МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ
Навчально-науковий інститут автоматики, кібернетики та обчислювальної техніки

"До захисту допущена"
Зав. кафедри комп'ютерних наук та прикладної математики
д.т.н., проф. Турбал Ю.В.
« » 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

Розробка модульної системи спостереження за параметрами середовища

(назва теми роботи)

Виконав: Середа Владислав Олександрович _____
(прізвище, ім'я, по батькові) (підпис)
студент групи ІПЗ-41

Керівник: к.т.н., доц. Климюк Ю. Є. _____
(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Рівне – 2023

ЗМІСТ

РЕФЕРАТ

Кваліфікаційна робота: 61 с. 24 малюнків, 4 таблиць, 1 додаток, 14 джерел.

Актуальність теми: Зростаюча доступність та популярність систем автоматизації дозволяє впроваджувати їх для багатьох завдань, проте, часто затребуваний час або ціна перешкоджають цьому. Актуальність теми полягає у вирішенні цієї проблеми шляхом дослідження принципів роботи таких систем та створенні власного доступного аналогу, що буде мати меншу ціну, більший радіус дії та пристосований до модифікацій та використанню у парі з іншими пристроями користувача. Це дозволить впроваджувати власну автоматизацію, не опираючись на готові ринкові рішення.

Мета кваліфікаційної роботи — аналіз та систематизація основних критеріїв та вимог до систем розумного будинку і домашньої автоматизації та процесів розробки таких систем; створення на основі отриманих знань власної системи автоматизації.

Об’єкт дослідження — робота систем розумного будинку та домашньої автоматизації.

Предметом досліджень — методи та моделі для проектування центральних контролерів для систем автоматизації.

Методи дослідження: на основі проаналізованих методів побудови автоматизованих систем, вибір найбільш відповідної апаратної бази, вибір засобів розробки програмного забезпечення для контролера, розробка алгоритмів керування роботою об'єкта та реалізація цих алгоритмів в програмному забезпеченні для контролера.

Реалізований центральний контролер та засоби для побудови автоматизованої модульної системи усуває проблему тривалого розгортання власних спеціалізованих систем домашньої автоматизації.

ВСТУП

Актуальність теми модульної системи спостереження за параметрами середовища розкривається у контексті організації систем та приладів розумного будинку. Розумний будинок — це інноваційний підхід до житлових просторів, який поєднує автоматизацію, збір та аналіз даних з різних датчиків для забезпечення зручності, енергоефективності та безпеки. Основною метою розробки таких систем є забезпечення оперативного контролю за параметрами середовища, щоб мати можливість вчасно та комплексно реагувати на ситуації різного характеру.

Сьогодні, попри великий ринок рішень для систем автоматизації та розумних будинків, періодично, для спеціальних завдань чи умов необхідно створювати власні системи, це може потребувати часу та ресурсів більше ніж вартує кінцевий результат. Для вирішення цієї проблеми в цій роботі буде досліджено вимоги та принципи побудови, використання і роботи систем розумного будинку, домашньої автоматизації, щоб створити модульне рішення для тих аспектів систем, що, зазвичай, не є унікальними, як от зв'язок чи робота з даними. На основі результату в подальшому можна буде швидко розгортати рішення для спеціалізованих задач зосередившись лише на унікальних аспектах завдання. Додатково, завдяки модульному підходу, створені системи зможуть легко в ході експлуатації розширюватись та адаптуватись до конкретних потреб користувачів, дозволяючи додавати нові периферійні пристрої системи або змінювати налаштування системи відповідно до змінюваних умов. Основні речі, що будуть розглянуті — це архітектура та топологія мережі, популярні комунікаційні протоколи та мікроконтролери для побудови таких систем.

Актуальність теми: Зростаюча доступність та популярність систем автоматизації дозволяє впроваджувати їх для багатьох завдань, проте, часто затребуваний час або ціна перешкоджають цьому. Актуальність теми полягає у вирішенні цієї проблеми шляхом дослідження принципів роботи таких систем та створенні власного доступного аналогу, що буде мати меншу ціну, більший радіус дії та пристосований до модифікацій та використанню у парі з іншими пристроями

користувача. Це дозволить впроваджувати власну автоматизацію, не опираючись на готові ринкові рішення.

Мета кваліфікаційної роботи — аналіз та систематизація основних критеріїв та вимог до систем розумного будинку і домашньої автоматизації та процесів розробки таких систем; створення на основі отриманих знань власної системи автоматизації.

Об’єкт дослідження — робота систем розумного будинку та домашньої автоматизації.

Предметом досліджень — методи та моделі для проектування центральних контролерів для систем автоматизації.

Методи дослідження: на основі проаналізованих методів побудови автоматизованих систем, вибір найбільш відповідної апаратної бази, вибір засобів розробки програмного забезпечення для контролера, розробка алгоритмів керування роботою об'єкта та реалізація цих алгоритмів в програмному забезпеченні для контролера.

РОЗДІЛ І. ЗАГАЛЬНІ ВІДОМОСТІ

1.1. Історичний огляд систем розумного будинку

Системи розумного будинку та домашньої автоматизації є результатом поєднання передових технологій з метою забезпечення зручності, енергоефективності та безпеки життя в будинку. Розвиток цієї галузі був процесом, що почався десятиліття тому і переживав значні зміни протягом історії.

Початкові кроки виникнення та розвитку домашньої автоматизації були зроблені в 20–30-х роках XX століття, коли виникли перші спроби застосування електромеханічних систем для керування різними пристроями в будинках. Початки ж систем розумного будинку можна відстежити дещо пізніше в 70-х роках, коли почалися перші експерименти з автоматизованими системами керування будинком. У цей період в основному використовувались проводові системи з комутацією за допомогою реле. Основною метою було забезпечення контролю за освітленням та опаленням.

Одним з перших популярних стандартів систем домашньої автоматизації став стандарт "X10", запропонована у 1975 році. Він передбачав використання електричної проводки для передачі сигналів керування між пристроями в будинку. Наступні ітерації цієї технології додали бездротову комунікацію та покращили функціональні можливості [1]. Сьогодні стандарт досі залишається популярним, попри наявність альтернатив з більшим числом можливостей.

Протягом 1980-х років з'явилися перші бездротові системи розумного будинку, що дозволило покращити гнучкість та легкість установки. Такі системи включали можливість керування освітленням, опаленням, а також деякими домашніми пристроями.

У 1990-х роках спостерігалося зростання інтересу до систем розумного будинку. Розвиток мережі Інтернет та стандартів передачі даних створили нові можливості для забезпечення віддаленого керування будинком. З'явилися перші комплексні системи, що об'єднували управління освітленням, опаленням, безпекою та іншими функціями в єдиному інтегрованому інтерфейсі. Також з'явилися перші

системи, які включали голосове керування. Наприклад, в 1999 році компанія Microsoft представила проект "Microsoft Home", який демонстрував можливості розумного будинку, заснованого на їхніх технологіях.

Технології продовжували розвиватися, але широке поширення отримали лише у 2000-х роках. З появою доступних мікропроцесорів, бездротових засобів зв'язку та Інтернету речей (IoT), розумні будинки стали більш доступними та ефективними. Великі компанії електроніки включили до своєї продукції рішення для "розумного будинку", що дозволило широкому загалу скористатися перевагами автоматизації та віддаленого контролю.

Сьогодні системи розумного будинку розвиваються в різних напрямках. Розширюється функціональність систем, включаючи інтеграцію зі штучним інтелектом, мобільними додатками та "розумними" пристроями, які, зокрема, можуть самостійно збирати та аналізувати дані для покращення життя мешканців. Впроваджуються енергоефективні та екологічні рішення, такі як керування енергоспоживанням та використання відновлювальних джерел енергії.

В найближчому майбутньому системи розумних будинків будуть далі розвиватись в напрямку використання штучного інтелекту, що дозволить значно розширити можливості та точність роботи таких систем.

1.2. Визначення та типові складові систем розумного будинку

Розумний будинок (smart home) — це концепція, що описує будинок, в якому різні електронні пристрої та системи зв'язку інтегровані в єдину систему для забезпечення автоматизованого керування та контролю над різними аспектами побуту. Також існує поняття "система домашньої автоматизації", воно має багато спільного з розумним будинком, проте, прийнято вважати, що розумний будинок — це більш інтелектуальна система, що опирається в більшій мірі на аналіз даних та на його основі коректує свою роботу.

Основна ідея розумного будинку полягає в тому, щоб створити середовище, в якому різні пристрої можуть взаємодіяти як між собою, так і з користувачем, забезпечуючи комфорт, безпеку, енергоефективність та зручність.

Основні складові системи розумного будинку поділяють на:

1. Сенсори та датчики — збирають дані про оточуюче середовище та стан будинку.
2. Актуатори — виконують дії відповідно до отриманих команд.
3. Центральний контролер, контрольні панелі та інтерфейси — надають засоби керування та моніторингу системи розумного будинку; сюди ж відносяться веб інтерфейси, мобільні додатки тощо.

Контролер розумного будинку, також відомий як "головний контролер" або "центральний хаб" є серцем системи, яке забезпечує інтеграцію, керування та автоматизацію різних пристроїв і систем в будинку. Він має такі ключові аспекти:

- Централізоване керування — головний контролер є місцем, де зібрані дані з різних пристроїв і систем розумного будинку та наданий доступ до керування цими пристроями та системами.
- Керування через інтерфейс — в ролі інтерфейсу керуючого центру можуть виступати: мобільний додаток, веб-інтерфейс, пульт дистанційного керування тощо.
- Автоматизація та сценарії роботи.

Компоненти системи розумного будинку взаємодіють між собою за допомогою комунікаційних протоколів. Популярними безпровідними протоколами зв'язку є Wi-Fi, ZigBee, Z-Wave та Bluetooth. Серед провідних протоколів поширені Ethernet, RS-485, KNX (EIB) та Modbus. Така інтеграція дозволяє системі розумного будинку працювати як єдиний організм, забезпечуючи зручне та ефективне керування різними аспектами життя в будинку.

1.3. Популярні рішення

На сучасному ринку існує багато виробників та брендів, які пропонують різні системи розумного будинку з різноманітними функціями та можливостями. Деякі з найпопулярніших систем розумного будинку на сьогоднішній день:

- Amazon Alexa: одна з провідних систем розумного будинку на ринку, має широкий спектр сумісних пристроїв та додатків.
- Google Home: інтегрується з платформою Google Assistant.
- Apple HomeKit: працює на основі інтеграції з Apple-пристроями.
- Samsung SmartThings: інтегрована система розумного будинку, яка працює на базі хмарних технологій. Підтримує широкий спектр пристроїв та сенсорів. Завдяки платформі SmartThings, користувачі можуть налаштовувати автоматичні сценарії та взаємодіяти з системою через мобільний додаток.

Програмне забезпечення є ключовим компонентом систем розумного будинку, оскільки воно відповідає за керування, автоматизацію та інтеграцію різних пристроїв та функцій будинку.

