний конструктор користувацького інтерфейсу заснований на елементах платформи «.NET Framework», що дозволяє, за допомогою мови розмітки «XAML», зручно та швидко розроблювати додатки.

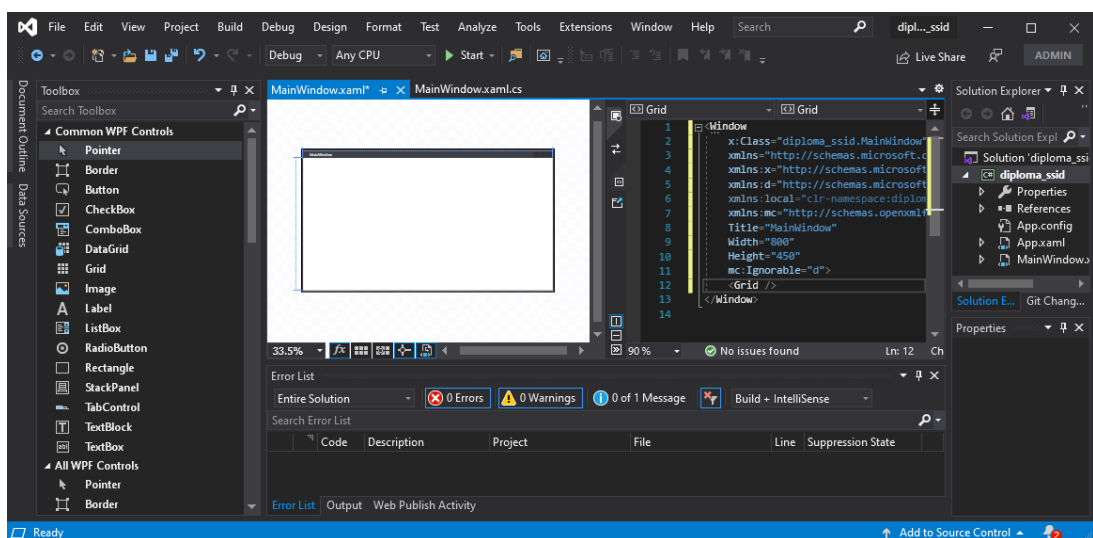


Рисунок 1.17 – Зовнішній вигляд Visual Studio

Висновки до розділу 1

Було проведено аналіз постановки задачі, що дозволило чіткіше розуміти мету даної роботи та її результат. Було розглянуто 3 аналоги систем управління розумним будинком: «Xiaomi Mi Smart Home Hub», «Ajax Hub 2 Plus» та «BroadLink RM-Pro Wi-Fi». Після дослідження їх переваг і недоліків, було виділено особливості, котрі лежатимуть в основі розробки власної системи управління розумним будинком.

Зважаючи на структурну схему, для реалізації системи управління розумним будинком були обрані такі мікроконтролери та плати:

- мікроконтролер «STM32F103C8»;
- плата «Arduino Nano V3»;
- радіомодуль «nRF24L01+»;
- Wi-Fi модуль «ESP8266»;
- плата «NodeMCU 0.9».

Зважаючи на такі параметри як зручність користування та написання програмного коду, швидкість розробки, функціональні можливості, було обрано інструментальні засоби для розробки системи: «CubeMX», «STM32CubeIDE», «Visual Studio Code», «Advanced Serial Port Terminal», «Visual Studio».

Хаб складається з центрального мікроконтролера (МК), модуля UART для зв'язку з ПК, модуля Wi-Fi для підключення до глобальної або локальної мережі та радіомодуля. Для початку роботи з хабом – його потрібно налаштувати. Це виконується за підключення до комп'ютера за допомогою UART. Потрібно запустити відповідне програмне забезпечення, обрати COM-порт, до якого підключений хаб та ввести SSID та пароль мережі. Після цього хаб збереже введені дані в енергонезалежну пам'ять мікроконтролера та використає їх при повторному підключенні до мережі.

Підключати пристрої автоматизації можна декількома способами:

- підключення каналом радіозв'язку за допомогою протоколу ESB (ПА1);
- дротовим способом, за допомогою шини I²C (ПА2);
- при підключенні до мережі Інтернет за допомогою модуля Wi-Fi (ПА3), при цьому, такий пристрій матиме змогу працювати автономно.

Робота пристроїв автоматизації через мережу Інтернет буде реалізована за допомогою протоколу MQTT та віддаленого MQTT-серверу.

2.2. Опис алгоритмів взаємодії окремих модулів системи

При першому включенні системи управління, потрібно з'єднати її з комп'ютером за допомогою UART-TTL конвертера, аби налаштувати SSID та пароль Wi-Fi-мережі.

UART (Universal Asynchronous Receiver/Transmitter) – це універсальний асинхронний приймач-передавач для послідовної передачі даних, швидкість та формат яких можуть бути сконфігуровані. Також, існує ще USART (Universal Synchronous/Asynchronous Receiver/Transmitter), який відрізняється лише тим, що дані в ньому можуть додатково передаватися синхронно. UART, зазвичай, використовується з іншими комунікаційними стандартами, наприклад RS-232 [11].

Протокол відіграє велику роль в організації зв'язку між пристроями. Вбудовані системи, мікроконтролери та комп'ютери, в основному,

використовують UART як форму апаратного протоколу зв'язку між пристроями. Однією з найважливіших переваг цього протоколу є те, що він використовує всього два дроти для передачі даних (рис. 2.2):

- RX – приймач;
- TX – передавач.

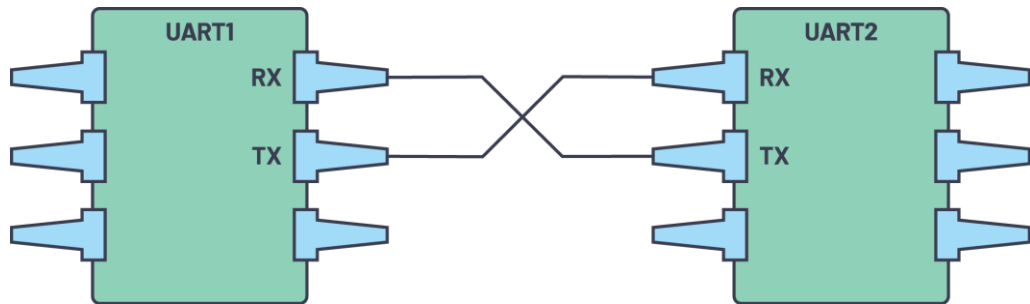


Рисунок 2.2 – Підключення двох пристроїв за допомогою UART

Також, в нашій системі, за допомогою UART підключені Wi-Fi-модулі. Для роботи системи через мережу Інтернет був використаний мережевий протокол MQTT (Message Queue Telemetry Transport). Це легкий та гнучкий мережевий протокол, побудований поверх TCP/IP стеку [12]. Його ключова особливість – модель публікації та підписки (рис. 2.3).

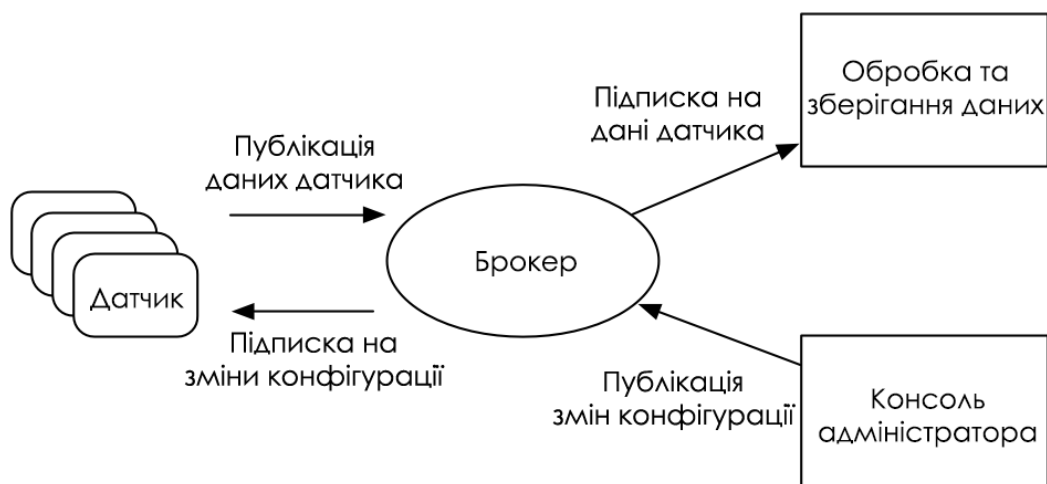


Рисунок 2.3 – Схематичне відображення роботи протоколу MQTT

Протокол MQTT працює з двома типами об'єктів мережі:

- брокер, лише один;
- клієнт, може бути декілька.

Брокер – це сервер, який приймає дані клієнту та, ґрунтуючись на

підписках, направляє ці дані іншим клієнтам. Клієнт – це все, що взаємодіє з брокером в процесі обміну повідомленнями. Клієнтом може бути не лише датчик, а і, наприклад, додаток, що обробляє дані датчиків.

Для зручної передачі даних в системі та, зокрема, даних MQTT-брокера між хабом та Wi-Fi-модулем, розробимо протокол поверх UART (табл. 2.1).

Таблиця 2.1

Опис протоколу передачі даних в системі поверх UART

Номер байта	0	1	2	3	4	5	6	7	8
Призначення	Сигнатура	Індекс пристрою	Тип даних	Дані				Контрольна сума	

В пакеті для передачі даних є такі поля:

1. Сигнатура – це стартовий байт «0xFF».
2. Індекс пристрою – його порядковий номер для ідентифікації. Індекс «0xFF» буде використовуватись для службових команд. Враховуючи це, максимальна кількість пристроїв – 254.
3. Тип даних – ідентифікатор для вказання типу, що буде передаватися.

Можливі такі варіанти:

- «0x01» - тип даних «bool»;
- «0x02» - тип даних «int8_t»;
- «0x03» - тип даних «uint8_t»;
- «0x04» - тип даних «int16_t»;
- «0x05» - тип даних «uint16_t»;
- «0x06» - тип даних «int32_t»;
- «0x07» - тип даних «uint32_t»;
- «0x08» - тип даних «float».

4. Дані, поміщені в 4 байти.
5. Контрольна сума. Для її обрахунку потрібно додати всі байти, інвертувати та прибавити 1.

Радіомодулі системи підключаються за допомогою інтерфейсу SPI (див. рис. 2.1). SPI (Serial Peripheral Interface, послідовний периферійний інтерфейс) –

один із найвживаніших інтерфейсів для реалізації зв'язку між мікроконтролером та периферійними мікросхемами такими як: датчики, АЦП, ЦАП, регістри зсуву, SRAM та інші [13]. Це синхронний повнодуплексний інтерфейс в основі якого можна описати принципами ведучого і веденого. Дані, що передаються, синхронізуються по зростаючому або спадаючому фронту імпульсів. Дані можуть одночасно передавати як ведучий, так і ведений пристрій.

Інтерфейс SPI буває трьохпровідним або чотирьохпровідним. Розглянемо детальніше чотирьохпровідний інтерфейс (рис. 2.3), так як він є сильно популярнішим, ніж аналог.

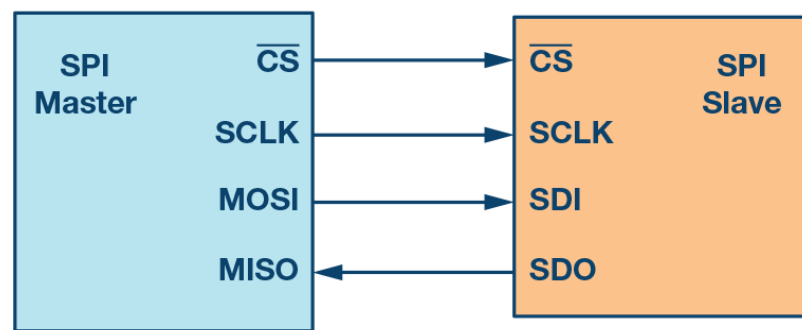


Рисунок 2.4 – Інтерфейс SPI з одним ведучим та веденим пристроєм

Для організації роботи SPI потрібно 4 сигнали:

- SCLK (SPI CLK) – тактовий сигнал;
- CS (Chip select) – сигнал початку або завершення передачі даних;
- MOSI (Master Out Slave In) – вихід ведучого пристрою, вхід веденого;
- MISO (Master in, slave out) – вихід веденого пристрою, вхід ведучого.

Дані, що передаються, синхронізуються за допомогою тактового сигналу, який генерується ведучим пристроєм. Інтерфейси SPI можуть мати лише одного ведучого та одного або декількох ведених пристроїв.

Для передавання даних через радіозв'язок слугує протокол ESB (Enhanced ShockBurst). Це базовий протокол, котрий підтримує двосторонній зв'язок пакетів даних, включаючи буферизацію пакетів, їх підтвердження та автоматичну повторну передачу втрачених пакетів [14]. Застосовується для мікроконтролерів серії nRF24L та nRF5.

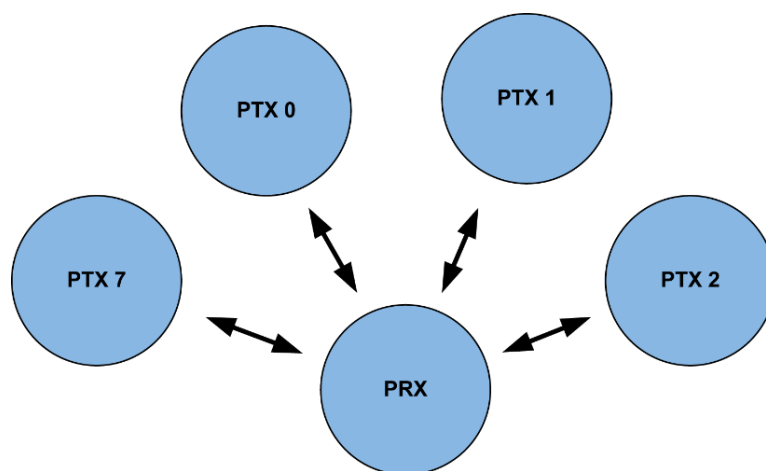


Рисунок 2.5 – Мережа ESB

ESB має автоматичну обробку пакетних транзакцій. Транзакція ESB – це обмін пакетами між двома пристроями, де є один первинний приймач (PRX) та один первинний передавач (PTX) (рис. 2.4). ESB дозволяє організувати цифровий інтерфейс з управляючим мікроконтролером по послідовній шині SPI. Цей протокол є вдосконаленою версією протоколу ShockBurst. Основна відмінність – ESB дозволяє організувати просту та ефективну передачу даних з зворотнім зв’язком. Пакет даних з корисними даними може бути розміром від 1 до 32 байт (рис. 2.5).