Деякі з популярних програмних платформ для систем розумного будинку:

- OpenHAB (Open Home Automation Bus) — це відкрите програмне забезпечення для систем розумного будинку. Воно надає можливості збору даних, керування пристроями та автоматизації через різні протоколи, такі як ZigBee, Z-Wave, KNX та інші. OpenHAB пропонує гнучкість та розширюваність, дозволяючи користувачам налаштовувати та інтегрувати різноманітні компоненти своєї системи розумного будинку.
- Home Assistant — популярна відкрита платформа для систем розумного будинку, базується на принципах відкритості, гнучкості та співпраці з різними пристроями та сервісами. Home Assistant підтримує широкий спектр протоколів та пристроїв, дозволяючи інтегрувати освітлення, безпеку, електроприлади та багато інших функцій. Вона також надає зручний веб-інтерфейс та мобільні додатки для керування системою.
- SmartThings — програмне забезпечення, розроблене компанією Samsung, яке дозволяє керувати та автоматизувати різні аспекти будинку. Воно підтримує широкий спектр пристроїв, які можна інтегрувати, включаючи освітлення, термостати, дверні замки, відеокамери та багато інших. SmartThings

пропонує інтуїтивний веб-інтерфейс та мобільні додатки для зручного керування та моніторингу системи.

- Apple HomeKit — програмна платформа, розроблена компанією Apple, яка спрощує керування різними пристроями та функціями в будинку через їх екосистему. За допомогою Apple HomeKit, користувачі можуть керувати освітленням, опаленням, безпекою та іншими пристроями за допомогою голосових команд, мобільних пристроїв або автоматичних сценаріїв.

1.4. Тенденції розвитку систем розумного будинку

Системи розумного будинку продовжують розвиватись та еволюціонувати, пропонуючи нові можливості та функції. Актуальний напрям розвитку — це використання нейронних мереж для оптимізації існуючих та побудови більш функціональних систем, що стає можливим за рахунок росту обчислювальних потужностей мікроконтролерів та інтеграції з хмарними технологіями.

В плані менш глобальних змін можна виділити наступне:

- Розширена інтеграція з пристроями Інтернету речей, дозволяє створювати більш складні та автоматизовані сценарії, що покращують зручність та ефективність систем розумного будинку.
- Розвиток розпізнавання та синтезу мовлення для голосового керування, це забезпечує зручність та швидкість взаємодії з системою, не потребуючи фізичного керування пристроями або додатками.
- Збільшення взаємодії з різноманітними сервісами та інтеграція з хмарними платформами, це дозволяє отримувати доступ до додаткових функцій та послуг, таких як погодні оновлення, керування енергозбереженням, мультимедійні потоки та інші. Інтеграція з хмарними платформами також забезпечує зручний дистанційний доступ до системи розумного будинку через мобільні додатки або веб-інтерфейси.
- Покращення методів кібербезпеки для захисту систем розумного будинку від несанкціонованого доступу.

РОЗДІЛ ІІ. ТЕХНОЛОГІЇ ТА СТАНДАРТИ

2.1. Архітектура системи

За способом організації взаємодії компонентів систем розумного будинку та автоматизації їх можна поділити на централізовані та децентралізовані системи [2].

У централізованій архітектурі всі пристрої та компоненти підключаються до центрального керуючого пристрою або хабу. Центральний пристрій відповідає за збір, обробку та аналіз даних з усіх підключених пристроїв, а також за управління ними. Всі рішення приймаються центральним пристроєм, і він координує всі дії у системі. Наприклад, якщо необхідно змінити налаштування освітлення або температури в будинку, потрібно взаємодіяти з центральним пристроєм, який передасть команди відповідним пристроям.

Переваги централізованої архітектури:

- централізований керуючий пристрій забезпечує єдиний пункт керування для всієї системи, що спрощує управління;
- централізована архітектура може бути більш ефективною у вирішенні конфліктів та взаємодії між пристроями, оскільки всі дії ініціалізує один пристрій.

Недоліки централізованої архітектури:

- один пристрій відповідає за функціонування всієї системи, тому необхідно приділити особливу увагу його стабільній та правильній роботі і захисту від несанкціонованого доступу.

У децентралізованій архітектурі кожен пристрій або компонент має свою власну функціональність та можливість приймати рішення самостійно. Кожен пристрій може комунікувати безпосередньо з іншими пристроями, обмінюючись інформацією та виконуючи дії на основі отриманих даних, без участі центрального пристрою.

Переваги децентралізованої архітектури:

- вихід з ладу центрального контролера не призведе до припинення функціонування всієї системи;

- центральний пристрій зберігає менше даних, що зменшує масштаб проблем при несанкціонованому доступі.

Недоліки децентралізованої архітектури:

- більша кількість самостійних приладів системи вимагає більшої уваги до організації порядку їх роботи;
- складніша мережа комунікації.

Варто зазначити, що збір інформації про стан компонентів системи в одному керуючому пристрої є однією з концепцій систем розумного будинку, тому в правильно організованій мережі пристрої системи тим чи іншим способом повинні сповіщати центральний контролер про свій стан. Такі системи залишаються децентралізованими, оскільки пристрої мережі діють самостійно, проте, за рахунок централізованого зберігання даних, з'являється можливість коригувати роботу компонентів на основі даних всієї мережі, що зменшує шанс виникнення конфліктних ситуацій, а відповідно спрощує організацію порядку роботи компонентів. Це також повертає проблеми, що виникають при несанкціонованому доступі до центрального контролера, проте питання безпеки мережі повинні мати високий пріоритет, не залежно від способу організації взаємодії компонентів між собою.

Окрім взаємодії компонентів, типи мереж поділяються за способом підключення компонентів мережі між собою [3].

Точка-точка (point-to-point) — найпростіша топологія з виділеним з'єднанням між двома вузлами.

Послідовне / ланцюжкове з'єднання (Daisy chain) — послідовне з'єднанням кожного пристрою з наступним. Вузли, що знаходяться між двома пристроями, що ведуть діалог, пересилають повідомлення від попереднього вузла до наступного. Така мережа може мати лінійну та кільцеву форми. Лінійна топологія повинна мати двосторонній зв'язок між вузлами. В кільцевій топології, коли вузол надсилає повідомлення, воно обробляється та ретранслюється кожним вузлом у кільці, доки не дійде до отримувача. З'єднання колом дозволяє здійснювати передачу тільки в

одному напрямку, проте у випадку виходу з ладу одного з вузлів, двосторонній зв’язок дозволить мережі далі функціонувати.

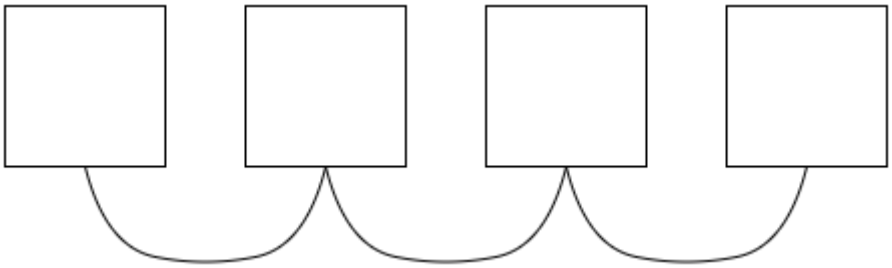

Рис. 2.1. Послідовне з’єднання.

Шина (Bus) — кожен вузол підключається до одного центрального кабелю. Усі дані передаються через спільну магістраль та можуть бути отримані всіма вузлами мережі одночасно. Така топологія має вищу вартість керування мережею.

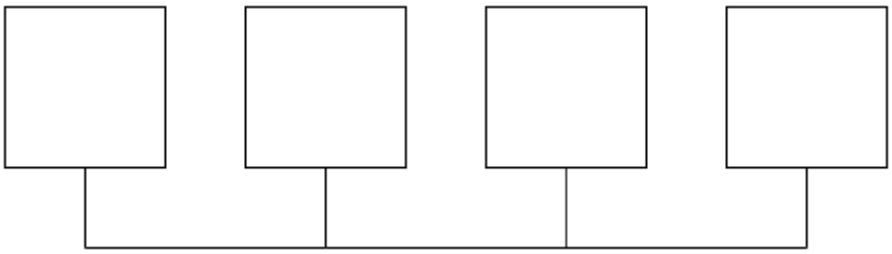

Рис. 2.2. Шина.

Зірка (Star) — кожен периферійний вузол підключений до центрального вузла. Однією з переваг зіркоподібної топології є простота додавання додаткових вузлів. Основним недоліком є те, що потрібна особлива увага до надійності центрального вузла та до його пропускної здатності. За таким типом топології працюють Wi-Fi мережі.

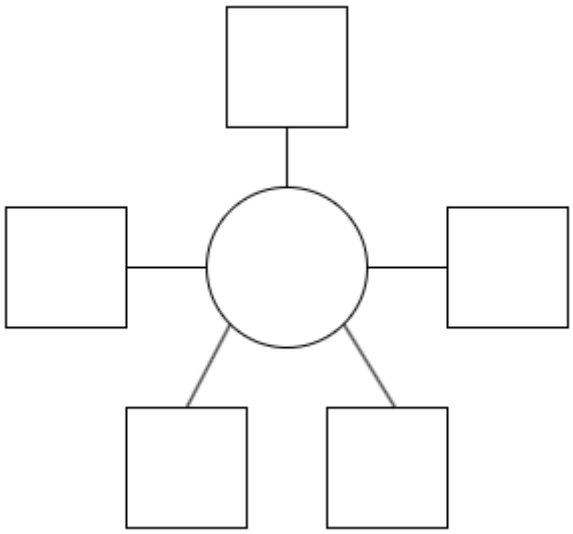

Рис. 2.3. Топологія Зірка.

Сітка (Mesh) — вузли мережі з’єднані між собою повністю або частково. Максимальну кількість з’єднань можна підрахувати за формулою повної кількості зв’язків у графі: $c = \frac{n(n-1)}{2}$, де c — кількість зв’язків, n — кількість вузлів. Такий тип топології використовують мережі ZigBee.

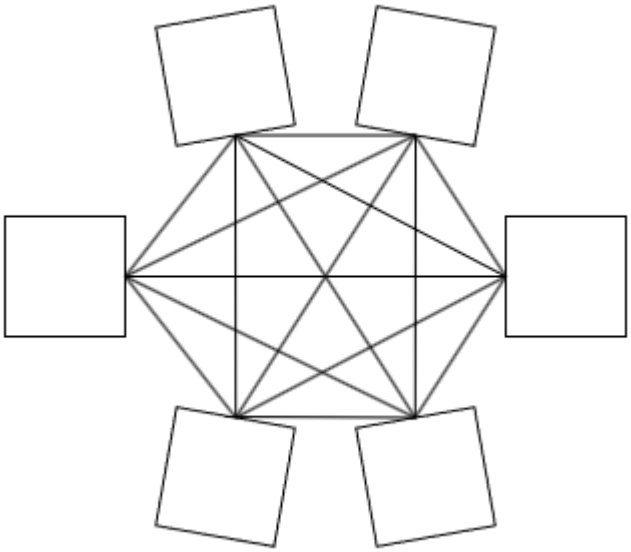

Рис. 2.4. Топологія Сітка.

Топологія мережі також може бути гібридною, поєднуючи в собі гілки кількох типів топології.

2.2. Мікроконтролери

Мікроконтролери для приладів системи розумного будинку обираються на основі типу архітектури системи, типу комунікаційних протоколів та складності логіки приладу [4]. Для центральних контролерів обираються потужніші мікроконтролери або міні комп'ютери. Для сенсорів та актуаторів, зазвичай, ставляться прості задачі на кшталт періодичного опитування датчика та відправки результатів до центрального контролера, тому для цих задач використовуються простіші мікроконтролери.

Популярні мікроконтролери для розробки пристроїв розумного будинку та IoT:

- ATmega328P, ATtiny85, ATtiny13;
- ESP32 та ESP8266;
- Raspberry Pi: мікропроцесори BCM;
- STM32.

У складі готових рішень від альянсу ZigBee використовуються мікроконтролери архітектури SoC (System-On-Chip), такі як Silicon Labs EFR32, Texas Instruments CC2530/CC2531, NXP JN516x тощо. Вони розроблялись спеціально для систем розумного будинку та пристроїв інтернету речей, тому є добре пристосованими до використання в таких пристроях, проте вони менш доступні для використання у власних розробках.

З популярних AVR мікроконтролерів можна виділити кілька основних від Microchip Technology (раніше Atmel), а саме ATmega328P, ATtiny85 та з певною умовністю ATtiny13. Вони працюють на частоті 20 МГц та мають розрядність 8 біт, для програмування можна використовувати мови C та Асемблер [5][7]. Популярним середовищем розробки є Arduino IDE. Основна перевага цих мікроконтролерів це висока функціональність та доступність. Додатково, внаслідок їх популярності, існує велика кількість документації і матеріалів по роботі з ними та сумісних модулів і датчиків, що дозволяє реалізувати за їх допомогою майже все, що не потребує великих обчислювальних потужностей та об'ємів пам'яті. В контексті розумного будинку чи пристроїв інтернету речей їх недоліком є відсутність вбудованої підтримки стандартів бездротового зв'язку.

Суттєвою різницею між ATmega328P, ATtiny85 та ATtiny13 є відсутність в tiny контролерів апаратної підтримки протоколів обміну даними SPI, I2C та UART, проте з певними умовностями частково їх можна реалізувати програмно через GPIO [6].

Мікроконтролер ATmega328 став дуже популярним у розвитку вбудованих систем та проектах з електронікою завдяки своїй простоті використання, надійності та широкому спектру функціональних можливостей. Він має архітектуру RISC (Reduced Instruction Set Computer) з 8-розрядним процесором і працює з тактовою частотою до 20 МГц.

Основні характеристики ATmega328:

- 32 кб Flash пам'яті, 2 кб оперативної пам'яті (SRAM) та 1 кб енергонезалежної пам’яті (EEPROM);
- 23 цифрових портів входу/виходу, 8 аналогових входів, які можуть приймати напругу від 0 до 5 вольт;
- 10-бітний АЦП;
- 2 8-ми та 1 16-бітний таймер/лічильник (T/C)
- підтримувані режими сну: Idle, ADC Noise Reduction, Power-down, Power-save, Standby та Extended Standby.

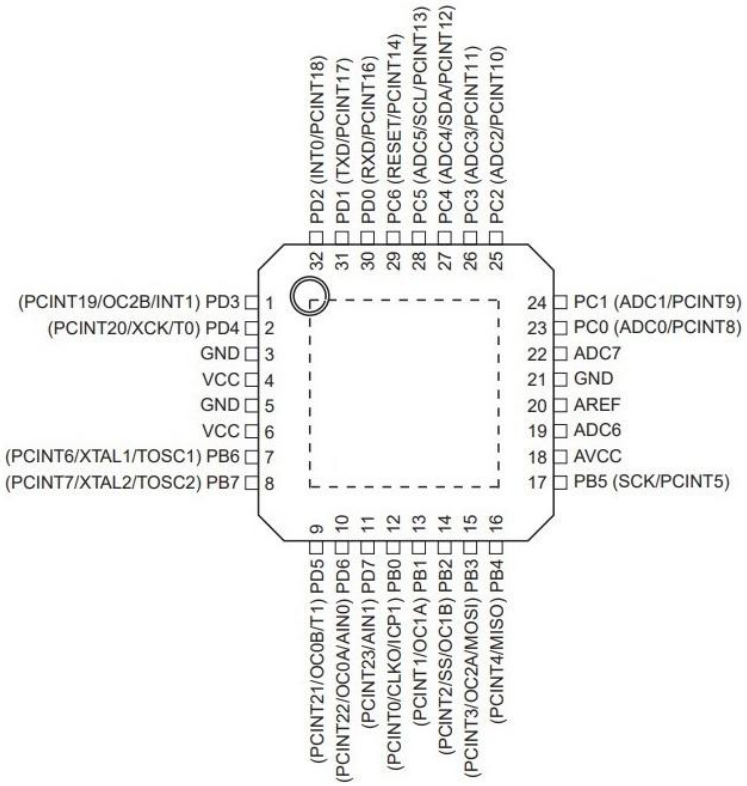

Рис. 2.5. ATmega328

ATtiny85 вирізняється своєю компактністю та низькою вартістю. В порівнянні з ATmega328 має менший об’єм пам’яті (8 кб Flash та 0.5 кб SRAM), меншу кількість портів (6 GPIO, з яких 4 можуть бути аналоговими), лише 2 таймери, не має апаратної підтримки частини протоколів передачі даних та підтримує лише 3 режими сну (Idle, ADC Noise Reduction, Power-down). ATtiny13 в порівнянні з ATtiny85 можна вважати спрощеною версією він має менший обсяг пам’яті (1 кб Flash, 64 б SRAM та 64 б EEPROM) та лише 1 таймер.

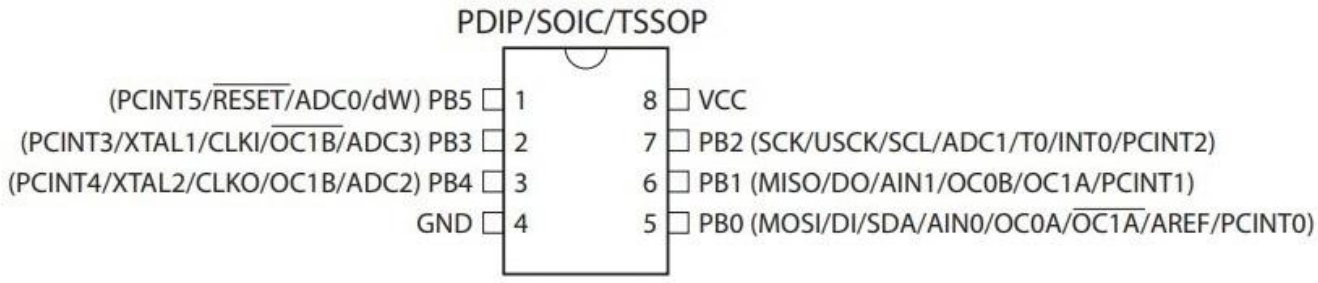

Рис. 2.6. ATtiny85

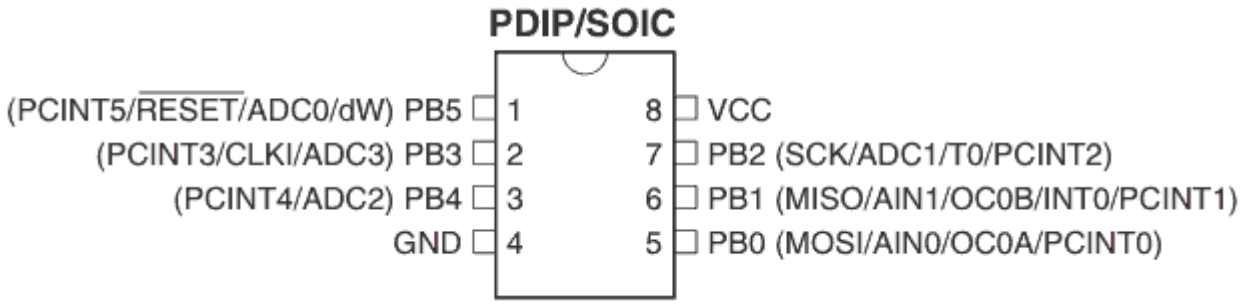

Рис. 2.7. ATtiny13

Таблиця 2.1. Характеристики мікроконтролерів ATmega328 та ATtiny 85/13

Мікроконтролер	Пам’ять, кб			Піни	
	Flash	EEPROM	SRAM	Цифрові	Аналогові
ATmega328	32	1	2	23	8
ATtiny85	8	0.5	0.5	6	4
ATtiny13	1	0.06	0.06	6	4

Таблиця 2.2. Режими сну мікроконтролерів ATmega328 та ATtiny 85/13

Мікроконтролер	Режими сну					
	Idle	ADC Noise Reduction	Power-down	Power-save	Standby	Extended Standby
ATmega328	+	+	+	+	+	+
ATtiny85	+	+	+			
ATtiny13	+	+	+			

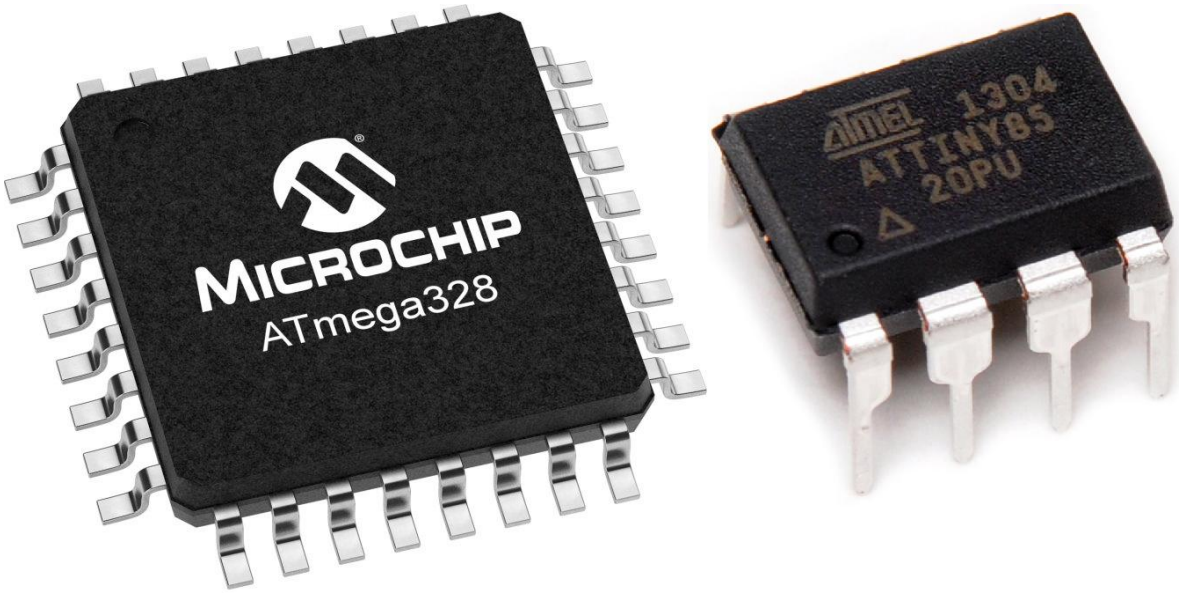

Рис. 2.8. Мікроконтролери ATmega328 та ATtiny85

Мікроконтролери ESP32 та ESP8266, розроблені компанією Espressif Systems, володіють бездротовими можливостями Інтернету речей (IoT). Обидва чипи мають активну спільноту розробників, багато документації та підтримуються різними програмними середовищами розробки, такими як Arduino та MicroPython. Їх спільні характеристики:

- вбудований Wi-Fi з підтримкою мережі 802.11 b/g/n;
- флеш пам’ять до 4 МБ та оперативну до 512 КБ;
- основні інтерфейси передачі даних, такі як UART, SPI, I2C, GPIO та інші;
- кілька режимів енергозбереження: Modem Sleep, Light Sleep, Deep Sleep та Hibernation (тільки ESP32).