Заголовок 1 байт	Адрес 3-5 байт	9 байт	Данные 1-32 байта	CRC 0/1/2 байта
---------------------	----------------	--------	-------------------	--------------------

Рисунок 2.6 – Пакет даних ESB

Також, під’єднання пристроїв автоматизації можливо за допомогою шини I²C. I²C – це двопровідний послідовний інтерфейс, розроблений компанією Philips для використання в споживчих товарах [15]. Наразі, відрахування за використання протоколу скасовані, але можуть зберігатися в випадках, коли потрібне виділення ексклюзивної адреси на шині.

Протокол побудований за принципами «ведучий-ведений», де ведучий синхронізує шину, звертається до підлеглих та записує або зчитує дані до і з регістрів. Ведені – це пристрої які реагують лише при зверненні на їх унікальну адресу ведучого пристрою, тому варто уникати дублювання адрес серед ведених. Ведені не можуть ініціювати передачу даних.

Шина I²C використовує лише дві двонаправлені лінії (рис. 2.4):

- SDA – послідовна лінія даних;
- SCL – послідовна тактова лінія.

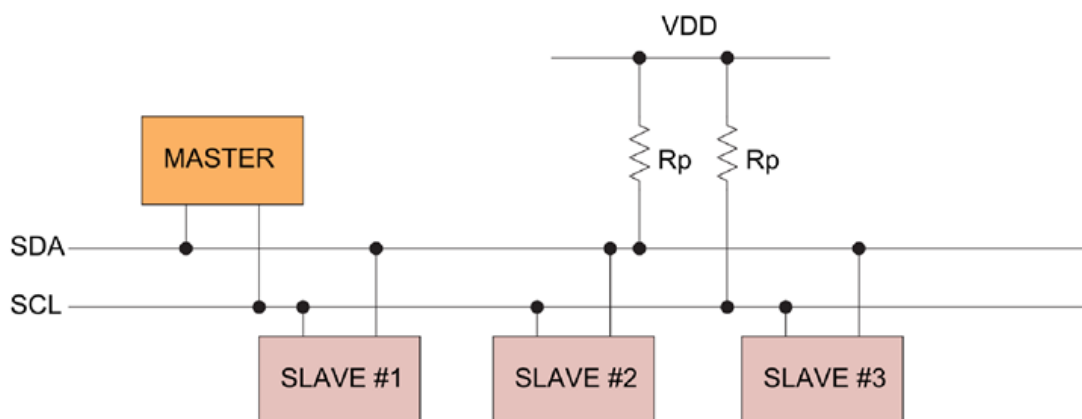


Рисунок 2.7 – Узагальнена схема підключення I²C

Передача даних відбувається за шляхом перемикавання логічних рівнів «0» та «1». Біти тактовані по спадаючим фронтам імпульсів. Стандартна швидкість передачі даних – 100 кбіт/с, швидкий режим – 400 кбіт/с. Шина I²C може підтримувати як декілька ведучих пристроїв так і декілька ведених. Обмеженнями є адресний простір (128 унікальних адрес) та ємність шини (400 пФ).

2.3. Конфігурування та комутація мікроконтролерів та модулів системи

Почнемо з налаштувань центрального мікроконтролера хаба. Сконфігуруємо 2 інтерфейса USART, один для з'єднання з ПК, другий для підключення модуля Wi-Fi. Швидкість передачі даних виставимо 115200 біт/с, режим асинхронний (рис. 2.9). Також ввімкнемо глобальне переривання UART (рис. 2.8). Налаштування для обох інтерфейсів аналогічні, за виключенням виходів мікроконтролера, на котрих працюватимуть інтерфейси.

✔ NVIC Settings	✔ DMA Settings		✔ GPIO Settings
✔ Parameter Settings		✔ User Constants	
NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
USART1 global interrupt	☒	0	0

Рисунок 2.8 – Увімкнення глобального переривання UART

Рисунок 2.9 – Налаштування параметрів UART

Для підключення радіомодуля сконфігуруємо інтерфейс SPI (рис. 2.10).

Рисунок 2.10 – Налаштування SPI

Також, налаштуємо можливість відлагодження (рис. 2.11) та ввімкнемо керамічний резонатор на платі (рис. 2.12).

Рисунок 2.11 – Налаштування відлагодження плати

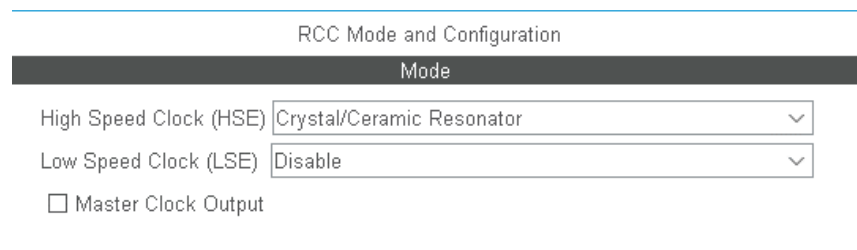


Рисунок 2.12 – Ввімкнення кварцового резонатору на платі

Для підключення пристроїв автоматизації дротовим каналом зв'язку – налаштуємо протокол I²C (рис. 2.13).

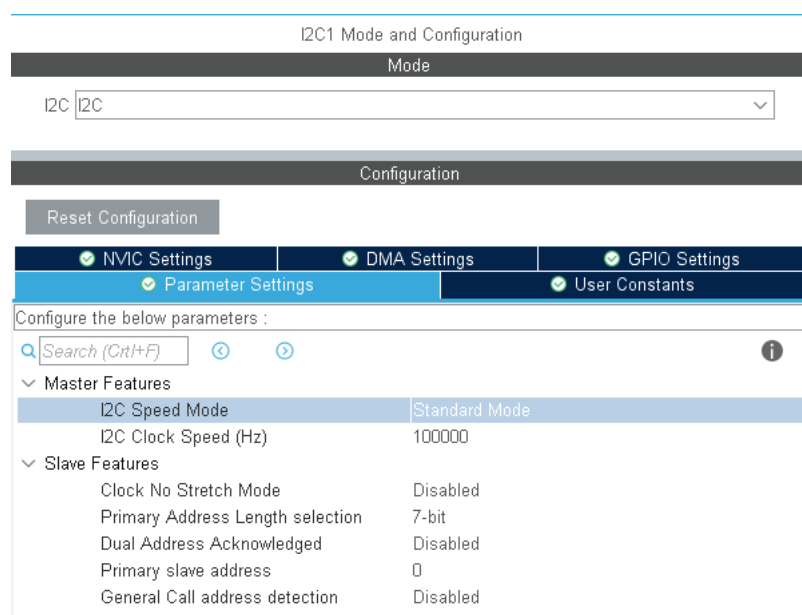


Рисунок 2.13 – Налаштування I²C

Також, налаштуємо один пін на вхід «INPUT» та один пін на вихід «LED», до якого підключений світлодіод (рис. 2.14).

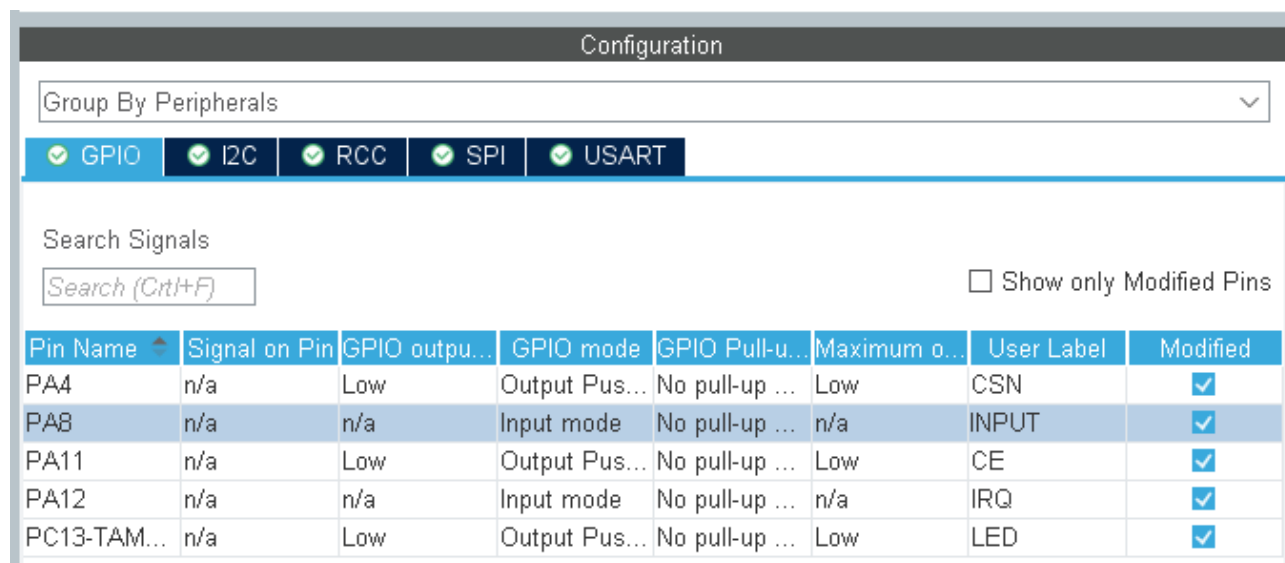


Рисунок 2.14 – Налаштування портів входів/виходів

Можемо бачити, що всі, відповідні протоколам, виходи мікроконтролера були сконфігуровані (рис. 2.15).

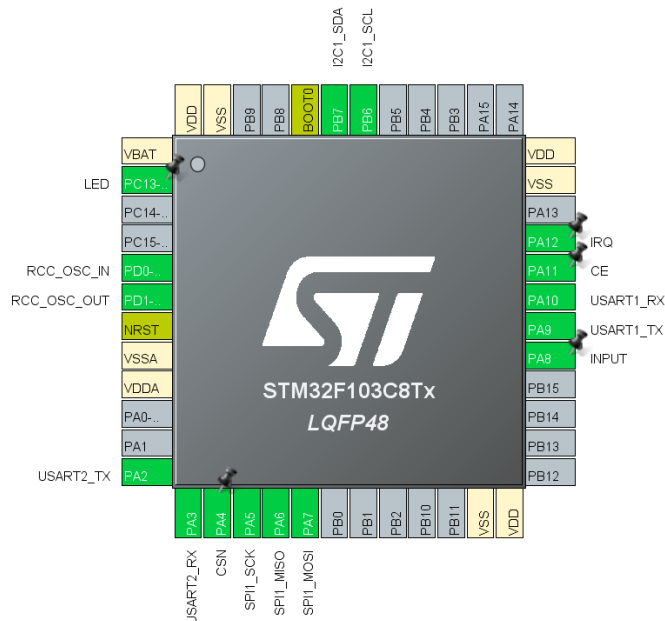


Рисунок 2.15 – Сконфігуровані виходи мікроконтролера

Через множник та кварцевий резонатор сконфігуруємо максимальну частоту мікроконтролера – 72 МГц (рис. 2.16).

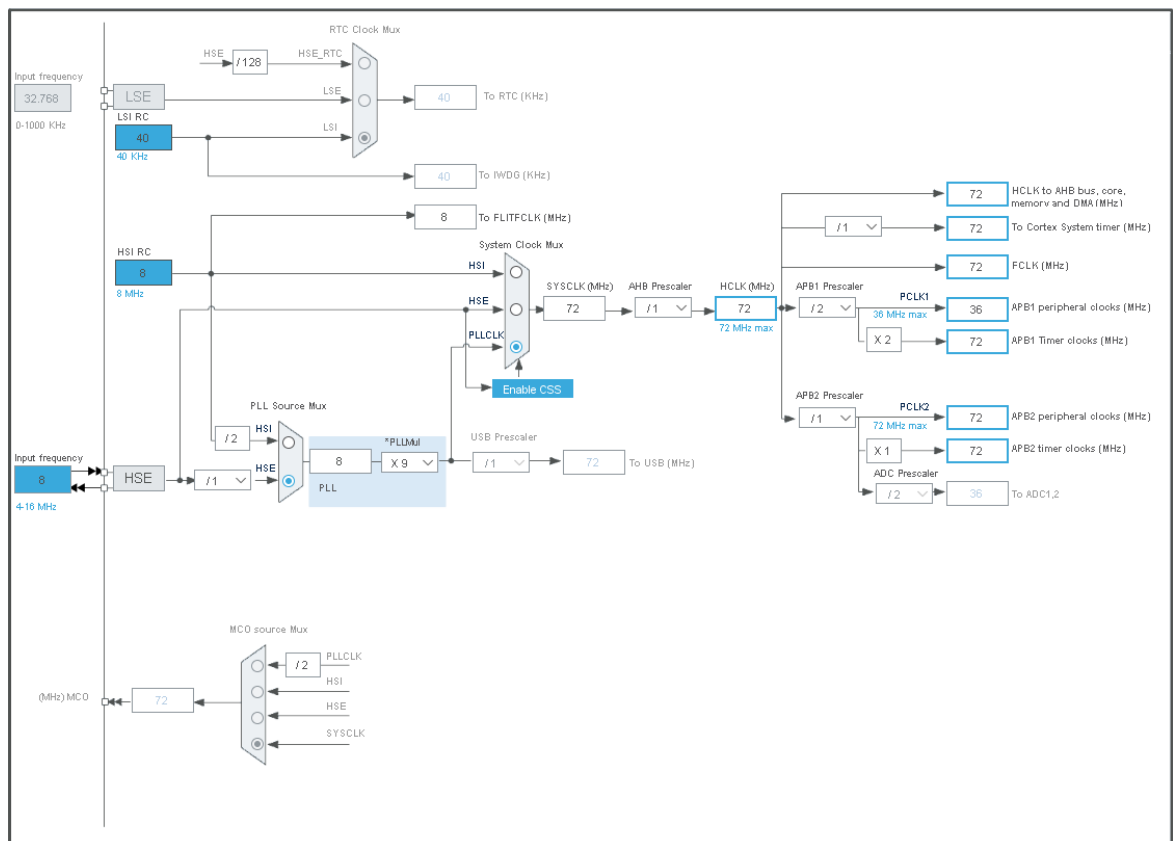


Рисунок 2.16 – Сконфігурована тактова частота мікроконтролера

Для використання плати «Arduino Nano» в якості UART-TTL конвертора – потрібно пін «RST» замкнути на землю. Після чого, для підключення до STM32 потрібно піни Arduino Nano «RX0» та «TX1» з'єднати з пінами STM32 «A3» та «A2» відповідно (рис. 2.17).