ESP8266 відомий своєю компактністю, простотою використання та доступністю. Використовує мікропроцесор Tensilica L106 з тактовою частотою до 80 МГц, підтримує до 17 цифрових входів/виходів загального призначення (GPIO) та має 10-бітний АЦП на одному аналоговому вході.

ESP8266 використовується в складі популярних плат для розробки пристроїв розумного будинку та інтернету речей, наприклад, NodeMCU, де використовується одночасно і для зв’язку по стандарту Wi-Fi, і для виконання програми користувача, таке використання має певні особливості, наприклад,

обробник Wi-Fi повинен викликатись не рідше ніж 20 мс, що дещо зменшує можливість використання аналогових датчиків. ESP8266 також використовується в складі зовнішніх, які з рядом нових умов все ще можна програмувати самостійно, та вбудованих Wi-Fi модулів, де виконанням програми користувача займається інший мікроконтролер. Прикладом такого поєднання є Arduino Uno WiFi, де ESP8266 працює як обробник Wi-Fi, а програму користувача виконує мікроконтролер ATmega328. Зовнішні Wi-Fi модулі можуть підключатись до існуючих рішень через протоколи SPI або UART.

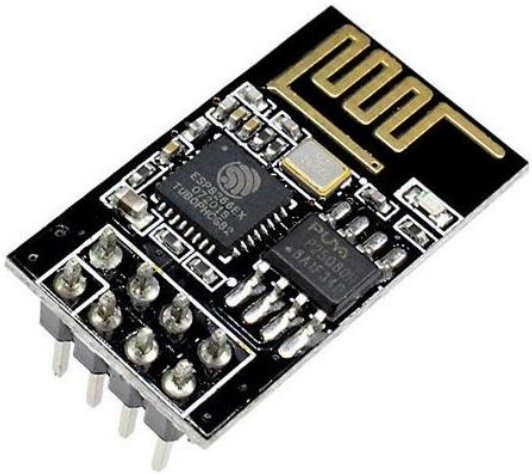

Рис. 2.9. Wi-Fi модуль на базі ESP8266

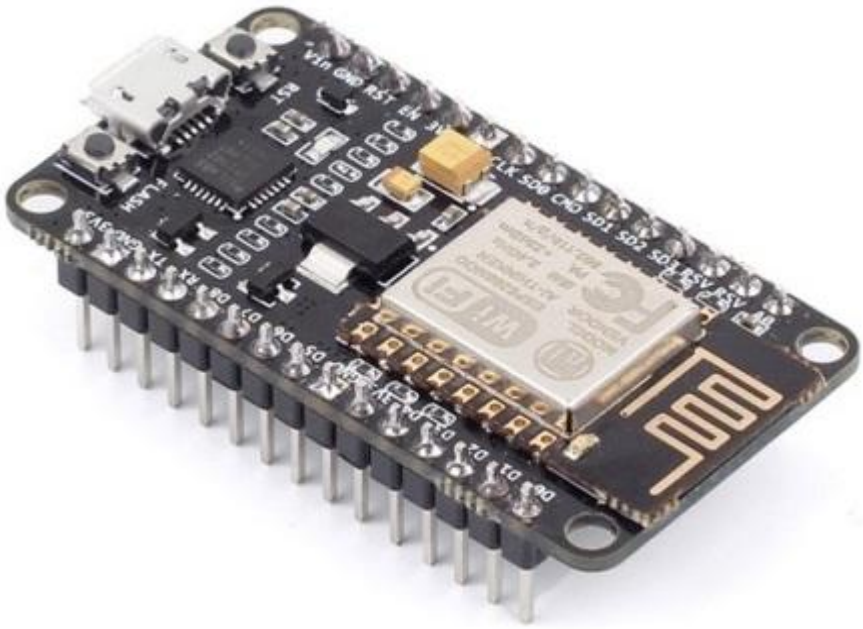

Рис. 2.10. ESP8266 у складі NodeMCU

ESP32 є наступником ESP8266, базується на двоядерному мікропроцесорі Xtensa LX6 з тактовою частотою до 240 МГц, теж розроблений компанією Tensilica. Окрім стандартів Wi-Fi також підтримує Bluetooth стандартів Classic та Low Energy. Підтримує до 34 цифрових входів/виходів загального призначення та має 12-бітний АЦП. Існують варіанти з додатковими інтерфейсами, як наприклад ESP32-CAM, який має 12-мм інтерфейс для підключення камер, що додає популярності у використанні для відповідних проектів

Для забезпечення безпеки мікроконтролери ESP32, на відміну від ESP8266, мають апаратне прискорення для методів шифрування даних, а саме апаратний криптографічний модуль, відомий як ESP32 Security Accelerator (SAES). Він прискорює розрахунки, пов'язані з шифруванням і розшифруванням, а також іншими криптографічними операціями, такими як хешування та генерація випадкових чисел. Підтримуються різні алгоритми шифрування, включаючи AES (Advanced Encryption Standard), DES (Data Encryption Standard), 3DES (Triple Data Encryption Standard), RSA (Rivest-Shamir-Adleman) та ECC (Elliptic Curve Cryptography). Для використання апаратного прискорення шифрування в ESP32 зазвичай використовують спеціальні криптографічні бібліотеки, такі як ESP-IDF (Espressif IoT Development Framework) або Arduino Crypto Library. Ці бібліотеки надають простий інтерфейс для використання апаратного прискорення шифрування на мікроконтролерах ESP32.

Рис. 2.11. Плата ESP32-WROOM

Мікроконтролери STM32 широко використовуються у галузі вбудованих систем і пропонують широкий спектр функціональності та характеристик. Основна мета STM32 це надати високу продуктивність, низьку споживану енергію та багатофункціональність для різних застосувань. Більшість мікроконтролерів STM32 базуються на архітектурі ARM Cortex-M, зокрема Cortex-M0, Cortex-M3, Cortex-M4 або Cortex-M7. Ці архітектури забезпечують високу продуктивність та низьке споживання енергії. Для розробки прошивки мікроконтролерів STM32 використовуються різні середовища, такі як STM32Cube IDE, Keil MDK, IAR Embedded Workbench. Ці середовища надають засоби для написання, компіляції та налагодження коду для STM32.

Також варто згадати популярну серію одноплатних комп'ютерів є Raspberry Pi, що випускаються на основі ARM-процесора. Ці плати мають набагато більше потужностей в порівнянні з розглянутими раніше мікроконтролерами, через що коштують дорожче та мають окреме призначення. Їх звісно можна використовувати для побудови рядових пристроїв системи розумного будинку, проте це майже завжди є недоцільним, оскільки більшість задач потребують ресурсів, що більш ніж цілком забезпечуються простішими мікроконтролерами такими як ATmega328 та ESP32. На основі одноплатних комп'ютерів доцільніше створювати багатофункціональні центральні контролери, оскільки окрім значних

потужностей вони часто оснащені тими ж основними інтерфейсами для передачі даних, що і звичайні персональні комп’ютери.

Рис. 2.12. Raspberry Pi 4 Model B

2.3. Комунікаційні протоколи

Системи розумного будинку та домашньої автоматизації використовують різні комунікаційні протоколи для обміну даними між пристроями. Основні популярні протоколи, що використовуються у таких системах це Wi-Fi (IEEE 802.11), Bluetooth, ZigBee, Z-Wave.

Wi-Fi (Wireless Fidelity, IEEE 802.11) є одним з ключових комунікаційних протоколів, який використовується для бездротового зв'язку між пристроями в розумних будинках та мережах Інтернету речей. Цей протокол забезпечує високу швидкість передачі даних та зручність встановлення підключення [8].

Основним компонентом Wi-Fi є точка доступу, яка дозволяє пристроям безпосередньо підключатись до локальної мережі та отримувати доступ до

Інтернету. Такий тип мережі відповідає топології Star, де пристрої з'єднані з центральним вузлом, але не між собою.

Протокол Wi-Fi базується на стандартах, розроблених Інститутом електротехніки та електроніки (IEEE). Для розумних будинків найпоширенішими стандартами є IEEE 802.11n, 802.11ac та 802.11ax (відомий як Wi-Fi 4, Wi-Fi 5 та Wi-Fi 6 відповідно). Кожен новий стандарт вносить покращення у швидкість передачі даних, пропускну здатність та стабільність підключення. Для роботи використовуються кілька основних частотних діапазони, таких як 2.4, 5 та 6 ГГц. Менша частота забезпечує більше покриття та здатність проникати крізь перешкоди, але має обмежену пропускну здатність, вища забезпечує вищу швидкість передачі даних, але має меншу зону покриття та вищу схильність до затухання сигналу через перешкоди.

Таблиця 2.3. Покоління Wi-Fi

Покоління	Стандарт IEEE	Прийнятий, рік	Максимальна швидкість	Частота
Wi-Fi 7	802.11be	(2024)	≤ 46120	2.4 / 5 / 6
Wi-Fi 6E	802.11ax	2020	≤ 9608	6
Wi-Fi 6	802.11ax	2019	≤ 9608	2.4 / 5
Wi-Fi 5	802.11ac	2014	≤ 6933	5
Wi-Fi 4	802.11n	2008	≤ 600	2.4 / 5
Wi-Fi 3	802.11g	2003	≤ 54	2.4
Wi-Fi 2	802.11a	1999	≤ 54	5
(Wi-Fi 1	802.11b	1999	≤ 11	2.4
Wi-Fi 0	802.11	1997	≤ 2	2.4

Для забезпечення безпеки Wi-Fi протокол використовує різні механізми захисту передачі даних. Один з найпоширеніших методів — це Wi-Fi Protected Access II (WPA2), він забезпечує шифрування даних та аутентифікацію для забезпечення безпеки мережі.

Для шифрування даних, що передаються по мережі WPA2 використовує протокол AES (Advanced Encryption Standard). AES є потужним симетричним алгоритмом шифрування, який забезпечує високий рівень безпеки та використовується в багатьох інших мережах. Операції алгоритму проводяться на блоках фіксованої кількості бітів (128), якщо вхідні дані не кратні бітності блоку, вони доповнюються до потрібного розміру. Алгоритм має кілька прийнятих стандартів довжини ключа, а саме 128, 192 та 256 біт, зазвичай вказується в назві — AES-128. Для довжин ключа було визначено оптимальну кількість раундів шифрування, що забезпечує надійний рівень шифрування та є оптимальним по часу виконання, для 128 біт — це 10 раундів, для 192 — 12, для 256 — 14. До процесу шифрування входять наступні операції:

- SubBytes — нелінійне перетворення, яке замінює кожен байт вхідного блоку на відповідний байт з S-блоку (таблиця заміни, фіксоване значення);
- ShiftRows — перестановка байтів у вхідному блоку. Кожен рядок байтів зсувається на фіксовану кількість позицій вліво;
- MixColumns — лінійне перетворення, яке комбінує кожний стовпець байтів у вхідному блоку, застосовуючи матричні операції;
- AddRoundKey — операція побітового додавання (XOR), в якій вхідний блок комбінується з ключем раунду.

Щодо аутентифікації, WPA2 використовує протокол 802.1X, який дозволяє автентифікувати користувачів, які намагаються підключитися до бездротової мережі. 802.1X базується на протоколі EAP (Extensible Authentication Protocol), який забезпечує безпечний обмін ідентифікаційною інформацією між користувачем і точкою доступу до мережі.