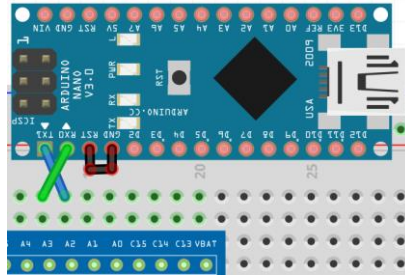


Рисунок 2.17 – Підключення UART-TTL конвертора до центрального мікроконтролера хабу

Далі під'єднаємо радіомодуль «nrf24l01+» (рис. 2.18). Піни STM32 з'єднаємо з такими пінами радіомодуля:

- A4 – CSN;
- A5 – SCK;
- A6 – MISO;
- A7 – MOSI;
- A11 – CE;
- A12 – IRQ.

Для радіомодуля дуже важливе стабільне живлення 3,3 В, тому під'єднаємо його до зовнішнього джерела живлення.

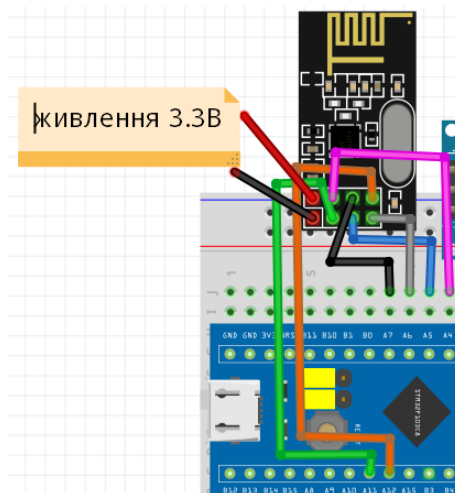


Рисунок 2.18 – Під'єднання до хабу радіомодуля

Під'єднаємо модуль Wi-Fi «ESP8266-01S» до центрального мікроконтролера хабу (рис. 2.19). Для цього з'єднаємо пін STM32 з такими пінами ESP8266:

- A9 – RX;
- A10 – TX.

До модуля Wi-Fi підключимо живлення від STM32. Також, пін модуля «EN» з'єднаємо з 3.3 В.

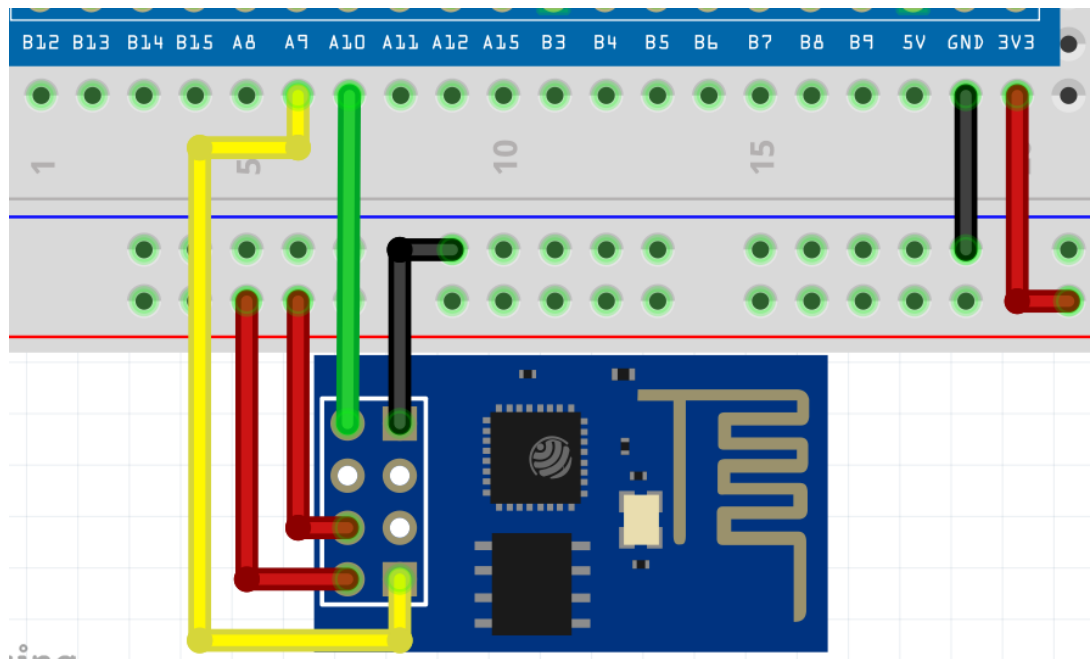


Рисунок 2.19 – Підключення до хабу ESP8266-01S

Наразі, хаб зібраний і залишилось скомутувати пристрої автоматизації. Спочатку підключимо радіомодуль до Arduino Nano (рис. 2.20), для цього потрібно з'єднати пін Arduino Nano з такими пінами nrf24l01+:

- D9 – CE;
- D10 – CSN;
- D11 – MOSI;
- D12 – MISO;
- D13 – SCK.

Як і у випадку з хабом, потрібно підключити зовнішнє джерело живлення на 3.3 В.

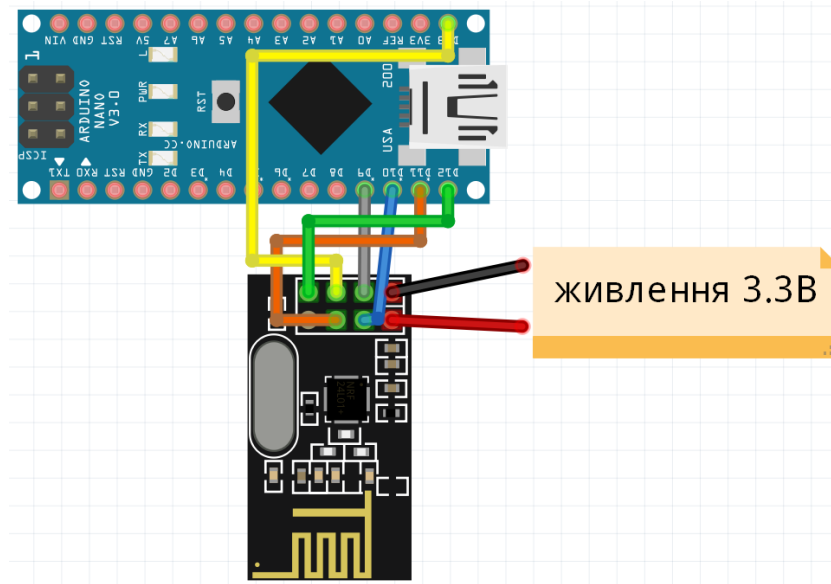


Рисунок 2.20 – Підключення nrf24l01+ до Arduino Nano

Підключимо Arduino Nano до STM32 за допомогою шини I2C (рис. 2.21).

Для цього з'єднаємо піни Arduino Nano з такими пінами STM32:

- A4 – B7 (SDA);
- A5 – B6 (SCL).

Додатково підтягнемо лінії до 5 В за допомогою резисторів з опором 10 кОм. Також, для коректної роботи, з'єднаємо землі Arduino Nano та STM32.

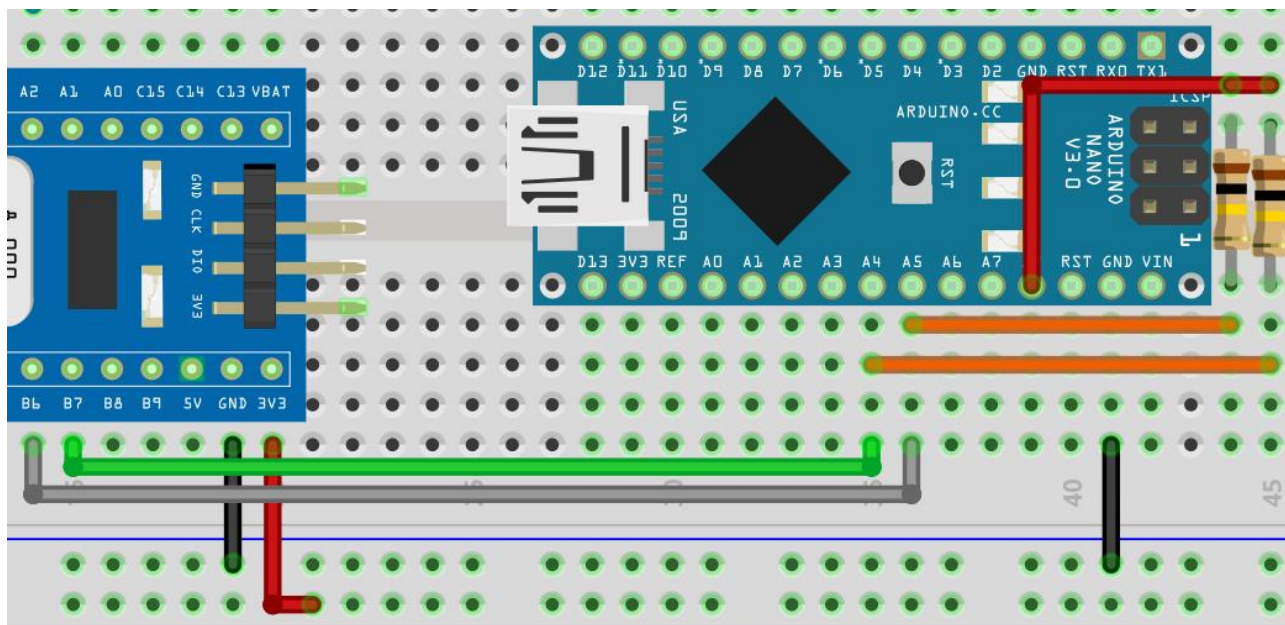


Рисунок 2.21 – Підключення Arduino Nano до хабу за допомогою I²C

Повну схему комутації системи наведено в додатку А.

2.4. Розробка програмного забезпечення системи управління розумним будинком

Розпочнемо реалізацію функціоналу з настільного додатка для налаштування підключення до Wi-Fi мережі. Для цього в Visual Studio створимо WPF. Для початку потрібно вивести для вибору список усіх доступних COM-портів (рис. 2.22).

```
public void RefreshAvailablePortNames()
{
    AvailablePortNames = new ObservableCollection<string>(SerialPort.GetPortNames());
    if (AvailablePortNames.Any())
    {
        SelectedPortIndex = 0;
    }
}
```

Рисунок 2.22 – Код для виведення усіх доступних COM-портів

Створимо функцію, в якій буде відкриватися обраний порт, проводитися його налаштування та відправлятимуться введені дані (рис. 2.23).

```
public void Configure()
{
    SerialPort serialPort = new SerialPort(AvailablePortNames[SelectedPortIndex]);
    serialPort.BaudRate = 115200;
    serialPort.Parity = Parity.None;
    serialPort.DataBits = 8;
    serialPort.StopBits = StopBits.One;

    serialPort.WriteTimeout = 1000;

    try
    {
        serialPort.Open();
        serialPort.WriteLine("<" + Ssid + "<" + Password + "<");
        serialPort.Close();

        MessageBox.Show("Done!");
    }
    catch (Exception ex)
    {
        MessageBox.Show("aaa, error!\n" + ex.Message);
    }
}
```

Рисунок 2.23 – Код для відправки введених даних мікроконтролеру

Тепер треба обробити відправлені дані в мікроконтролері.

Після конфігурації мікроконтролера «STM32F103C8» в додатку

«CubeMX», був створений пустий проект. Для програмування STM32, підключимо її до комп'ютера за допомогою Плата прошиватиметься за допомогою програматора «ST-LINK V2» (рис. 2.24) та утиліти «STM32CubeProgrammer».



Рисунок 2.24 – Зовнішній вигляд програматора «ST-LINK V2»

Напишемо код для обробника переривань, що спрацювуватимуть при отриманні даних через UART (рис. 2.25). Він, після прийняття даних, буде розпарсувати їх на окремо SSID і пароль, передаватиме для перевірки дані до комп'ютеру та відправлятиме їх по UART в ESP8266.

```
void HAL_UART_IDLE_Callback(UART_HandleTypeDef *huart) {
    if(huart == &huart2) {

        UART_PC_Parse_settings(pc_rx_buffer, ssid, password);

        sprintf(str1, "SSID: %s \r\n", ssid);
        HAL_UART_Transmit(&huart2, (uint8_t*)str1, strlen(str1), 0x1000);

        sprintf(str1, "password: %s \r\n", password);
        HAL_UART_Transmit(&huart2, (uint8_t*)str1, strlen(str1), 0x1000);

        UART_MOTT_Send_settings_float(ssid, password);
    }
}
```

Рисунок 2.25 – Обробник вхідних даних для налаштування мережі

Для програмування ESP8266-01S, потрібно підключити її до комп'ютера за допомогою UART-TTL конвертера. При цьому, пін «GPIO0» підтягнути до землі (рис. 2.26).

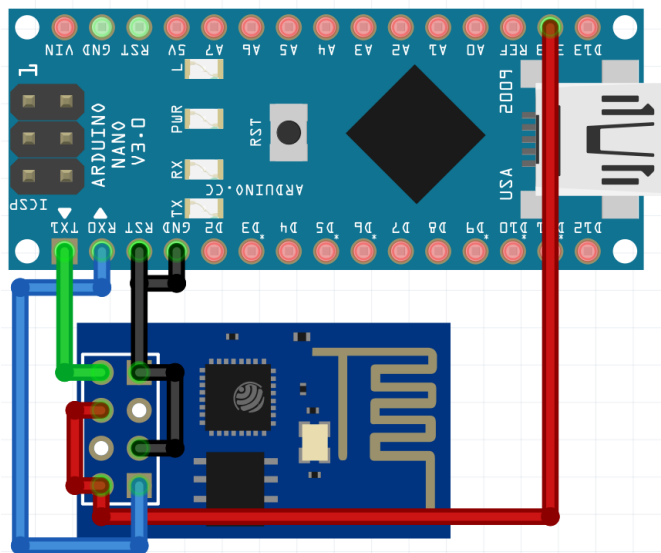


Рисунок 2.26 – Підключення ESP8266 для програмування

В ESP8266 потрібно конфігураційні дані прийняти, підключитись до мережі та зберегти її в енергонезалежну пам'ять EEPROM. Для цього створимо структуру, поля якої – це дані конфігурації (SSID чи пароль) та їх адреса в EEPROM (рис. 2.27).

```
struct string_eeprom{
    String val;
    int addr;
};

string_eeprom saved_ssid = {ssid, 0};
string_eeprom saved_password = {pass, saved_ssid.addr+sizeof(String)};
```

Рисунок 2.27 – Структура для збереження SSID та паролю мережі

Коли приходитимуть дані по UART, перевірятьимемо по сигнатурі, чи це конфігураційні дані (рис. 2.28).

```
void uart_parsing(void) {
    if (Serial.available() > 0) {
        Serial.readBytes(esp_rx_buffer, 9);

        if (esp_rx_buffer[0] == 0xFD) { // перевірка, чи прийшов SSID
            Serial.readBytes(esp_rx_buffer, configuration_size);
            memcpy(esp_rx_buffer, ssid, sizeof(esp_rx_buffer));

            char *temp = new char[sizeof(ssid)];
            for (int i = 0; i < sizeof(ssid); i++) temp[i]=i+1;
            for (int i = 0; i < sizeof(ssid) -1 ; i++) ssid[i]=temp[i+1];
            delete[] temp;
        }
    }
}
```

Рисунок 2.28 – Перевірка чи прийняті конфігураційні дані

Коли прийдуть нові SSID та пароль, потрібно їх записати в енергонезалежну пам'ять EEPROM та перезапустити мережу (рис. 2.29).

```

if (esp_rx_buffer[0] == 0xFE) { // перевірка, чи прийшов пароль
    Serial.readBytes(esp_rx_buffer, configuration_size);
    memcpy(esp_rx_buffer, pass, sizeof(esp_rx_buffer));

    char *temp = new char[sizeof(pass)];
    for (int i = 0; i < sizeof(pass); i++) temp[i]=i+1;
    for (int i = 0; i < sizeof(pass) -1 ; i++) pass[i]=temp[i+1];
    delete[] temp;

    EEPROM.begin(configuration_size * 2);
    EEPROM.put(saved_ssid.addr, saved_ssid.val);
    EEPROM.put(saved_password.addr, saved_password.val);
    EEPROM.commit();
    EEPROM.end();
    delay(500);

    WiFi.disconnect();
    wifi_not_ok();
    WiFi.begin(ssid, pass);

    if (WiFi.waitForConnectResult() != WL_CONNECTED) {
        wifi_ok();
    }
}

```

Рисунок 2.29 – Запис даних в EEPROM та перезапуск мережі

Також, при включенні модуля Wi-Fi, потрібно перевірити, чи записані в EEPROM дані мережі та примініти їх (рис. 2.30).

```

EEPROM.begin(configuration_size * 2);
delay(500);
EEPROM.get(saved_ssid.addr, saved_ssid.val);
EEPROM.get(saved_password.addr, saved_password.val);
EEPROM.end();

```

Рисунок 2.30 – Читання даних з енергонезалежної пам'яті

Реалізуємо роботу протоколу передачі даних в системі. Для початку, створимо функції, що дозволять формувати пакет та відправляти його за допомогою UART (рис. 2.31).

```

void Protocol_Send_bool(uint8_t signature, bool value, uint8_t *tx_buffer);
void Protocol_Send_int8_t(uint8_t signature, int8_t value, uint8_t *tx_buffer);
void Protocol_Send_uint8_t(uint8_t signature, int8_t value, uint8_t *tx_buffer);
void Protocol_Send_int16_t(uint8_t signature, int16_t value, uint8_t *tx_buffer);
void Protocol_Send_uint16_t(uint8_t signature, uint16_t value, uint8_t *tx_buffer);
void Protocol_Send_int32_t(uint8_t signature, int32_t value, uint8_t *tx_buffer);
void Protocol_Send_uint32_t(uint8_t signature, uint32_t value, uint8_t *tx_buffer);
void Protocol_Send_float(uint8_t signature, float value, uint8_t *tx_buffer);
bool Protocol_Receive_bool(uint8_t *rx_buffer);

```

Рисунок 2.31

Розглянемо одну таку функцію більш детально (рис. 2.32).

```
void Protocol_Send_bool(uint8_t signature, bool value, uint8_t *tx_buffer)
{
    tx_buffer[0] = t_signature;
    tx_buffer[1] = signature;
    tx_buffer[2] = t_bool;
    tx_buffer[3] = 0x0;
    tx_buffer[4] = 0x0;
    tx_buffer[5] = 0x0;
    tx_buffer[6] = value;
    uint16_t crc = Calculate_Checksum(tx_buffer);
    tx_buffer[7] = crc >> 8;
    tx_buffer[8] = crc;

    Serial.write(tx_buffer, 9);
}
```

Рисунок 2.32 – Код функції для генерації пакету з відправкою даних типу
«bool»

Створимо такі ж функції, але для отримання даних різних типів з пакету (рис. 2.33).

```
int8_t Protocol_Receive_int8_t(uint8_t *rx_buffer);
uint8_t Protocol_Receive_uint8_t(uint8_t *rx_buffer);
int16_t Protocol_Receive_int16_t(uint8_t *rx_buffer);
uint16_t Protocol_Receive_uint16_t(uint8_t *rx_buffer);
int32_t Protocol_Receive_int32_t(uint8_t *rx_buffer);
uint32_t Protocol_Receive_uint32_t(uint8_t *rx_buffer);
```

Рисунок 2.33 – Список функцій для отримання даних з пакету

Останній крок написання програмного коду для роботи з пакетами – розробити функцію для підрахування контрольної суми (рис 2.34) та для її перевірки в отриманому пакеті (рис. 2.35).

```
uint16_t Calculate_Checksum(uint8_t *tx_buffer) {
    return (~(tx_buffer[1] +
    tx_buffer[2] +
    tx_buffer[3] +
    tx_buffer[4] +
    tx_buffer[5] +
    tx_buffer[6])) + 1;
}
```

Рисунок 2.34 – Функція для обрахунку контрольної суми

```

bool Protocol_Checksum_validation(uint8_t *rx_buffer)
{
    bool result;
    uint16_t crc_rx_buffer = 0x0000;
    uint16_t crc_check = 0x0000;
    crc_rx_buffer = Calculate_Checksum(rx_buffer);
    crc_check = (crc_check | rx_buffer[7] << 8) | (crc_check | rx_buffer[8]);

    if (crc_rx_buffer == crc_check)
    {
        result = true;
    }
    else
    {
        result = false;
    }
    return result;
}

```

Рисунок 2.35 – Функція для перевірки правильності контрольної суми

Створимо MQTT-брокер для нашої системи управління розумним домом. Для цього скористаємося послугами хмарного брокера під назвою «CloudMQTT». CloudMQTT дозволяє легко створювати та масштабувати системи обміну повідомленнями. Також, його гарними рисами є інтуїтивно зрозумілий користувальницький інтерфейс, спрощене адміністрування та можливість сплачувати за його послуги лише за потрібні функції [16] .

Створимо новий брокер MQTT (рис. 2.36).

Рисунок 2.36 – Створення нового MQTT-брокера

Після створення з'явиться вікно з деталями конфігурації брокера (2.37).
Перенесемо ці дані в прошивку ESP8266 (2.38).

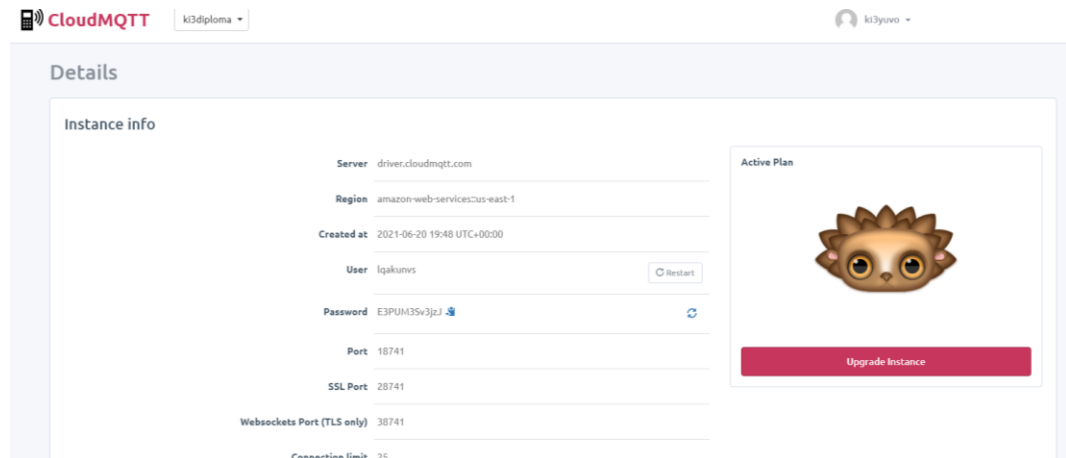


Рисунок 2.37 – Деталі конфігурації MQTT-брокера

```
const char *mqtt_server = "driver.cloudmqtt.com";
const int mqtt_port = 18741;
const char *mqtt_user = "lqakunvs";
const char *mqtt_pass = "E3PUM3Sv3jzJ";
```

Рисунок 2.38 – Дані для підключення до MQTT-брокера

Для початку, напишемо код, за допомогою якого підключимось до Wi-Fi.
Якщо підключення успішне – підключимось до MQTT-брокера та підпишемося на потрібні топіки (рис. 2.39).

```
if (WiFi.status() != WL_CONNECTED)
{
    Send_wifi_status(false);
    WiFi.begin(ssid, pass);

    if (WiFi.waitForConnectResult() != WL_CONNECTED)
    {
        return;
    }
    Send_wifi_status(true);
}

if (WiFi.status() == WL_CONNECTED)
{
    if (!client.connected())
    {
        Send_mqtt_status(false);
        if (client.connect(MQTT::Connect(mqtt_name).set_auth(mqtt_user, mqtt_pass)))
        {
            Send_mqtt_status(true);
            client.set_callback(callback);

            client.subscribe("diploma/led");
            client.subscribe("diploma/analog");
            client.subscribe("diploma/color");
        }
    }
}
```

Рисунок 2.39 – Підключення до Wi-Fi та MQTT-брокера

Повний лістинг прошивки для ESP8266-01S наведений в додатку Б.

Перейдемо до реалізації радіозв'язку. Для початку, аби обрати канал, на якому працюватиме радіозв'язок, перевіримо загальну завантаженість каналів

частотного діапазону смуги 2.4 ГГц. Напишемо простий код, що скануватиме кожен із 128 каналів діапазону, та виведемо в монітор порта рядок зі значеннями завантаженості – від «0» (найкраще) до «f» (найгірше) (рис. 2.40).

```
void loop(void)
{
    memset(values,0,sizeof(values));
    int rep_counter = num_reps;
    while (rep_counter--) {
        int i = num_channels;
        while (i--) {
            radio.setChannel(i);
            radio.startListening();
            delayMicroseconds(128);
            radio.stopListening();
            if (radio.testCarrier())
                ++values[i];
        }
    }
}
```

Рисунок 2.40 – Код для перевірки завантаженості каналів

Відкриємо монітор COM-порта та перевіримо результат сканування (рис. 2.41). Для зручності, представимо їх в виді графіку (рис. 2.42).

[illegible]

Рисунок 2.41 – Результат сканування завантаженості каналів

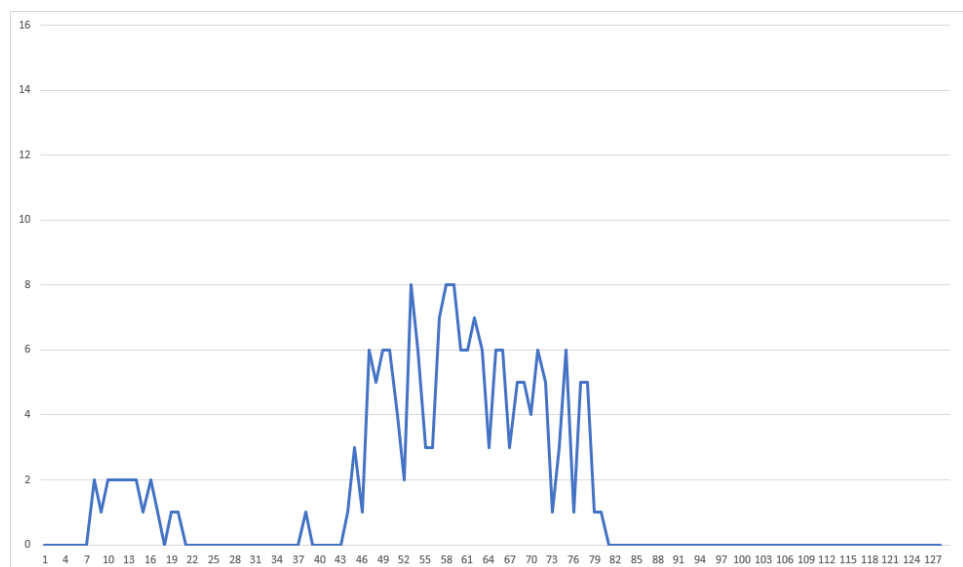


Рисунок 2.42 – Графік завантаженості каналів частотного діапазону

Зважаючи на те, що кінець смуги частотного діапазону завантаження немає, оберемо звідти канал, на якому працюватиме радіозв'язок нашої системи – 96 («0x60»). Таким чином, радіозв'язок працюватиме на частоті приблизно 2.496 ГГц.

Для роботи радіомодуля «nRF24L01+» з STM32, скористаємося бібліотекою, за основу якої була взята бібліотка «RF24» платформи «Arduino»[17]. Це важливо тому, що, аби радіозв'язок працював коректно, всі налаштування радіомодуля в хабі та в пристрої автоматизації мають співпадати.