Одним із значних покращень, яке надає WPA2, порівняно з його попередником WEP (Wired Equivalent Privacy), є використання протоколу цілісності тимчасового ключа (TKIP — Temporal Key Integrity Protocol), який забезпечує більш сильний рівень захисту шифрування, додаючи до ключа унікальний вектор ініціалізації для кожного пакета даних, що передається.

Wi-Fi також підтримує різні комунікаційні протоколи, що дозволяють розумним пристроям обмінюватись даними. Наприклад, протоколи TCP/IP (Transmission Control Protocol/Internet Protocol) використовуються для передачі даних через Інтернет, а UDP (User Datagram Protocol) дозволяє швидку передачу даних без необхідності підтвердження отримання.

Bluetooth — це бездротовий комунікаційний протокол, який широко використовується для з'єднання пристроїв в розумних будинках та пристроях Інтернету речей (IoT). Він дозволяє пристроям обмінюватись даними на невеликій відстані, зазвичай до 10 метрів, без необхідності підключення до Інтернету. В розумних будинках це дозволяє створювати мережу зв'язку між цими пристроями та забезпечувати їх взаємодію та контроль через один додаток або платформу [9].

Протокол Bluetooth базується на стандарті Bluetooth Special Interest Group (SIG) і має кілька версій, таких як Bluetooth 4.0, 4.2 та 5.0. Кожна нова версія протоколу вносить покращення у швидкість передачі даних, енергоефективність та забезпечення безпеки.

Bluetooth також має різні режими роботи, наприклад, Bluetooth Classic та Bluetooth Low Energy (BLE). Bluetooth Classic використовується для швидкої передачі великого обсягу даних та аудіо/відео потоків. З іншого боку, Bluetooth LE розроблений для енергоефективної передачі малих обсягів даних і використовується для підключення енергозберігаючих пристроїв, таких як датчики руху або вимикачі.

Однією з переваг Bluetooth є його простота встановлення з'єднання між пристроями. Протокол автоматично виявляє та підключає сумісні пристрої без необхідності ручної настройки. Крім того, Bluetooth підтримує режими з’єднання точка-точка між двома пристроями та створення мережі між кількома пристроями.

З огляду на безпеку, Bluetooth використовує різні механізми шифрування та аутентифікації для захисту передачі даних між пристроями, зокрема часто використовується алгоритм шифрування AES та Secure Simple Pairing (SSP) або Secure Connections (LE SC) для автентифікації. Крім того, існує можливість обмеження доступу до Bluetooth-пристроїв шляхом налаштування паролів, використання ідентифікаторів пристроїв та приховування видимості пристрою від пристроїв, з якими не встановлений зв'язок. Для забезпечення надійності передачі даних протокол Bluetooth має механізми для виявлення та управління інтерференцією, щоб розподілити зв'язок по каналам, що не використовуються іншими пристроями.

Bluetooth модулі, як і Wi-Fi модулі, можуть підключатись до існуючих рішень через протоколи SPI або UART, залежно від типу роботи.

ZigBee є бездротовим комунікаційним протоколом, розробленим для низькопотужних мереж приладів Інтернету речей. Він забезпечує надійний та енергоефективний обмін даними між пристроями [10].

Працює на частотах 2.4 ГГц або 868/915 МГц, що дозволяє залежно від завдань балансувати між пропускною здатністю та радіусом дії. Це особливо корисно для IoT-пристроїв, які передають невеликі обсяги даних.

Основний режим топології мережі — це Сітка (Mesh), проте на його основі можливо реалізувати і інші види топології за потреби. На основній топології варто акцентувати увагу, оскільки це досягається за рахунок того, що кожен пристрій самостійно встановлює з'єднання з іншим пристроєм напряму або шукає шлях до нього через вузли мережі в ролі яких виступають такі ж пристрої, це дозволяє масштабувати систему до великих розмірів, залежно від типу пристроїв, їх число в мережі може варіюватись від 35 до більше 100. Також слід відзначити, що в Mesh топології не обов'язково встановлювати з'єднання всіх пристроїв мережі з центральним пристроєм, його наявність взагалі не обов'язкова, тому можна

організовувати гнучкі підсистеми, розширюючи таким чином всю систему до промислових масштабів. Теоретично, така мережа може налічувати близько 65 тисяч пристроїв, проте реальна кількість обмежується пропускною здатністю, що складає всього 250 кбіт/с (40 на нижчій частоті). Такий спосіб розширення мережі — це ключова відмінність від мереж пристроїв, що використовують Wi-Fi, оскільки там така ж схема роботи неможлива через топологію зірки, де, до того ж, потрібно вручну розміщувати та налаштовувати окремі маршрутизатори, через які і будуть спілкуватись пристрої мережі.

ZigBee, має схожі до інших описаних протоколів вбудовані механізми безпеки для захисту передачі даних. Це шифрування AES-128 для конфіденційності при передачі даних та аутентифікація на рівні мережі. Шифрування теж підтримується на апаратному рівні, що робить його більш ефективним.

Документація по роботі пристроїв ZigBee визначає стек протоколів на різних рівнях OSI, включаючи фізичний, канальний та мережевий рівні. Зокрема надає протоколи верхнього рівня, такі як ZigBee Cluster Library (ZCL) та ZigBee Application Support Layer (ZASL). Ці засоби значно спрощують розробку додатків на основі ZigBee, оскільки вони надають готові рішення для різних типів пристроїв та функціональності. Розробники можуть скористатися цими інструментами для швидкого створення додатків, забезпечення сумісності та прискорення впровадження систем IoT на основі ZigBee.

ZCL є набором стандартних класів та кластерів, які визначають типові функції та можливості пристроїв ZigBee. Кластери описують різні аспекти функціональності пристрою, наприклад, освітлення, контроль температури, безпека, сенсори тощо. Кожен клас та кластер визначає команди, атрибути та профілі для взаємодії між пристроями в мережі ZigBee. За допомогою ZCL розробники можуть створювати додатки для різних пристроїв, використовуючи готові класи та кластери, що спрощує розробку і забезпечує сумісність між пристроями різних виробників.

ZASL забезпечує додаткову підтримку для розробки додатків на основі ZigBee. Він надає різні сервіси та функції, які спрощують взаємодію додатків зі

стеком протоколів ZigBee. Наприклад, ZASL забезпечує можливість відправляти та отримувати повідомлення, керувати класами та кластерами, обробляти події та помилки. Він також надає API, які дозволяють розробникам просто використовувати функціональність ZigBee для своїх додатків.

Z-Wave — це ще один бездротовий протокол для мережі розумних пристроїв. Він розроблявся як аналог ZigBee, тому має ряд спільних деталей і, загалом, він теж забезпечує надійну та енергоефективну комунікацію між пристроями. Відмінністю від інших розглянутих протоколів є нижча частота передачі, що збільшує радіус дії та енергоефективність. Частота роботи регулюється місцевим законодавством в межах 800–900 МГц, для України та країн Європи використовується частота 868.42 МГц. ZigBee теж може працювати на схожих частотах, проте Z-Wave робить це ефективніше. Можливість роботи на нижчій частоті для обох протоколів дає не тільки більшу зону покриття та енергоефективність, але і більшу стабільність сигналу, оскільки приладів, які працюють на частоті 800–900 МГц значно менше ніж на частоті 2.4 ГГц [11].

Оскільки Z-Wave розроблявся як аналог ZigBee, більшість речей в них по принципу роботи схожі, тому можна коротко охарактеризувати протокол акцентувавши увагу на відмінностях протоколів:

- швидкість передачі складає 100 кбіт/с, тоді коли ZigBee на близькій частоті має 40 кбіт/с;
- тип топології — сітка, проте кількість вузлів між пристроями обмежена і складає 4 вузла, що обмежує кількість пристроїв до 232, ZigBee не обмежує кількість вузлів, а максимально можлива кількість пристроїв складає 65000;
- має більшу кількість сумісних пристроїв;
- існує специфікація, що має в 4 рази більшу область покриття від звичайної — Z-Wave Long Range (LR);

- Z-Wave — це запатентована технологія, що належить компанії Sigma Design, вони розвивають технологію та керують альянсом Z-Wave, який контролює сертифікацію приладів, що передбачає їх сумісність між собою, ZigBee має відкритий вихідний код, має менші вимоги до сумісності приладів та підтримує міжнародний стандарт IEEE 802.15.4.

В інших аспектах, як наприклад безпека, вони схожі або однакові.

На основі опрацьованих матеріалів можна зробити коротку таблицю основних характеристик розглянутих протоколів.

Таблиця 2.4. Порівняння безпровідних протоколів

Характеристика	Bluetooth	Wi-Fi	ZigBee	Z-Wave
Частота	2.4 ГГц	2.4 ГГц	2.4 ГГц, 868/915 МГц	868/915 МГц
Топологія	Точка-точка, зірка	Зірка	Сітка	Сітка, до 4-х вузлів між пристроями
Теоретична макс. кількість пристроїв	До 8, якщо зірка	255 / 1000+	65000	232
Швидкість	125 Кбіт/с – 24 Мбіт/с	Залежно від покоління, див. табл. 2.3	100 Кбіт/с	250 / 40 кбіт/с
Безпека	AES-128, SPP / LE SC	WPA2 / WPA3 (EAP, AES-128)	AES-128, SEP 1.1	AES-128, DH / ECC

РОЗДІЛ ІІІ. РОЗРОБКА МОДУЛЬНОЇ СИСТЕМИ

3.1. Життєвий цикл

Для стандартизації процесів розробки програмних продуктів, до яких також відносяться системи розумного будинку та суміжні речі, існують міжнародні стандарти, наприклад стандарти Міжнародної організації зі стандартизації (ISO).

Актуальним та основним стандартом, що описує життєвий цикл розробки програмних продуктів, є ISO/IEC 12207:2017 Systems and software engineering — Software life cycle processes (Системна та програмна інженерія — Процеси життєвого циклу програмних продуктів).