Налаштуємо радіомодуль хаба в якості передавача (рис. 2.43). Перепишемо функцію «NRF_Init» бібліотеки «RF24» так, аби всі її параметри співпадали з аналогічними в пристрої автоматизації (рис. 2.44).

```
uint8_t res = isChipConnected();

snprintf(str1, 64, "Connected: %s\n", res ? "OK" : "NOT OK");
HAL_UART_Transmit(&huart2, (uint8_t*)str1, strlen(str1), 1000);

res = NRF_Init();

snprintf(str1, 64, "Init: %s\n", res > 0 && res < 255 ? "OK" : "NOT OK");
HAL_UART_Transmit(&huart2, (uint8_t*)str1, strlen(str1), 1000);

enableAckPayload();
setChannel(0x60);
openWritingPipe(pipe0);
```

Рисунок 2.43 – Налаштування радіомодуля в якості передавача

```
uint8_t NRF_Init(void)
{
    uint8_t setup = 0;
    p_variant = false;
    payload_size = 32;
    dynamic_payloads_enabled = false;
    addr_width = 5;
    pipe0_reading_address[0] = 0x787878787878;

    ce(LOW);
    csn(HIGH);
    HAL_Delay(5);

    write_register(NRF_CONFIG, 0x0C);
    setRetries(5, 15);
    setPALevel(RF24_PA_MAX);

    setDataRate(RF24_250KBPS);

    setup = read_register(RF_SETUP);

    toggle_features();
    write_register(FEATURE, 0);
    write_register(DVLPD, 0);
    dynamic_payloads_enabled = false;

    write_register(NRF_STATUS, (1 << RX_DR) | (1 << TX_DS) | (1 << MAX_RT));
    setChannel(0x60);
    flush_rx();
    flush_tx();
    powerUp();
    write_register(NRF_CONFIG, (read_register(NRF_CONFIG) & ~(1 << PRIM_RX));
    return (setup != 0 && setup != 0xff);
}
```

Рисунок 2.44 – Функція ініціалізації радіомодуля

Налаштуємо аналогічну ініціалізацію радіомодуля для Arduino Nano та

запустимо його в якості приймача (рис. 2.45).

```
void setup()
{
  Serial.begin(9600);

  radio.begin();
  radio.setPayloadSize(32);
  radio.setAddressWidth(5);
  radio.setAutoAck(1);
  radio.setRetries(5, 15);
  radio.enableAckPayload();
  radio.setPALevel(RF24_PA_MAX);
  radio.setDataRate(RF24_250KBPS);
  radio.openReadingPipe(1, pipe0);
  radio.setChannel(0x60);
  radio.powerUp();

  Serial.print("Started listening: ");
  radio.startListening();
}
```

Рисунок 2.45 – Ініціалізація та налаштування радіомодуля в якості приймача

Для роботи з шиною I²C, скористаємося бібліотекою «Wired». Хаб виступатиме в якості ведучого, а пристрій автоматизації – в якості веденого, тому при його ініціалізації вказувати його адрес не потрібно. При отриманні даних, що стосуються даного пристрою автоматизації (топик «diploma/led»), від MQTT-брокера, відправимо їх через шину I²C (рис. 2.46).

```
switch (signature) {
  case (0x01):
    if (UART_MQTT_Checksum_validation(esp_rx_buffer)) {
      led = UART_MQTT_Receive_data_bool(esp_rx_buffer);

      Wire.beginTransaction(8);
      Wire.write(led);
      Wire.endTransmission();
    }
}
```

Рисунок 2.46 – Відправлення даних шиною I²C

Налаштуємо прийом даних для пристрою автоматизації, оснований на мікроконтролері «ATmega328P». Важливо зауважити, що при ініціалізації шини потрібно вказати адресу пристрою, яку буде використовувати хаб для відправлення даних (рис. 2. 47).

```
void setup()
{
  Serial.begin(9600);

  Wire.begin(8);
  Wire.onReceive(receive_handler);
}
```

Рисунок 2.47 – Ініціалізація шини I²C

Напишемо обробник переривань, котрий буде спрацьовувати при

отриманні даних по шині (рис. 2.48).

```
void receive_handler(int count)
{
    byte value = Wire.read();

    if (value == 1)
    {
        digitalWrite(LED_BUILTIN, HIGH);
        Serial.println("LED on");
    }
    else
    {
        digitalWrite(LED_BUILTIN, LOW);
        Serial.println("LED off");
    }

    delay(500);
}
```

Рисунок 2.48 – Обробник прийому даних з шини I²C

Висновки до розділу 2

В даному розділі була побудована структурна схема системи, що, узагальнюючи, складається з: хабу, трьох пристроїв автоматизації та пристроїв для керування системою. Пристрої автоматизації можуть під'єднуватись дротовим способом, радіозв'язком або мережею Інтернет.

Були описані протоколи та інтерфейси для взаємодії модулів системи, такі як: «UART», «MQTT», «SPI», «ESB», «I²C». Також, був розроблений власний протокол, який працює поверх протоколу «UART» та слугує для передачі даних між хабом та Wi-Fi-модулем.

Далі була виконана конфігурація мікроконтролерів та модулів системи, після чого, ті були комутовані, згідно структурної схеми системи.

В останньому пункті було розроблено програмне забезпечення, яким було запрограмовані мікроконтролери та модулі системи. Також, був створений та інтегрований в систему управління розумним будинком MQTT-брокер.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ УПРАВЛІННЯ

3.1. Порядок підключення та налаштування системи. Структура інтерфейсу

При першому ввімкненні системи потрібно ввести SSID та пароль Wi-Fi мережі, до якої користувач хоче під'єднатися. Зробити це можна за допомогою розробленого раніше додатка для операційної системи «Windows». Він не потребує ніяких вимог від системи, окрім встановленого «.NET Framework». Інтерфейс додатка простий та інтуїтивно зрозумілий: є поле для вибору COM-порта, кнопка для оновлення списку відкритих COM-портів, поля для введення даних та кнопка, за допомогою якої відбувається конфігурація (рис. 3.1). Після натискання кнопки «Configure» дані про мережу будуть відправлені до хабу, котрий перешле їх в Wi-Fi модуль, де вони будуть збережені в енергонезалежну пам'ять.



Рисунок 3.1 – Інтерфейс додатка для підключення до мережі Wi-Fi

При вдалому підключенні буде показано повідомлення з відповідним текстом (рис. 3.2).

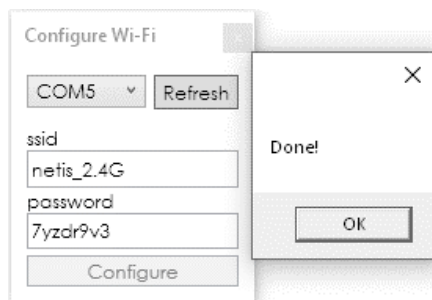


Рисунок 3.2 – Вдале налаштування мережі

У разі, якщо трапилась помилка, наприклад, обраний порт було закрито, або якщо порт зайнятий – при спробі передати дані, буде показано вікно з текстом помилки (рис. 3.3).

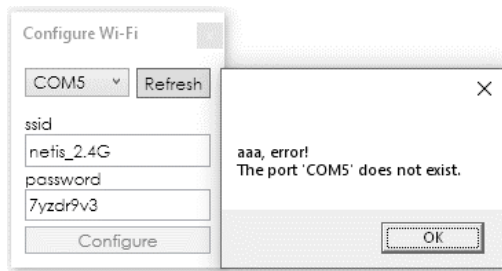


Рисунок 3.3 – Виведення повідомлення про помилку

3.2. Можливості роботи з системою за допомогою стороннього програмного забезпечення

Завдяки тому факту, що система взаємодіє з мережею Інтернет за допомогою віддаленого MQTT-брокера, є можливість керувати нею за допомогою стороннього програмного забезпечення. Прикладом такого є додаток «MQTT Dash» для операційної системи «Android».

Підключимо MQTT-брокер в додатку. Для цього, спочатку введемо адресу та порт брокера (рис. 3.4), потім ім'я користувача, пароль та унікальний ідентифікатор клієнта (рис. 3.5).

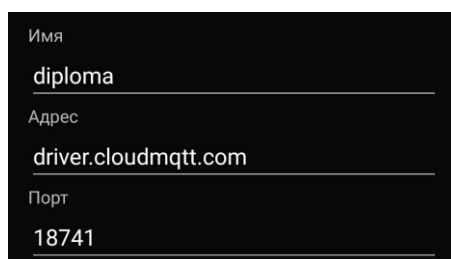


Рисунок 3.4 – Введення адреси та порта MQTT-брокера

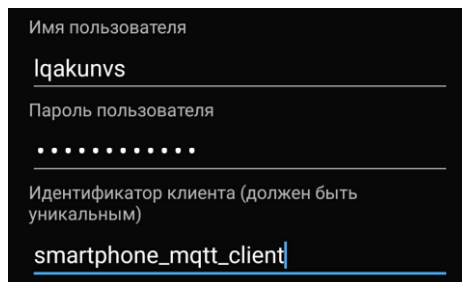


Рисунок 3.5 – Введення ім'я користувача, паролю та ідентифікатору

Після введення конфігураційних даних, підключимось до брокера. Наявність підключення перевіримо в адміністративній панелі сайту брокера (рис 3.6).

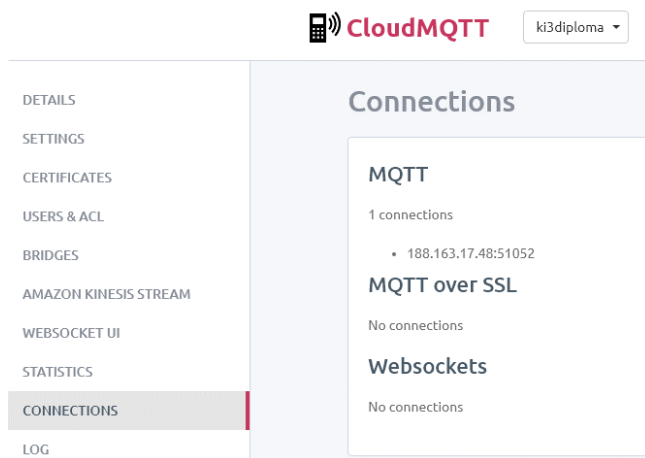


Рисунок 3.6 – Наявність підключення смартфона до MQTT-брокера

Додамо віджет для управління світлодіодом. Для цього потрібно ввести відповідний топик та налаштувати відображення віджету (рис. 3.6).

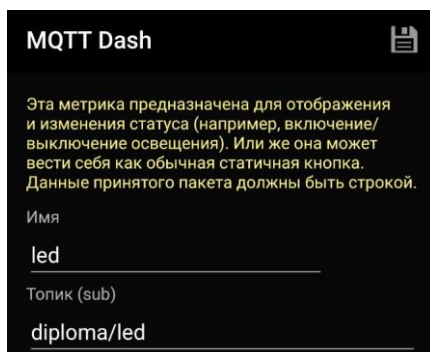


Рисунок 3.6 – Створення віджету для управління світлодіодом

Додамо віджет для управління значенням від 0 до 1023 (рис. 3.7).



Рисунок 3.7 – Створення віджету для управління значенням від 0 до 1023

Додамо віджет для управління кольором (рис. 3.8).

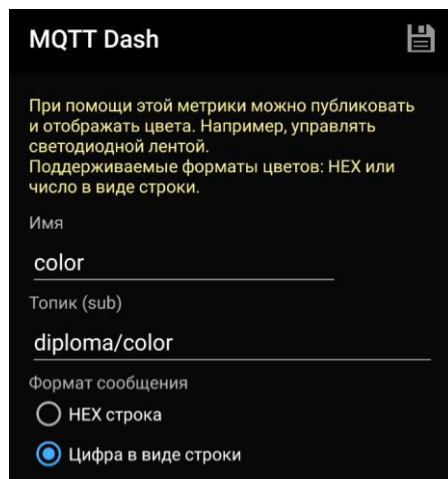


Рисунок 3.8 – Віджет для управління кольором

Після підключення топіків та налаштування віджетів, можна зручно управляти та моніторити пристрої, що підключені до MQTT-сервера (рис. 3.9).

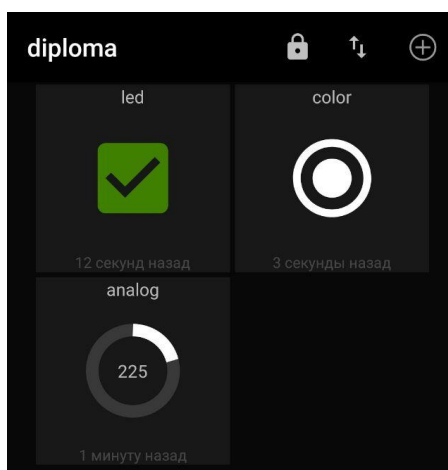


Рисунок 3.9 – Інтерфейс з підключеними віджетами

Перевіримо роботу додатка з підключеними топіками. Для цього змінимо значення віджетів та перевіримо, чи прийшли повідомлення на MQTT-сервер. Можемо переконатися, що все працює (рис. 3.10).

Received messages		▼
Topic	Message	
diploma/led	0	
diploma/led	1	
diploma/color	11366286	
diploma/analog	1023	

Рисунок 3.10 – Отримані повідомлення MQTT-сервером

При сценарії, коли інший клієнт відправляє повідомлення з відповідними топіками – дані в додатку оновлюються правильно.

Загалом, завдяки тому, що протокол MQTT досить популярний, існує багато іншого програмного забезпечення, що може задовольнити багато побажань користувачів. Так, наприклад, за допомогою платформи «Dialogflow» можна синхронізувати пристрої, підключені до MQTT-брокера з екосистемою «Google Home» та налаштувати голосове управління ними.

3.3. Тестування роботи системи

Для початку, перевіримо підключення модуля Wi-Fi до домашньої мережі. Для цього увімкнемо хаб та введемо SSID та пароль мережі. Після передачі даних в адміністративній панелі роутера з'явиться запис про клієнт ESP (рис. 3.11).

Список беспроводных клиентов						
ID	SSID	имя узла	MAC-адрес	RSSI	Передано пакетов	получено пакетов
1	netis_2.4G	ESP-654F46	50:02:91:65:4f:46	75	14	49

Рисунок 3.11 – ESP8266 під'єднався до Wi-Fi мережі

Переглянемо IP-адресу (рис. 3.12) та перевіримо зв'язок з клієнтом (рис. 3.13).