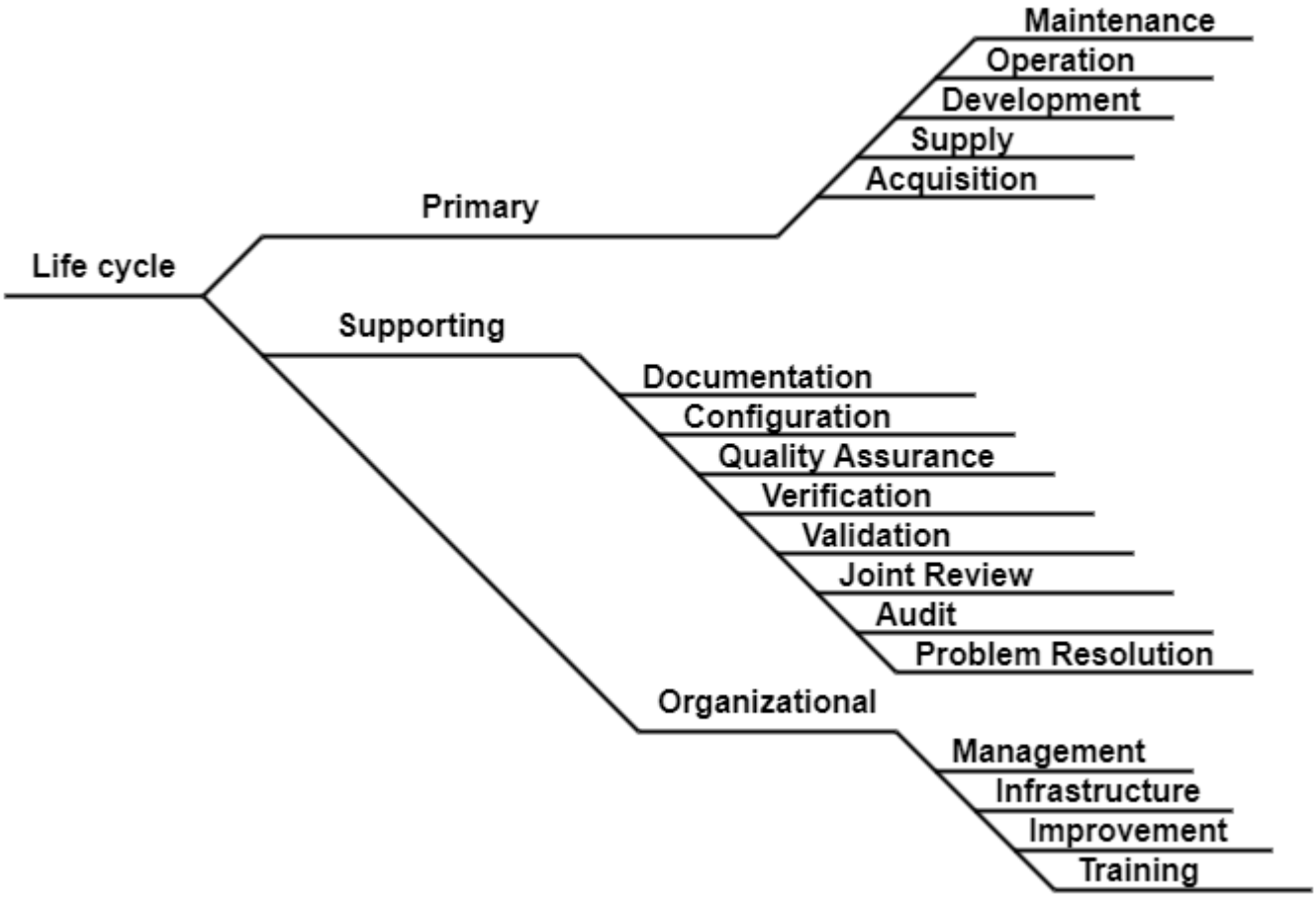

Рис. 3.1. Узагальнений поділ процесів розробки згідно стандарту ISO/IEC 12207

Оскільки поняття "програмний продукт" заключає в собі найрізноманітніші можливі проекти, то цей стандарт надає лише загальний огляд процесів життєвого циклу програмного продукту та не є покроковою інструкцією. Організація кожного

процесу та етапу залежить від конкретного проекту, включаючи пропуск частини процесів, їх інтеграцію в інші процеси чи їх повне перепрацювання.

Основні етапи та процеси, визначені цим стандартом:

- Планування програмного забезпечення.
- Визначення вимог.
- Проектування програмного забезпечення.
- Реалізація програмного забезпечення.
- Верифікація та перевірка.
- Введення в експлуатацію.
- Експлуатація та підтримка.
- Вилучення програмного забезпечення.

Процес планування програмного забезпечення є першим етапом у життєвому циклі розробки програмного продукту. Його метою є визначення стратегії, обсягу, ресурсів, графіка та завдань, необхідних для створення цілісного та реалістичного плану, який дозволяє ефективно керувати проектом розробки програмного забезпечення.

Основні кроки процесу планування:

1. Визначення мети проекту.
2. Встановлення попереднього обсягу робіт.
3. Оцінка ресурсів необхідних для виконання проекту.
4. Розробка графіка робіт.
5. Визначення завдань та розподіл роботи.
6. Управління ризиками.
7. Визначення стратегії контролю якості.
8. Визначення комунікаційної стратегії — звітність, засоби комунікації між учасниками проекту.
9. Визначення бюджету.
10. Документування плану.

Процес визначення вимог є критичним етапом у життєвому циклі розробки програмного забезпечення. Його метою є збір, аналіз та документування вимог до програмного продукту, які встановлюють що програмний продукт має робити і його функціональні та нефункціональні характеристики, що мають бути задоволені.

Процес проектування (Software Design) є одним з ключових етапів у життєвому циклі розробки програмного продукту. Він вимагає глибокого розуміння вимог до програмного продукту. Його метою є визначення архітектурної структури та деталей програмного продукту, які забезпечують виконання вимог, визначених у процесі визначення вимог.

Основні кроки процесу проектування:

1. Архітектурне проектування — визначення загальної структури.
2. Детальне проектування — уточнення архітектурних елементів шляхом розробки детальних дизайнів компонентів.
3. Вибір технологій та інструментів.
4. Розробка інтерфейсів.
5. Документування дизайну.

Процес конструювання (Software Construction) є етапом реалізації програмного продукту на основі визначених вимог і проекту. Його метою є перетворення дизайну програмного продукту на функціонуючий код, включаючи часткове тестування та внесення корекцій в дизайн та вимоги.

Процес тестування та верифікації програмного забезпечення відіграє важливу роль у забезпеченні якості та відповідності програмного продукту до вимог та специфікацій. Його метою є перевірка та підтвердження, що програмне забезпечення працює правильно, виконує задані функції та задовольняє очікування користувачів. Вибір методології та інструментів тестування проводиться на основі визначених вимог до програмного продукту.

Останнім етапом розробки програмного продукту є етап його розгортання. Він включає в себе процеси, що пов'язані з введенням програмного продукту в

експлуатацію та розповсюдження серед користувачів. Основною метою цього етапу є забезпечення успішного впровадження програмного продукту, що є критично важливим для більшості проектів. Зміст етапу залежить від типу проекту.

Після впровадження продукту існує ще 2 етапи, що варіюються залежно від проекту: підтримка проекту під час експлуатації та виведення проекту з експлуатації. Їх основна мета, як і в інших етапах, відповідає назві, а зміст напряму залежить від типу та змісту проекту.

Крім опису основних процесів, стандарт ISO/IEC 12207:2017 також окремо описує додаткові процеси, такі як управління якістю, управління ризиками, управління конфігураціями та інші.

3.2. Визначення вимог

Визначення вимог до системи розумного будинку є важливим кроком у процесі розробки системи. Вимоги допомагають визначити, які функції повинна виконувати система, які характеристики має мати технологія, а також рівень безпеки, зручності використання та інтеграції з іншими системами. Основні категорії вимог до системи розумного будинку можуть включати:

- функціональні вимоги;
- вимоги до зручності використання;
- вимоги до безпеки;
- вимоги до інтеграції;
- вимоги до масштабованості.

В контексті розроблюваної модульної системи спостереження за параметрами будуть ігноруватись вимоги до інтеграції, оскільки все ж така система не є повноцінною системою розумного будинку, а виступає швидше її аналогом для завдань, де готові рішення, з якими і повинна бути інтеграція, не підходять і є необхідність проектування власних сенсорів та актуаторів.

Функціональні вимоги до модульної системи:

- Зв’язок:

    - швидке налаштування мережі зв’язку;
    - аналіз мережі на чистоту каналів зв’язку;
    - захист від часткової чи повної втрати даних внаслідок перешкод на радіо каналі.
- Дані:
    - зберігання отриманих даних;
    - демонстрація даних користувачу та стороннім пристроям;
    - аналіз даних.
- Наявність налаштовуваного режиму енергоефективної роботи.
- Зовнішній інтерфейс для налаштування системи.

Вимоги до зручності використання:

- Інтуїтивно зрозуміла взаємодія користувача з компонентами системи через відповідні інтерфейси.
- Легка та швидка розробка пристроїв мережі.
- Легка модифікація центрального пристрою.

Вимоги до безпеки:

- Шифрування даних

Інші вимоги:

- Використання легко доступних компонентів

3.3. Підбір компонентів

Завдання цієї роботи передбачає створення центрального контролера та підбір програмних елементів для швидкого розгортання пристроїв мережі.

Ключові аспекти:

- мікроконтролер;
- зв’язок;
- збереження даних;
- допоміжні інтерфейси взаємодії.

Серед доступних та функціональних мікроконтролерів в попередніх розділах серед інших були виділені 2 ключових мікроконтролери серії ESP та кілька архітектури AVR. Для побудови центрального контролера було обрано мікроконтролер архітектури AVR — ATmega328, оскільки є можливість після завершення розробки спроектувати на чистому мікроконтролері кінцевий пристрій, а на момент розробки використовувати готові платформи для розробників, зокрема Arduino Nano та Uno, на яких і буде вестись розробка.

Оскільки цей мікроконтролер не має вбудованих інтерфейсів мережевого зв'язку необхідно доповнити його відповідним модулем. Класичні Wi-Fi модулі на ESP8266 мають малий радіус дії та погану проникність. Для розгортання мереж, що не потребують інтернет з'єднання існує модуль радіозв'язку від компанії Nordic Semiconductor — Nrf24l01+, який працює на частоті 2.4 ГГц та має версію з підсилювачем (Nrf24l01+ PA LNA), це один з кращих модулів двосмугового радіозв'язку, що пропонує засоби для підйому якісної і надійної мережі та має доступну ціну. Версія з підсилювачем має хорошу проникність сигналу, як для такої частоти роботи, або радіус дії більше 1 км при прямій видимості. Модуль працює по протоколу SPI [13].

Вбудованої пам'яті мікроконтролера вистачить для збереження актуальної інформації з пристроїв мережі, проте для статистичної інформації потрібне додаткове сховище. Для цих цілей можна використати модуль читання та запису SD-карт, що дозволить зберігати інформацію про роботу мережі в зручних форматах даних, які можуть бути записані без використання спеціалізованих середовищ, наприклад, таблиці Excel. Цей модуль теж працює по протоколу SPI.

Для можливості координації мережі залежно від часу доби та прив'язки до збережених даних до часу необхідно використовувати сторонній модуль реального часу, оскільки тактування мікроконтролера не є точною мірою, окрім того відсутня можливість рахувати актуальну дату, поки пристрій вимкнений. Точність досягається за рахунок корекції значення, отриманого від тактування залежно від температури модуля, оскільки вона напряму впливає на тривалість одного такту. Існують кілька популярних модулів реального часу, з поміж них виділяється

модуль DS3231, що має похибку ±2 хвилини на рік. Модуль використовує протокол I2C.

Додатково центральний контролер може мати дисплей та елементи керування. Більшість дисплеїв мають бібліотеки, що розбивають площу на рядки, тому взаємодія з ними майже не відрізняється, єдиним критерієм вибору є можливість підключення по протоколу I2C. В якості елемента керування виступить енкодер, що може зчитувати напрям та натиснення, чого вистачить для побудови зручного, не перевантаженого керування.

3.4. Побудова схеми та концепту функціоналу

Компоненти приладу:

- Arduino Nano на базі ATmega328
- NRF24L01 — зв'язок, SPI
- DS3231 — модуль реального часу, I2C
- Модуль читання-запису SD карт пам'яті, SPI
- Рідкокристалічний дисплей з інтерфейсом I2C
- Енкодер

Схема ATmega328 була розглянута раніше, на рис. 2.5, схема виходів плати Arduino Nano, що побудована на цьому мікроконтролері [12], зображена на рис. 3.2.