Список клиентов DHCP					
ID	IP-адрес	MAC-адрес	имя узла	Зарезервов.	Операция
1	192.168.1.107	50:02:91:65:4f:46	ESP-654F46	Динамический	

Рисунок 3.12 – IP-адреса клієнта

```
C:\Users\Meow>ping 192.168.1.107

Pinging 192.168.1.107 with 32 bytes of data:
Reply from 192.168.1.107: bytes=32 time=64ms TTL=255
Reply from 192.168.1.107: bytes=32 time=83ms TTL=255
Reply from 192.168.1.107: bytes=32 time=95ms TTL=255
Reply from 192.168.1.107: bytes=32 time=107ms TTL=255

Ping statistics for 192.168.1.107:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 64ms, Maximum = 107ms, Average = 87ms
```

Рисунок 3.13 – Перевірка зв'язку з клієнтом

Перезапустимо модуль ESP8266, але під'єднавши його замість хаба до комп'ютера за допомогою UART-TTL конвертера. Відкривши, перед включенням модуля, COM-порт, можемо спостерігати пакети, які передає

модуль. Перевіримо, чи підключився модуль до MQTT-брокера. Для цього в панелі адміністрування перейдемо на відповідну вкладку. Можемо спостерігати, що підключено 2 пристрої (рис. 3.14).

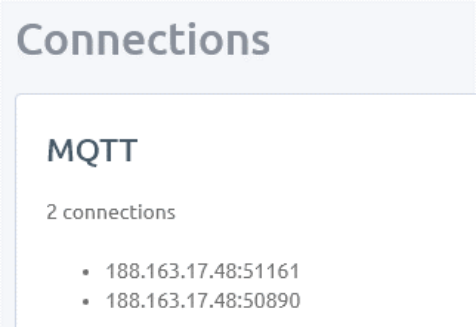


Рисунок 3.14 – До MQTT-брокера підключено 2 пристрої

Перший – це смартфон, другий модуль ESP8266. В додатку на смартфоні переключимо світлодіод в стан «увімкнено». Після відправки повідомлення, в СОМ-порті можемо спостерігати, що були отримані деякі дані (рис. 3.15). Це пакет, згідно протоколу, розробленого раніше, в котрому можна виділити сигнатуру, тип даних, дані та контрольну суму.

```
000000f0: ff dd dd dd dd fa 8e ff
00000100: ff 01 01 00 00 00 01 ff
```

Рисунок 3.15 – Дані з СОМ-порту, де перший рядок – повідомлення про підключення до MQTT-брокеру, другий – пакет з даними

Повернемо під’єднання ESP8266 до хабу. При отриманні повідомлення з сигнатурою «0x01» (що відповідає топіку «diploma/led»), центральний мікроконтролер хаба відправляє результат всім пристроям автоматизації. Таким чином можна перевірити зв’язок по радіоканалу та зв’язок по шині I²C. Для цього підключимо їх через UART до комп’ютера, відкриємо програму «Advanced Serial Port Terminal» та підключимось до відповідного СОМ-порту. Після цього, відкриємо на смартфоні додаток «MQTT Dash» та помигаємо світлодіодом. Можемо спостерігати, що по шині I²C дані коректно було отримано (рис. 3.16).

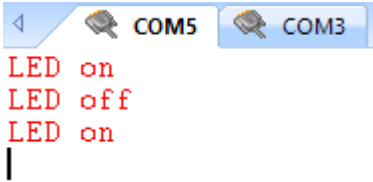


Рисунок 3.16 – Отримані дані по шині I²C

Через радіозв'язок дані також було отримано коректно (рис. 3.17).

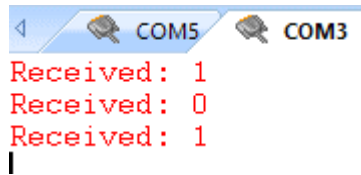


Рисунок 3.17 – Отримані дані через радіозв'язок

Висновки до розділу 3

В даному розділі був описаний порядок роботи з системою при першому запуску. Наведено алгоритм налаштування мережі Wi-Fi.

Було проаналізовано можливості застосування стороннього програмного забезпечення та продемонстровано створення інтерфейсу для зручного управління пристроями розумного будинку за допомогою додатка «MQTT Dash».

В останньому пункті було проведено тестування кожного вузла мережі. Для початку, перевірено роботу модуля ESP8266 з MQTT-брокером за допомогою MQTT Dash. Були перевірені пакети, що формуються для центрального мікроконтролера хабу. Підключивши пристрої автоматизації, що підключені за допомогою I²C та радіозв'язку, також було перевірено, що до них приходять коректні дані.

ВИСНОВКИ

У випускній кваліфікаційній роботі бакалавра було спроектовано та побудовано інформаційну систему для демонстрації можливостей управління розумним будинком. При розробці системи були вирішені наступні завдання:

1. Проведений аналіз серед наявних аналогів. Результатом аналізу були виділені особливості для побудови власної системи, такі як: можливість вибору каналу зв'язку, підключення хабу до мережі Інтернет, можливість потрібних пристроїв працювати автономно, невисока ціна.

2. Була створена узагальнена схема системи задля чіткого розуміння плану розробки. За цією схемою, для кожного окремого блока були обрані та обґрунтовані мікроконтролери або модулі для їх реалізації. Також були обрані інструментальні засоби, такі як додатки для конфігурування мікроконтролерів, інтегровані середовища розробки та додатки для моніторингу COM-порта.

3. Була розроблена повна структурна схема системи, що включала в собі усі модулі для реалізації кожного окремого блоку системи та всі протоколи і інтерфейси, за допомогою яких з'єднуються пристрої та модулі системи. Також було проведено дослідження роботи протоколів схеми, таких як: SPI, UART, I²C, ESB, MQTT.

4. Для передачі даних між центральним мікроконтролером схеми та модулем Wi-Fi був розроблений протокол, що дозволяє передавати пакети даних, які включають в собі: сигнатуру, індекс пристрою, тип даних, самі дані та контрольну суму. Також, в пакет даних була закладена можливість виконання службових команд.

5. Було досліджено методи для створення конфігурації та програмування мікроконтролерів та модулів. Було написано програмний код для мікроконтролерів та пристроїв автоматизації для роботи системи. Для її підключення до мережі Інтернет, був налаштований віддалений MQTT-брокер. Система була комутована, згідно схеми розробленої раніше.

6. Також, був продемонстрований алгоритм для налаштування

підключення до Wi-Fi мережі. Після чого, було продемонстровано можливості роботи системи зі стороннім програмним забезпеченням та створений інтерфейс для управління розумним будинком.

7. Було перевірено під'єднання модуля «ESP8266» до мережі Wi-Fi та протестовано зв'язок з ним. За допомогою інтефейсу, створеного раніше, було перевірено, що модуль Wi-Fi правильно приймає повідомлення від MQTT-брокера та передає їх до центрального мікроконтролера хабу. Також, було перевірено зв'язок з пристроями, з'єднаними з хабом за допомогою шини I²C та за допомогою радіозв'язку.

Було реалізовано інформаційну систему, яка демонструє можливості управління розумним будинком з різними варіантами підключення пристроїв та датчиків.

В подальшому планується розширити систему, шляхом додання різноманітних датчиків та пристроїв автоматизації, для створення конкурентноспроможного аналога готових систем управління розумним будинком. Завдяки розробленому протоколу передачі даних та спроектованій схемі системи її розширення буде зручним та швидким.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. A power line communication network infrastructure for the smart home [Електронний ресурс] / Y.Lin, H. Latchman, M. Lee, S. Katar // IEEE Wirel. Commun.. – 2002. – Режим доступу до ресурсу: <https://www.semanticscholar.org/paper/A-power-line-communication-network-infrastructure-Lin-Latchman/30f11d456739d8d83d8cbf240dea46a26bca6509>.
2. G. Gaurav S. The Evolution of Smart Home Technology [Електронний ресурс] / Sinha G. Gaurav. – 2018. – Режим доступу до ресурсу: <https://blog.bccresearch.com/the-evolution-of-smart-home-technology>.
3. Riahi Sfar A. A systemic and cognitive vision for IoT security: A case study of military live simulation and security challenges [Електронний ресурс] / A. Riahi Sfar, Z. Chtourou, Y. Challal // 2017 International Conference on Smart, Monitored and Controlled Cities (SM2C). – 2017. – Режим доступу до ресурсу: https://www.researchgate.net/publication/320637095_A_systemic_and_cognitive_vision_for_IoT_security_A_case_study_of_military_live_simulation_and_security_challenges.
4. Wolpin S. How To Choose A Smart Home System [Електронний ресурс] / Stewart Wolpin. – 2020. – Режим доступу до ресурсу: <https://www.twice.com/product/product-spotlight/smart-home/how-to-choose-a-smart-home-system>.
5. STM32F103C8T6 - Blue Pill Development Board [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: <https://components101.com/microcontrollers/stm32f103c8t8-blue-pill-development-board>.
6. Nano v.3 [Електронний ресурс] // Hk Shanhai Group Limited – Режим доступу до ресурсу:

https://www.terraelectronica.ru/pdf/show?pdf_file=%252Fds%252Fpdf%252FA%252FArduino_Nano3.pdf.

7. nrf24l01+. Preliminary Product Specification v1.0 [Электронный ресурс] // Nordic Semiconductor – Режим доступа до ресурсу: https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf.
8. ESP-01 WiFi Module. Version 1.0. [Электронный ресурс] // Shenzhen Anxinke Technology Co., Ltd.. – 2015. – Режим доступа до ресурсу: <https://www.microchip.ua/wireless/esp01.pdf>.
9. NodeMCU ESP8266 [Электронный ресурс]. – 2020. – Режим доступа до ресурсу: <https://components101.com/development-boards/nodemcu-esp8266-pinout-features-and-datasheet>.
10. Schnyder M. Migrating from Arduino IDE to Visual Studio Code to PlatformIO [Электронный ресурс] / Michael Schnyder. – 2020. – Режим доступа до ресурсу: <https://blog.emtwo.ch/2020/05/migrating-from-arduino-ide-to-visual.html>.
11. Peña E. UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter [Электронный ресурс] / E. Peña, M. L. Grace – Режим доступа до ресурсу: <https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html>.
12. Yuan M. Getting to know MQTT [Электронный ресурс] / Michael Yuan. – 2017. – Режим доступа до ресурсу: <https://developer.ibm.com/technologies/messaging/articles/iot-mqtt-why-good-for-iot/>.
13. Dhaker P. Introduction to SPI Interface [Электронный ресурс] / Piyu Dhaker // 2017 – Режим доступа до ресурсу: <https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html#>.
14. Enhanced ShockBurst User Guide [Электронный ресурс] // Nordic Semiconductor. – 2016. – Режим доступа до ресурсу: <https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.s>

[dk5.v12.0.0%2Fesb_users_guide.html](#).

15. Afzal S. I2C Primer: What is I2C? (Part 1) [Электронный ресурс] / Sal Afzal – Режим доступа до ресурсу: <https://www.analog.com/en/technical-articles/i2c-primer-what-is-i2c-part-1.html>.
16. Johansson L. The MQTT Beginner's Guide - What is MQTT? [Электронный ресурс] / Lovisa Johansson. – 2020. – Режим доступа до ресурсу: <https://www.cloudmqtt.com/blog/the-mqtt-beginners-guide-what-is-mqtt.html>.
17. STM32 - NRF24L01 [Электронный ресурс]. – 2019. – Режим доступа до ресурсу: <https://istarik.ru/blog/stm32/127.html>

ДОДАТКИ

					КІК.КР.Б. – 123 – 22 – ПЗ	
						61

Лістинг коду для реалізації додатка для підключення до Wi-Fi

```

<Window
  x:Class="diploma_ssld.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:local="clr-namespace:diploma_ssld"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  Title="Configure Wi-Fi"
  Width="171.334"
  Height="194"
  FontFamily="Century Gothic"
  ResizeMode="NoResize"
  WindowStartupLocation="CenterScreen"
  WindowStyle="ToolWindow"
  mc:Ignorable="d">

  <Window.DataContext>
    <local:ComWorker />
  </Window.DataContext>

  <Grid>
    <ComboBox
      Width="76"
      Margin="10,10,0,0"
      HorizontalAlignment="Left"
      VerticalAlignment="Top"
      ItemsSource="{Binding AvailablePortNames}"
      SelectedIndex="{Binding SelectedPortIndex}" />
    <Button
      Width="54"
      Height="22"
      Margin="91,10,0,0"
      HorizontalAlignment="Left"
      VerticalAlignment="Top"
      Command="{Binding RefreshCommand}"
      Content="Refresh" />
    <Button
      Width="135"
      Height="20"
      Margin="10,0,0,10"
      HorizontalAlignment="Left"
      VerticalAlignment="Bottom"
      Command="{Binding ConfigureCommand}"
      Content="Configure" />
    <TextBox
      Width="135"
      Height="23"
      Margin="10,0,0,34"
      HorizontalAlignment="Left"
      VerticalAlignment="Bottom"
      VerticalContentAlignment="Center"
      Text="{Binding Password}"
      TextWrapping="Wrap" />
    <TextBlock
      Margin="10,0,0,57"
      HorizontalAlignment="Left"
      VerticalAlignment="Bottom"
      Text="password"
      TextWrapping="Wrap" />
    <TextBox

```

```

        Width="135"
        Height="24"
        Margin="10,0,0,73"
        HorizontalAlignment="Left"
        VerticalAlignment="Bottom"
        VerticalContentAlignment="Center"
        Text="{Binding Ssid}"
        TextWrapping="Wrap" />
<TextBlock
    Margin="10,0,0,97"
    HorizontalAlignment="Left"
    VerticalAlignment="Bottom"
    Text="ssid"
    TextWrapping="Wrap" />

</Grid>
</Window>

using System;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.IO.Ports;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Windows;

namespace diploma_ssid
{
    public class ComWorker : INotifyPropertyChanged
    {
        #region NotifyPropertyChanged

        public event PropertyChangedEventHandler PropertyChanged;

        protected void OnPropertyChanged([CallerMemberName] string propertyName = "")
        {
            var changed = PropertyChanged;
            if (changed == null)
                return;

            changed.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }

        #endregion

        public ComWorker()
        {
            RefreshAvailablePortNames();
        }

        private int _selectedPortIndex;
        public int SelectedPortIndex
        {
            get
            {
                return _selectedPortIndex;
            }
            set
            {
                _selectedPortIndex = value;
                OnPropertyChanged();
            }
        }
    }
}