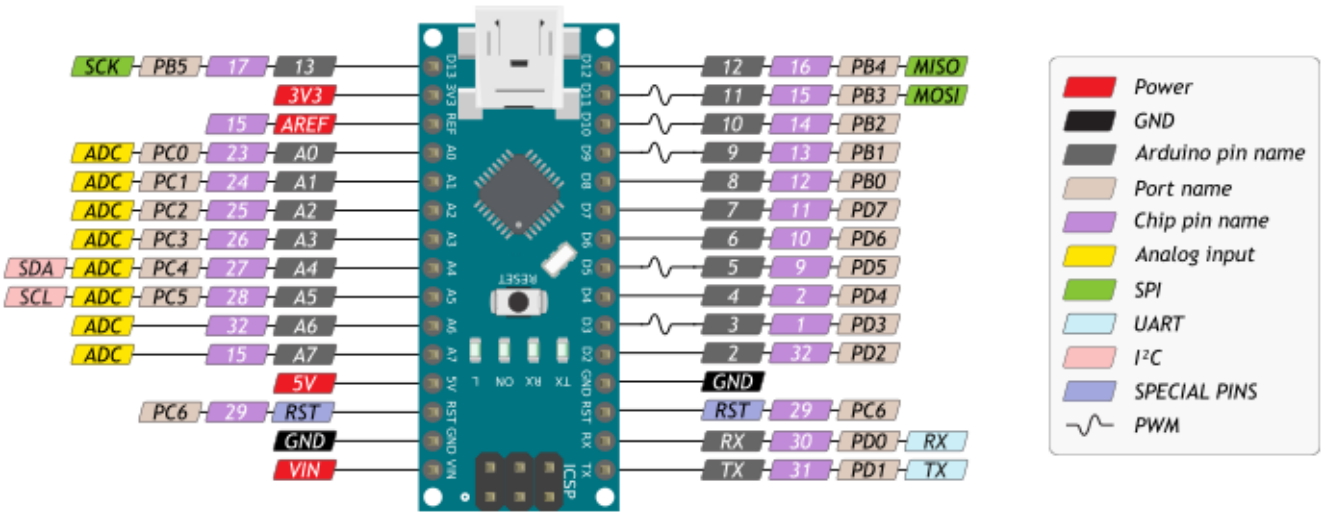

Рис. 3.2. Arduino Nano

Протокол SPI в мікроконтролері ATmega328 реалізований на цифрових портах 11, 12 та 13, що відповідає значенням MISO, MOSI та SCK відповідно. Додатково для кожного пристрою обирається порт для його контролю. Окрім того радіомодуль може генерувати сигнал по певним подіям, наприклад, отримання даних, що можна використовувати для зовнішнього переривання, мікроконтролер має 2 зовнішніх переривання на цифрових портах 2 та 3 [7]. Проте в роботі це не задіяно через специфіку сценарію з головним пристроєм в ролі приймача, тому ця деталь залишається лише на схемі, як можливе розширення.

Інтерфейс I2C, реалізовано на портах, що відповідають 4-му та 5-му аналоговим портам, це значення SDA та SCL. Цей протокол працює по адресам пристроїв, тому додаткові окремі підключення, як в SPI, не потрібні.

Напрям обертання енкодера визначається порівнянням логічних значень на його виходах, що зазвичай позначаються як A та B, додатково він має вихід для відстеження натиснень, його теж можна використати для переривань.

Враховуючи описані деталі була складена електронна схема, що зображена на рис. 3.3.

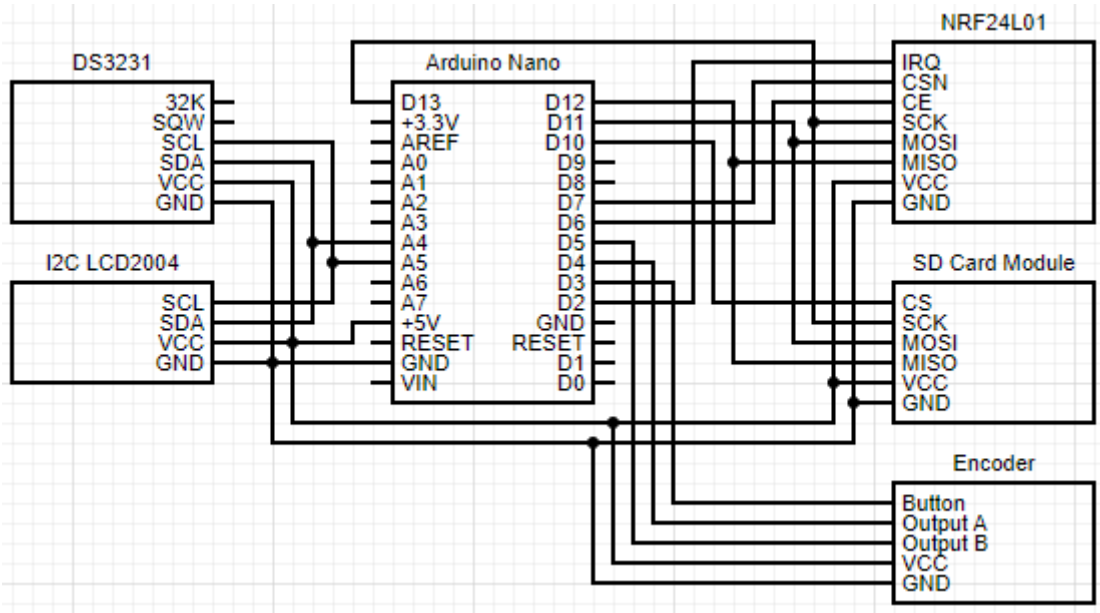

Рис. 3.3. Схема центрального контролера

Використаний RF модуль повинен працювати від 3.3 В з стабілізатором напруги в електричному колі, проте в роботі розглядається версія з підсилювачем та вбудованим стабілізатором, який дозволяє підключати модуль напряму до

спільної лінії живлення +5 В, через що розведення схеми при складанні приладу дещо спрощується.

Щодо функціоналу розроблюваної модульної системи спостереження за параметрами середовища, то буде реалізовано такий функціонал:

- Надсилання інструкцій периферійним пристроям.
- Вивід оброблених даних на екран пристрою та в послідовний порт.
- Зберігання даних на SD карті.
- Зберігання внутрішніх налаштувань в енергонезалежній пам’яті.
- Інтерфейс взаємодії з користувачем
- Налаштування приладу через спеціальний додаток

3.5. Реалізація функціоналу

Весь функціонал, що повинен бути реалізований можна умовно поділити на кілька категорій, а саме на:

- зв’язок;
- збереження даних;
- система налаштувань;
- інші компоненти та функції.

Зв’язок.

Для зв’язку був обраний радіомодуль nrf24l01+. Як вже було зазначено, він працює на частоті 2.4 ГГц, на одному з 127 радіоканалів, який містить до 6 виділених каналів даних, з якими може працювати почергово, проте можливий перезапуск мережі з іншим набором адрес каналів даних. Радіоканали поділені по частоті, перший канал має частоту 2401 МГц, другий 2402, а останній, 127-й, 2527 МГц. Канали даних мають 5-ти байтову адресу, де спільні перші 4 байти, наприклад: “node1”, “node2”.

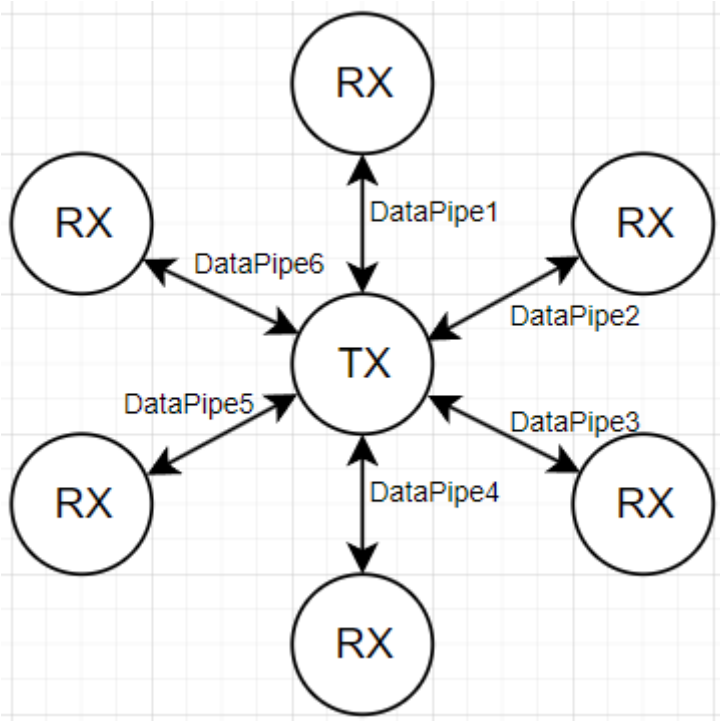

Рис. 3.4. Схема роботи мережі Nrf24l01

Інші характеристики модуля:

- швидкість передачі: 0.25, 1, 2 Мб/с;
- тип модуляції: GFSK (Гаусівська частотна модуляція);
- мінімальний / максимальний струми: 26 мкА / 13.5 мА

Оскільки цей модуль в режимі приймача не може окремо обробляти кожен відкритий канал, передбачається побудова на його основі мережі зіркоподібної топології, де центральний пристрій працює в режимі передачі та опитує периферійні пристрої, що працюють в режимі прийому [13].

В модульній системі спостереження за параметрами середовища передбачається, що периферійний пристрій зможе по готовності передавати до центрального контролера дані, а сам центральний контролер повинен надсилати інструкції актуаторам. Таким чином існує кілька сценаріїв побудови такої системи: центральний контролер працює в режимі прийому, та за потреби змінює режим на передачу або постійно активний режим передачі.

Якщо розглядати схему, де центральний пристрій працює в ролі приймача, то є певна вірогідність, що два чи більше пристроїв відправлять дані центральному

пристрою одночасно та / або поки вже йде прийом, що викличе інтерференцію та втрату даних, що відправлялися в цей момент.

Інший сценарій, за якого можливо організувати правильну почергову взаємодію периферійних пристроїв — це постійне почергове опитування всіх пристроїв. В такому випадку, якщо сенсор готовий відправити дані, він вмикає радіомодуль в режим прослуховування та, коли отримає запит, відправить вже підготований пакет даних. Основним недоліком такого алгоритму є затримка отримання даних від сенсорів, що реагують на раптові події, проте, за умови якісного зв'язку, діалог між парою пристроїв не перевищує 2 мс, таким чином варто очікувати максимальну затримку реакції в кілька сотень мс, більш серйозні затримки можуть виникнути у зв'язку з виконанням центральним контролером якихось операцій по запису, обробці або читанню даних. Іншим недоліком є необхідність постійно опитувати сенсори, через що головний пристрій повинен бути активним, в той час як при першому сценарію, головний пристрій може перейти до режиму сну, а пробудитись за допомогою сигналу переривання, що генерує nrf24l01 на порті IRQ, коли отримано дані, проте, пристрій на чистому мікроконтролері споживає незначні значення струму, в порівнянні з робочим RF приймачем, тому економія може скласти всього 30–40 %, в той час як периферійні пристрої, вимикаючи свої передавачі зможуть за допомогою сну економити більш ніж 90% енергії.

Опрацювавши всі плюси та недоліки обох сценаріїв роботи, другий, в якому центральний контролер працює постійно в режимі передавача, обрано як першочерговий до розробки.