```

```

private ObservableCollection<string> _availablePortNames;
public ObservableCollection<string> AvailablePortNames
{
    get
    {
        return _availablePortNames;
    }
    set
    {
        _availablePortNames = value;
        OnPropertyChanged();
    }
}
public void RefreshAvailablePortNames()
{
    AvailablePortNames = new
ObservableCollection<string>(SerialPort.GetPortNames());
    if (AvailablePortNames.Any())
    {
        SelectedPortIndex = 0;
    }
}
public DelegateCommand RefreshCommand =>
    new DelegateCommand(obj => { RefreshAvailablePortNames(); });

private string _ssid = "";
public string Ssid
{
    get
    {
        return _ssid;
    }
    set
    {
        _ssid = value;
        OnPropertyChanged();
    }
}

private string _password = "";
public string Password
{
    get
    {
        return _password;
    }
    set
    {
        _password = value;
        OnPropertyChanged();
    }
}

public DelegateCommand ConfigureCommand =>
    new DelegateCommand(obj => { Configure(); });
public void Configure()
{
    SerialPort serialPort = new
SerialPort(AvailablePortNames[SelectedPortIndex]);
    serialPort.BaudRate = 115200;
    serialPort.Parity = Parity.None;
    serialPort.DataBits = 8;
    serialPort.StopBits = StopBits.One;
}

```

```

serialPort.WriteTimeout = 1000;

try
{
    serialPort.Open();
    serialPort.WriteLine("<" + Ssid + "<" + Password + "<");
    serialPort.Close();

    MessageBox.Show("Done!");
}
catch (Exception ex)
{
    MessageBox.Show("aaa, error!\n" + ex.Message);
}
}
}

```

Лістинг коду для програмування STM32

```

/* USER CODE BEGIN Header */
/**
 * *****
 * @file      : main.c
 * @brief     : Main program body
 * *****
 * @attention
 *
 * <h2><center>&copy; Copyright (c) 2021 STMicroelectronics.
 * All rights reserved.</center></h2>
 *
 * This software component is licensed by ST under BSD 3-Clause license,
 * the "License"; You may not use this file except in compliance with the
 * License. You may obtain a copy of the License at:
 *
 *             opensource.org/licenses/BSD-3-Clause
 *
 * *****
 */
/* USER CODE END Header */
/* Includes ----- */
#include "main.h"
#include "i2c.h"
#include "spi.h"
#include "usart.h"
#include "gpio.h"

#include "uart_pc_lib.h"
#include "protocol.h"
#include "Wire.h"

#define esp_rx_buffer_size 256
#define pc_rx_message_size 32
#define VARIABLE_ID esp_rx_buffer[1]
extern bool MQTT_flag, WIFI_flag;
extern uint8_t esp_rx_buffer[esp_rx_buffer_size];
extern uint8_t esp_tx_buffer[9];
extern uint16_t esp_rx_buffer_len;

extern uint8_t pc_rx_buffer[pc_rx_message_size];
uint8_t ssid[pc_rx_message_size];
uint8_t password[pc_rx_message_size];

bool led;
uint16_t analog = 0;
int32_t color = 0

uint32_t Time_LED;
uint32_t Time_Counter;
uint32_t T;
uint8_t str1[100];

/* Private includes ----- */
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef ----- */

```

```

/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

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

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

/* USER CODE END PM */

/* Private variables -----*/

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----*/
void SystemClock_Config(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code -----*/
/* USER CODE BEGIN 0 */

int _write(int file, uint8_t *ptr, int len)
{
    for (int DataIdx = 0; DataIdx < len; DataIdx++)
    {
        ITM_SendChar(*ptr++);
    }
    return len;
}

void UART_MQTT_Receive_Callback(void)
{
    sprintf(str1, "UART_MQTT_Receive_Callback \r\n");
    HAL_UART_Transmit(&huart2, (uint8_t *)str1, strlen(str1), 0x1000);

    sprintf(str1, "v: 0x%02X \r\n", signature);
    HAL_UART_Transmit(&huart2, (uint8_t *)str1, strlen(str1), 0x1000);

    sprintf(str1, "code: 0x%02X 0x%02X 0x%02X 0x%02X 0x%02X 0x%02X 0x%02X 0x%02X \r\n",
    esp_rx_buffer[0], esp_rx_buffer[1], esp_rx_buffer[2], esp_rx_buffer[3],
    esp_rx_buffer[4], esp_rx_buffer[5], esp_rx_buffer[6], esp_rx_buffer[7],
    esp_rx_buffer[8]);
    HAL_UART_Transmit(&huart2, (uint8_t *)str1, strlen(str1), 0x1000);

    HAL_UART_Transmit(&huart2, (uint8_t *)esp_rx_buffer, strlen(esp_rx_buffer),
    0x1000);

    switch (signature)
    {
    case (0x01):
        if (Protocol_Checksum_validation(esp_rx_buffer))
        {
            led = Protocol_Receive_bool(esp_rx_buffer);
            Time_led = HAL_GetTick();
        }
    }
}

```

```

        if (led)
        {
            GPIOC->BSRR = (uint32_t)LED_Pin << 16u;

            sprintf(str1, "led_on\r\n");
            HAL_UART_Transmit(&huart2, (uint8_t *)str1, strlen(str1),
0x1000);
        }
        else
        {
            GPIOC->BSRR = CLOCK_led_Pin;

            sprintf(str1, "led_off\r\n");
            HAL_UART_Transmit(&huart2, (uint8_t *)str1, strlen(str1),
0x1000);
        }

        if (isChipConnected()) {
            write(led, sizeof(led));
        }

        Wire.beginTransaction(8);
        Wire.write(led);
        Wire.endTransmission();
    }
    break;

case (0x02):
    if (Protocol_Checksum_validation(esp_rx_buffer))
    {
        analog = Protocol_Receive_uint16_t(esp_rx_buffer);
    }
    break;

case (0x03):
    if (Protocol_Checksum_validation(esp_rx_buffer))
    {
        color = Protocol_Receive_int32_t(esp_rx_buffer);
    }
    break;

case (0xFF):
    UART_MQTT_Check_net();
    break;
}

}

void HAL_UART_IDLE_Callback(UART_HandleTypeDef *huart)
{
    if (huart == &huart2)
    {
        UART_PC_Parse_settings(pc_rx_buffer, ssid, password);

        sprintf(str1, "SSID: %s \r\n", ssid);
        HAL_UART_Transmit(&huart2, (uint8_t *)str1, strlen(str1), 0x1000);

        sprintf(str1, "password: %s \r\n", ssid);
        HAL_UART_Transmit(&huart2, (uint8_t *)str1, strlen(str1), 0x1000);

        UART_MQTT_Send_settings_float(ssid, password);
    }
}

```

```

    if (huart == &huart1)
    {
        sprintf(str1, "HAL_UART_IDLE_Callback \r\n");
        HAL_UART_Transmit(&huart2, (uint8_t *)str1, strlen(str1), 0x1000);

        UART_MQTT_IDLE_Callback_setting();
    }
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    if (huart == &huart1)
    {
        sprintf(str1, "HAL_UART_RxCpltCallback \r\n");
        HAL_UART_Transmit(&huart2, (uint8_t *)str1, strlen(str1), 0x1000);

        UART_MQTT_RxCpltCallback_setting();
    }
}

/* USER CODE END 0 */

/**
 * @brief The application entry point.
 * @retval int
 */
int main(void)
{
    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

    /* MCU Configuration-----*/

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

    /* USER CODE BEGIN Init */

    /* USER CODE END Init */

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

    /* USER CODE BEGIN SysInit */

    /* USER CODE END SysInit */

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_USART1_UART_Init();
    MX_USART2_UART_Init();
    MX_I2C1_Init();
    MX_SPI1_Init();
    /* USER CODE BEGIN 2 */

    led = 0;
    T = HAL_GetTick();
    UART_MQTT_init();

```

```

    sprintf(str1, "begin.\r\n");
    HAL_UART_Transmit(&huart2, (uint8_t *)str1, strlen(str1), 1000);

    uint8_t res = isChipConnected();

    snprintf(str1, 64, "Connected: %s\n", res ? "OK" : "NOT OK");
    HAL_UART_Transmit(&huart2, (uint8_t *)str1, strlen(str1), 1000);

    res = NRF_Init();

    snprintf(str1, 64, "Init: %s\n", res > 0 && res < 255 ? "OK" : "NOT OK");
    HAL_UART_Transmit(&huart2, (uint8_t *)str1, strlen(str1), 1000);

    enableAckPayload();
    setChannel(0x60);
    openWritingPipe(pipe0);

    /* USER CODE END 2 */

    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1)
    {
        /* USER CODE END WHILE */

        if (HAL_GetTick() - T >= 1000)
        {
            T = HAL_GetTick();
            analog = analog + 0.4256f;

            if (WIFI_flag && MQTT_flag)
            {
                while (huart1.gState != HAL_UART_STATE_READY)
                ;
                UART_MQTT_Send_data_float(0x03, analog, esp_tx_buffer);

                sprintf(str1, "Counter_float\r\n");
                HAL_UART_Transmit(&huart2, (uint8_t *)str1, strlen(str1),
0x1000);
            }
        }

        if (HAL_GetTick() - Time_led >= 5000)
        {
            Time_led = HAL_GetTick();

            if (WIFI_flag && MQTT_flag)
            {
                while (huart1.gState != HAL_UART_STATE_READY);

                UART_MQTT_Send_data_bool(0x02, led, esp_tx_buffer);

                sprintf(str1, "led\r\n");
                HAL_UART_Transmit(&huart2, (uint8_t *)str1, strlen(str1),
0x1000);
            }
        }

        /* USER CODE BEGIN 3 */

```

```

    }
    /* USER CODE END 3 */
}

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

    /** Initializes the RCC Oscillators according to the specified parameters
    * in the RCC_OscInitTypeDef structure.
    */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
    RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
    RCC_OscInitStruct.HSIState = RCC_HSI_ON;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
    RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
        Error_Handler();
    }
    /** Initializes the CPU, AHB and APB buses clocks
    */
    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_SYCLK |
RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2;
    RCC_ClkInitStruct.SYCLKSource = RCC_SYCLKSOURCE_PLLCLK;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
    {
        Error_Handler();
    }
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
 * @brief This function is executed in case of error occurrence.
 * @retval None
 */
void Error_Handler(void)
{
    /* USER CODE BEGIN Error_Handler_Debug */
    /* User can add his own implementation to report the HAL error return state */
    __disable_irq();
    while (1)
    {
        HAL_UART_Transmit(&huart2, "error\r\n", 7, 0x1000);
        HAL_Delay(1000);
    }
    /* USER CODE END Error_Handler_Debug */
}

```

```

}

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

/***** (C) COPYRIGHT STMicroelectronics *****/

uint8_t NRF_Init(void)
{
    uint8_t setup = 0;
    p_variant = false;
    payload_size = 32;
    dynamic_payloads_enabled = false;
    addr_width = 5;
    pipe0_reading_address[0] = 0x787878787878;

    ce(LOW);
    csn(HIGH);
    HAL_Delay(5);

    write_register(NRF_CONFIG, 0x0C);
    setRetries(5, 15);
    setPALevel(RF24_PA_MAX);

    setDataRate(RF24_250KBPS);

    setup = read_register(RF_SETUP);

    toggle_features();
    write_register(FEATURE, 0);
    write_register(DYNPD, 0);
    dynamic_payloads_enabled = false;

    write_register(NRF_STATUS, (1 << RX_DR) | (1 << TX_DS) | (1 << MAX_RT));
    setChannel(0x60);
    flush_rx();
    flush_tx();
    powerUp();
    write_register(NRF_CONFIG, (read_register(NRF_CONFIG) & ~(1 << PRIM_RX)));
    return (setup != 0 && setup != 0xff);
}

```

Бібліотека для протоколу організації протоколу передачі даних між ESP8266 та STM32

```
#include "protocol.h"
uint8_t esp_tx_buffer[9];
uint8_t status_tx_buffer[9];

uint16_t Calculate_Checksum(uint8_t *tx_buffer)
{
    return (~(tx_buffer[1] +
              tx_buffer[2] +
              tx_buffer[3] +
              tx_buffer[4] +
              tx_buffer[5] +
              tx_buffer[6])) +
           1;
}

void Protocol_Send_bool(uint8_t signature, bool value, uint8_t *tx_buffer)
{
    tx_buffer[0] = t_signature;
    tx_buffer[1] = signature;
    tx_buffer[2] = t_bool;
    tx_buffer[3] = 0x0;
    tx_buffer[4] = 0x0;
    tx_buffer[5] = 0x0;
    tx_buffer[6] = value;
    uint16_t crc = Calculate_Checksum(tx_buffer);
    tx_buffer[7] = crc >> 8;
    tx_buffer[8] = crc;

    Serial.write(tx_buffer, 9);
}

void Protocol_Send_int8_t(uint8_t signature, int8_t value, uint8_t *tx_buffer)
{
    tx_buffer[0] = t_signature;
    tx_buffer[1] = signature;
    tx_buffer[2] = t_int8;
    tx_buffer[3] = value >> 8;
    tx_buffer[4] = value >> 8;
    tx_buffer[5] = value >> 8;
    tx_buffer[6] = value;
    uint16_t crc = Calculate_Checksum(tx_buffer);
    tx_buffer[7] = crc >> 8;
    tx_buffer[8] = crc;

    Serial.write(tx_buffer, 9);
}
```

```

void Protocol_Send_uint8_t(uint8_t signature, int8_t value, uint8_t *tx_buffer)
{
    tx_buffer[0] = t_signature;
    tx_buffer[1] = signature;
    tx_buffer[2] = t_uint8;
    tx_buffer[3] = value >> 8;
    tx_buffer[4] = value >> 8;
    tx_buffer[5] = value >> 8;
    tx_buffer[6] = value;
    uint16_t crc = Calculate_Checksum(tx_buffer);
    tx_buffer[7] = crc >> 8;
    tx_buffer[8] = crc;

    Serial.write(tx_buffer, 9);
}

void Protocol_Send_int16_t(uint8_t signature, int16_t value, uint8_t *tx_buffer)
{
    tx_buffer[0] = t_signature;
    tx_buffer[1] = signature;
    tx_buffer[2] = t_int16;
    tx_buffer[3] = value >> 16;
    tx_buffer[4] = value >> 16;
    tx_buffer[5] = value >> 8;
    tx_buffer[6] = value;
    uint16_t crc = Calculate_Checksum(tx_buffer);
    tx_buffer[7] = crc >> 8;
    tx_buffer[8] = crc;

    Serial.write(tx_buffer, 9);
}

void Protocol_Send_uint16_t(uint8_t signature, uint16_t value, uint8_t *tx_buffer)
{
    tx_buffer[0] = t_signature;
    tx_buffer[1] = signature;
    tx_buffer[2] = t_uint16;
    tx_buffer[3] = value >> 16;
    tx_buffer[4] = value >> 16;
    tx_buffer[5] = value >> 8;
    tx_buffer[6] = value;
    uint16_t crc = Calculate_Checksum(tx_buffer);
    tx_buffer[7] = crc >> 8;
    tx_buffer[8] = crc;

    Serial.write(tx_buffer, 9);
}

void Protocol_Send_int32_t(uint8_t signature, int32_t value, uint8_t *tx_buffer)
{
    tx_buffer[0] = t_signature;
    tx_buffer[1] = signature;
    tx_buffer[2] = t_int32;

```

```

    tx_buffer[3] = value >> 24;
    tx_buffer[4] = value >> 16;
    tx_buffer[5] = value >> 8;
    tx_buffer[6] = value;
    uint16_t crc = Calculate_Checksum(tx_buffer);
    tx_buffer[7] = crc >> 8;
    tx_buffer[8] = crc;

    Serial.write(tx_buffer, 9);
}
void Protocol_Send_uint32_t(uint8_t signature, uint32_t value, uint8_t *tx_buffer)
{
    tx_buffer[0] = t_signature;
    tx_buffer[1] = signature;
    tx_buffer[2] = t_uint32;
    tx_buffer[3] = value >> 24;
    tx_buffer[4] = value >> 16;
    tx_buffer[5] = value >> 8;
    tx_buffer[6] = value;
    uint16_t crc = Calculate_Checksum(tx_buffer);
    tx_buffer[7] = crc >> 8;
    tx_buffer[8] = crc;

    Serial.write(tx_buffer, 9);
}
void Protocol_Send_float(uint8_t signature, float value, uint8_t *tx_buffer)
{
    union
    {
        uint32_t res;
        float f;
    } conv;
    conv.f = value;
    tx_buffer[0] = t_signature;
    tx_buffer[1] = signature;
    tx_buffer[2] = t_float;
    tx_buffer[3] = conv.res >> 24;
    tx_buffer[4] = conv.res >> 16;
    tx_buffer[5] = conv.res >> 8;
    tx_buffer[6] = conv.res;
    uint16_t crc = Calculate_Checksum(tx_buffer);
    tx_buffer[7] = crc >> 8;
    tx_buffer[8] = crc;

    Serial.write(tx_buffer, 9);
}

bool Protocol_Receive_bool(uint8_t *rx_buffer)
{
    bool value = 0x00000000;

```

```

    value = rx_buffer[6];
    return value;
}
int8_t Protocol_Receive_int8_t(uint8_t *rx_buffer)
{
    int8_t value = 0x00000000;
    value = ((value | rx_buffer[3]) << 8) | ((value | rx_buffer[4]) << 8) | ((value
| rx_buffer[5]) << 8) | ((value | rx_buffer[6]));
    return value;
}
uint8_t Protocol_Receive_uint8_t(uint8_t *rx_buffer)
{
    uint8_t value = 0x00000000;
    value = ((value | rx_buffer[3]) << 8) | ((value | rx_buffer[4]) << 8) | ((value
| rx_buffer[5]) << 8) | ((value | rx_buffer[6]));
    return value;
}
int16_t Protocol_Receive_int16_t(uint8_t *rx_buffer)
{
    int16_t value = 0x00000000;
    value = ((value | rx_buffer[3]) << 16) | ((value | rx_buffer[4]) << 16) | ((val
ue | rx_buffer[5]) << 8) | ((value | rx_buffer[6]));
    return value;
}
uint16_t Protocol_Receive_uint16_t(uint8_t *rx_buffer)
{
    uint16_t value = 0x00000000;
    value = ((value | rx_buffer[3]) << 16) | ((value | rx_buffer[4]) << 16) | ((val
ue | rx_buffer[5]) << 8) | ((value | rx_buffer[6]));
    return value;
}
int32_t Protocol_Receive_int32_t(uint8_t *rx_buffer)
{
    int32_t value = 0x00000000;
    value = ((value | rx_buffer[3]) << 24) | ((value | rx_buffer[4]) << 16) | ((val
ue | rx_buffer[5]) << 8) | ((value | rx_buffer[6]));
    return value;
}
uint32_t Protocol_Receive_uint32_t(uint8_t *rx_buffer)
{
    uint32_t value = 0x00000000;
    value = ((value | rx_buffer[3]) << 24) | ((value | rx_buffer[4]) << 16) | ((val
ue | rx_buffer[5]) << 8) | ((value | rx_buffer[6]));
    return value;
}
float Protocol_Receive_float(uint8_t *rx_buffer)
{
    union
    {
        uint32_t res;

```

```

        float f;
    } conv;
    uint32_t value = 0x00000000;
    conv.res = ((value | rx_buffer[3]) << 24) | ((value | rx_buffer[4]) << 16) | ((
value | rx_buffer[5]) << 8) | ((value | rx_buffer[6]));
    float _float = conv.f;
    return (_float);
}
bool Protocol_Checksum_validation(uint8_t *rx_buffer)
{
    bool result;
    uint16_t crc_rx_buffer = 0x0000;
    uint16_t crc_check = 0x0000;
    crc_rx_buffer = Calculate_Checksum(rx_buffer);
    crc_check = (crc_check | rx_buffer[7] << 8) | (crc_check | rx_buffer[8]);

    if (crc_rx_buffer == crc_check)
    {
        result = true;
    }
    else
    {
        result = false;
    }
    return result;
}

void wifi_ok(void)
{
    esp_tx_buffer[0] = 0xFF;
    esp_tx_buffer[1] = 0xFF;
    esp_tx_buffer[2] = 0xFF;
    esp_tx_buffer[3] = 0xAA;
    esp_tx_buffer[4] = 0xAA;
    esp_tx_buffer[5] = 0xAA;
    esp_tx_buffer[6] = 0xAA;
    esp_tx_buffer[7] = 0xFB;
    esp_tx_buffer[8] = 0x5A;
    Serial.write(esp_tx_buffer, 9);
}

void wifi_not_ok(void)
{
    esp_tx_buffer[0] = 0xFF;
    esp_tx_buffer[1] = 0xFF;
    esp_tx_buffer[2] = 0xFF;
    esp_tx_buffer[3] = 0xBB;
    esp_tx_buffer[4] = 0xBB;
    esp_tx_buffer[5] = 0xBB;
    esp_tx_buffer[6] = 0xBB;

```

```

    esp_tx_buffer[7] = 0xFB;
    esp_tx_buffer[8] = 0x16;
    Serial.write(esp_tx_buffer, 9);
}

```

```

void mqtt_ok(void)
{
    esp_tx_buffer[0] = 0xFF;
    esp_tx_buffer[1] = 0xFF;
    esp_tx_buffer[2] = 0xFF;
    esp_tx_buffer[3] = 0xCC;
    esp_tx_buffer[4] = 0xCC;
    esp_tx_buffer[5] = 0xCC;
    esp_tx_buffer[6] = 0xCC;
    esp_tx_buffer[7] = 0xFA;
    esp_tx_buffer[8] = 0xD2;
    Serial.write(esp_tx_buffer, 9);
}

```

```

void mqtt_not_ok(void)
{
    esp_tx_buffer[0] = 0xFF;
    esp_tx_buffer[1] = 0xFF;
    esp_tx_buffer[2] = 0xFF;
    esp_tx_buffer[3] = 0xDD;
    esp_tx_buffer[4] = 0xDD;
    esp_tx_buffer[5] = 0xDD;
    esp_tx_buffer[6] = 0xDD;
    esp_tx_buffer[7] = 0xFA;
    esp_tx_buffer[8] = 0x8E;
    Serial.write(esp_tx_buffer, 9);
}

```

Лістинг коду для програмування ESP8266

```

#include <Arduino.h>

#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <EEPROM.h>

#include "protocol.h"
#include "esp_status.h"

#define signature rx_buffer[1]
#define mqtt_name "esp8266_hub"
#define configuration_size 16

char *ssid = "netis_2.4G";
char *pass = "7yzdr9v3";

struct string_eeprom
{
    String val;
    int addr;
};

string_eeprom saved_ssid = {ssid, 0};
string_eeprom saved_password = {pass, saved_ssid.addr + sizeof(String)};

const char *mqtt_server = "driver.cloudmqtt.com";
const int mqtt_port = 18741;
const char *mqtt_user = "lqakunvs";
const char *mqtt_pass = "E3PUM3Sv3jzJ";

uint8_t rx_buffer[9] = {
    0,
};

uint8_t tx_buffer[9] = {
    0,
};

bool led = 0;
float analog = 0.0f;
int32_t color = 0;

WiFiClient wclient;
PubSubClient client(wclient, mqtt_server, mqtt_port);

void callback(const MQTT::Publish &pub);
void uart_parsing(void);

```

```

void callback(const MQTT::Publish &pub)
{
    String payload = pub.payload_string();

    if (String(pub.topic()) == "diploma/led")
    {
        int test_led = payload.toInt();

        if (test_led == 1)
        {
            Protocol_Send_bool(0x01, test_led, tx_buffer);
            //digitalWrite(1, HIGH);
            //digitalWrite(2, HIGH);
        }
        else if (test_led == 0)
        {
            Protocol_Send_bool(0x01, test_led, tx_buffer);
            //digitalWrite(1, LOW);
            //digitalWrite(2, LOW);
        }
    }

    if (String(pub.topic()) == "diploma/analog")
    {
        analog = payload.toFloat();
        Protocol_Send_float(0x02, analog, tx_buffer);
    }

    if (String(pub.topic()) == "diploma/color")
    {
        color = payload.toInt();
        Protocol_Send_int32_t(0x03, analog, tx_buffer);
    }
}

void uart_parsing(void)
{
    if (Serial.available() > 0)
    {
        Serial.readBytes(rx_buffer, 9);

        if (rx_buffer[0] == 0xFD)
        {
            Serial.readBytes(rx_buffer, configuration_size);
            memcpy(rx_buffer, ssid, sizeof(rx_buffer));

            char *temp = new char[sizeof(ssid)];
            for (int i = 0; i < sizeof(ssid); i++)
                temp[i] = i + 1;
            for (int i = 0; i < sizeof(ssid) - 1; i++)

```

```

        ssid[i] = temp[i + 1];
        delete[] temp;
    }
    if (rx_buffer[0] == 0xFE)
    {
        Serial.readBytes(rx_buffer, configuration_size);
        memcpy(rx_buffer, pass, sizeof(rx_buffer));

        char *temp = new char[sizeof(pass)];
        for (int i = 0; i < sizeof(pass); i++)
            temp[i] = i + 1;
        for (int i = 0; i < sizeof(pass) - 1; i++)
            pass[i] = temp[i + 1];
        delete[] temp;

        EEPROM.begin(configuration_size * 2);
        delay(500);
        EEPROM.put(saved_ssid.addr, saved_ssid.val);
        EEPROM.put(saved_password.addr, saved_password.val);
        EEPROM.commit();
        EEPROM.end();
        delay(500);

        WiFi.disconnect();
        wifi_not_ok();
        WiFi.begin(ssid, pass);

        if (WiFi.waitForConnectResult() != WL_CONNECTED)
        {
            wifi_ok();
        }
    }

    switch (signature)
    {
    case (0x01):
        if (Protocol_Checksum_validation(rx_buffer))
        {
            led = Protocol_Receive_bool(rx_buffer);
            client.publish("diploma/led", String(led));
        }
        break;
    case (0x02):
        if (Protocol_Checksum_validation(rx_buffer))
        {
            analog = Protocol_Receive_float(rx_buffer);
            client.publish("diploma/analog", String(analog));
        }
        break;
    }

```

```

    case (0x03):
        if (Protocol_Checksum_validation(rx_buffer))
        {
            analog = Protocol_Receive_int32_t(rx_buffer);
            client.publish("diploma/color", String(analog));
        }
        break;
    }
    memset(rx_buffer, 0, 9);
}

void setup(void)
{
    EEPROM.begin(configuration_size * 2);
    delay(500);
    EEPROM.get(saved_ssid.addr, saved_ssid.val);
    EEPROM.get(saved_password.addr, saved_password.val);
    EEPROM.end();

    Serial.begin(9600);
    delay(100);
    Serial.setTimeout(50);
    Serial.printf("rrr");

    //led = 1;
}

void loop(void)
{
    if (WiFi.status() != WL_CONNECTED)
    {
        wifi_not_ok();
        WiFi.begin(ssid, pass);

        if (WiFi.waitForConnectResult() != WL_CONNECTED)
            return;
        wifi_ok();
    }

    if (WiFi.status() == WL_CONNECTED)
    {
        if (!client.connected())
        {
            mqtt_not_ok();
            if (client.connect(MQTT::Connect(mqtt_name).set_auth(mqtt_user, mqtt_pass)))
            {
                mqtt_ok();
                client.set_callback(callback);

                client.subscribe("diploma/led");
            }
        }
    }
}

```

```

        client.subscribe("diploma/analog");
        client.subscribe("diploma/color");
    }
    else
    {
        mqtt_not_ok();
    }
}

if (client.connected())
{
    client.loop();
    uart_parsing();
}
}
}

```

Лістинг коду для програмування пристрою автоматизації, з'єднаного за
допомогою радіозв'язку

```

void setup(){
  Serial.begin(9600);
  radio.begin();
  radio.setAutoAck(1);
  radio.setRetries(0,15);
  radio.enableAckPayload();
  radio.setPayloadSize(32);

  radio.openReadingPipe(1,address[0]);
  radio.setChannel(0x60);

  radio.setPALevel (RF24_PA_MAX);
  radio.setDataRate (RF24_250KBPS);

  radio.powerUp();
  Serial.print("Started listening: ");
  radio.startListening();
}

void loop() {
  byte pipeNo, gotByte;
  while( radio.available(&pipeNo)){
    radio.read( &gotByte, sizeof(gotByte) );

    Serial.print("Recieved: "); Serial.println(gotByte);
  }
}

```

Лістинг коду для програмування пристрою автоматизації (Arduino Nano),
з'єднаного за допомогою шини I²C

```
#include <Arduino.h>
#include <Wire.h>

#define LED 7

void setup()
{
    Serial.begin(9600);

    Wire.begin(8);
    Wire.onReceive(receive_handler);
}

void loop()
{
    delay(100);
}

void receive_handler(int count)
{
    byte value = Wire.read();

    if (value == 1)
    {
        digitalWrite(LED_BUILTIN, HIGH);
        Serial.println("LED on");
    }
    else
    {
        digitalWrite(LED_BUILTIN, LOW);
        Serial.println("LED off");
    }

    delay(500);
}
```