Перед розробкою алгоритму варто розглянути додаткові особливості модулю по способу передачі даних, а саме на структуру пакету даних і функції підтвердження отримання та перевірки цілісності даних.

Щоб підтвердити отримання даних, модуль, що прийняв та перевірив їх, через 130 мкс автоматично відправляє повідомлення про підтвердження, якщо повідомлення не отримано, відправник спробує ще раз [13]. Кількість спроб та

наявність підтвердження налаштовується. Це повідомлення позначається як Ack, що є скороченням від acknowledgment — підтвердження.

Для підтвердження цілісності даних використовується контрольна сума, підрахована алгоритмом циклічного надлишкового коду, CRC. Контрольна сума додається до пакету даних, при отриманні підраховується повторно та порівнюється з надісланою. Шанс, що інтерференція призведе до порушення даних таким чином, що їх контрольна сума не зміниться близький до нуля, тому цей алгоритм має високу надійність. Алгоритм базується на математичних операціях, таких як ділення з відштовхуванням.

Кроки алгоритму CRC:

1. Дані розбиваються на біти та обробляються у вигляді бінарних значень.
2. Якщо необхідно, до даних додаються нульові біти для забезпечення відповідної довжини для обчислення контрольної суми.
3. Дані діляться на поліном CRC, це виконується шляхом виконання побітової операції XOR між даними та поліномом.
4. Остача від ділення стає контрольною сумою CRC.

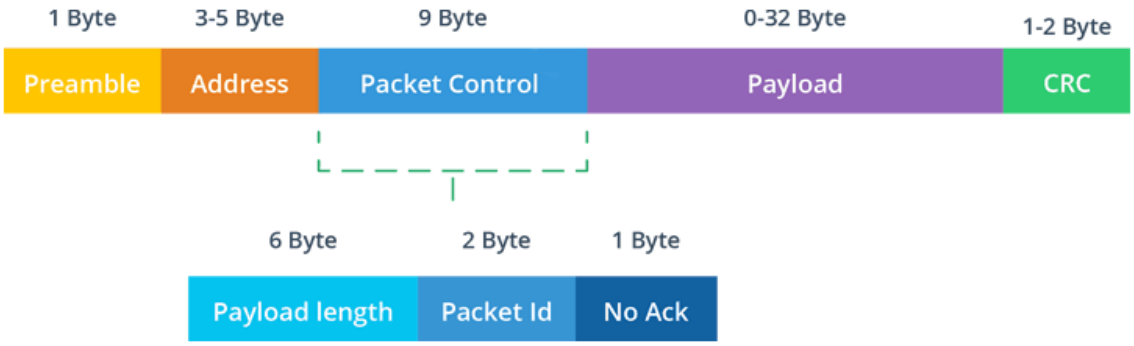

Рис. 3.5. Структура пакету даних, що відправляє Nrf24l01

Структура пакету даних (рис. 3.5) включає в себе:

- преамбулу, необхідну для налаштування приймача;
- адресу отримувача, що повинен обробити наступні частини пакету;
- контрольну інформацію, що зокрема містить вказівки про відправку підтвердження отримання;

- корисне навантаження — дані, що передаються;
- контрольну суму.

Далі необхідно розглянути програмну взаємодія мікроконтролера з RF модулем. Вона, як і взаємодія з іншими модулями, відбувається через команди регістру, проте для зручності програмування використовуються спеціалізовані бібліотеки, зокрема бібліотеки “nRF24L01” та “RF24” (функціонують в парі). Наприклад, ввімкнення відповіді з корисним навантаженням за допомогою регістру виглядає наступним чином:

```
write_register(FEATURE, read_register(FEATURE) | _BV(EN_ACK_PAY) |
    _BV(EN_DPL));
write_register(DYNPD, read_register(DYNPD) | _BV(DPL_P1) | _BV(DPL_P0));
dynamic_payloads_enabled = true;
ack_payloads_enabled = true;
```

Натомість бібліотека дозволяє використовувати звичні методи для роботи з модулями.

Основна взаємодія з радіомодулем за допомогою цих бібліотек відбувається через наступні методи:

- setAutoAck(bool) — підтвердження отримання, присутнє перевантаження методу з вказанням потрібного каналу даних перед значенням
- enableAckPayload(void) / disableAckPayload(void) — можливість відправляти, разом з повідомленням про отримання, пакету даних у відповідь, не працює на мінімальній швидкості передачі
- setPayloadSize(uint8_t) — розмір пакету в байтах, не більше 32
- openWritingPipe(uint64_t) — відкрити канал даних для передачі за адресою
- openReadingPipe(uint8_t, uint64_t) — відкриття каналу для отримання даних за адресою
- setChannel(uint8_t) — встановити канал зв’язку
- setPALevel(uint8_t) — рівень потужності: MIN, LOW, HIGH, MAX
- setDataRate(rf24_datarate_e) — швидкість обміну: 2MBPS, 1MBPS, 250KBPS
- powerUp(void) — подача живлення, початок роботи

- stopListening(void) — зупинка прослуховування радіоефіру
- startListening(void) — початок прослуховування відкритих каналів
- available(void) — чи отримано значення, дозволяє зберегти адресу відправника, якщо вказати адресу змінної
- read(void*, uint8_t) / write(void*, uint8_t) — зчитати отримані дані / відправити дані, необхідно вказувати вказівник на адресу даних та їх розмір
- writeAckPayload(uint8_t, const void*, uint8_t) — відправка даних разом з підтвердженням отримання, додатково до звичайних параметрів write необхідно вказати адресу отримувача

Наприклад, для відправки повідомлення потрібно створити об’єкт класу, вказавши цифрові порти, до яких підключені піни CE та CSN у такому ж порядку. Далі необхідно почати роботу з модулем методом begin, встановити потрібний канал, відкрити адресу для запису та відправити повідомлення, вказавши його адресу в оперативній пам’яті та його розмір:

```
const byte address[6] = "node1";
const char text[] = "msg";
RF24 radio(6, 7);

radio.begin();
radio.setChannel(0x5a);
radio.openWritingPipe(address);
radio.write(&text, sizeof(text));
```

Для отримання цього повідомлення, замість відкриття адреси для запису відкрити її для читання та почати прослуховувати мережу, якщо щось було отримано, ці дані можна буде прочитати з пам’яті RF модуля та зберегти їх за адресою потрібної змінної:

```
radio.openReadingPipe(0, address);
radio.startListening();
if (radio.available()) {
   char text[32] = {0};
   radio.read(&text, sizeof(text));
}
```

Для опитування периферійних пристроїв зручно використовувати метод відправки даних разом з підтвердженням отримання даних. Це відбувається так само як і відправка даних, але методом writeAckPayload по адресі відправника. Відправник, щоб зчитати вміст відповіді, повинен після відправки використати метод для читання. Таким чином діалог між головним та периферійним пристроєм виглядає наступним чином: надсилається запит на отримання даних, пристрій приймає його та разом з повідомленням про підтвердження надсилає підготовані раніше дані.

Одна з функцій, що повинна бути в розроблюваній модульній системі, це сканер радіоканалів, щоб можна було налаштувати систему на чистий канал, що не використовується іншими пристроями. Для цього необхідно певний час прослуховувати кожен канал певний час та записувати чи було щось отримано , повторити це кілька разів, в прикладі 100 та вивести отриманий результат. Наприклад, наступна функція може сканувати за виклик всі канали та виводити в послідовний порт результат:

```
const uint8_t num_channels = 128;
uint8_t values[num_channels];
memset(values, 0, sizeof(values));
int rep_counter = 100;
while (rep_counter--) {
  int i = num_channels;
  while (i--) {
    radio.setChannel(i);
    radio.startListening();
    delayMicroseconds(128);
    radio.stopListening();
    if ( radio.testCarrier() )
      ++values[i];
  }
}
int i = 0;
while ( i < num_channels ) {
  printf("%x", min(0xf, values[i] & 0xf));
  ++i;
}
printf("\n\r");
```

Приклад роботи цієї функції зображено на рис. 3.6. По отриманим результатам можна зробити висновок, що пристрої, присутні поряд з макетом центрального контролера, використовують активно канали з шістнадцятковими адресами від 0x2b по 0x3d та періодично канали 0x0b, 0x0c та 0x0d. Обравши для роботи чисті канали буде знижено шанс інтерференції, а відповідно зросте надійність розгорнутої мережі.

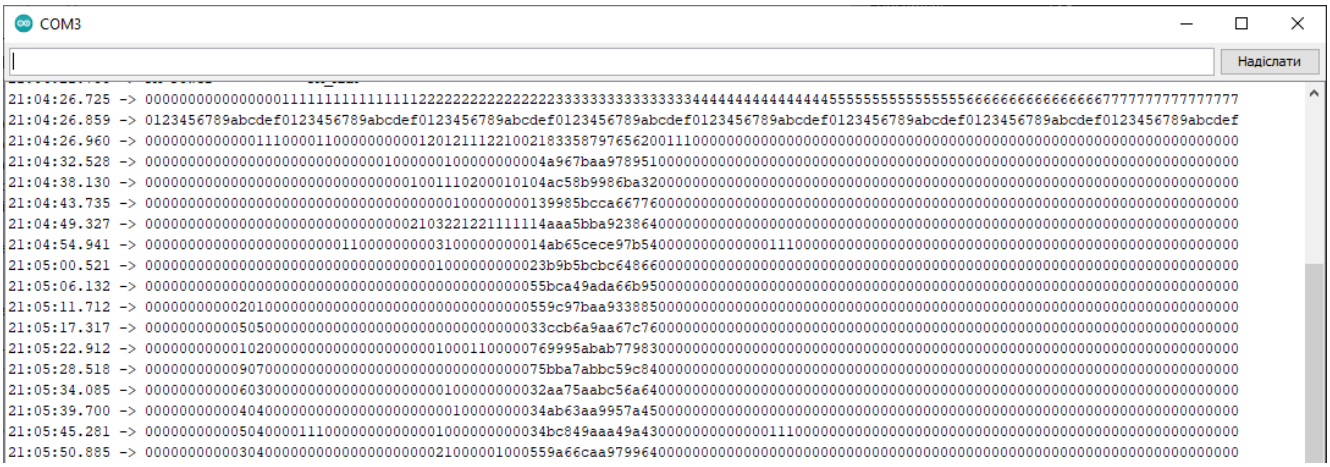

Рис. 3.6. Сканування мережі за допомогою NRF24L01

Стосовно зв’язку залишилась одна не розглянута деталь — безпека. Розглянуті в розділі 2.3. комунікаційні протоколи мають алгоритми шифрування для передачі даних та методи для авторизації. Алгоритм шифрування виявився спільним для всіх мереж, це AES-128, відносно складний алгоритм з точки зору часу, необхідного для шифрування та дешифрування, це пов’язано з кількістю операцій, як було встановлено, в 128-ми бітній версії це послідовно по 4 операції шифрування 10 раундів підряд. Мікроконтролери, що його використовують, для цих цілей мають модулі апаратного прискорення, тому чудово можуть використовувати цей алгоритм, розроблювана ж модульна система використовує мікроконтролер архітектури AVR, який апаратного прискорення не має, проте, враховуючи, що периферійні пристрої повинні підготувати пакет даних до відправки та чекати певний проміжок часу, то цей алгоритм цілком має право на використання.

Для спрощення парсингу отриманих даних, дані передаються рядком повним блоком в 32 байти, що якраз відповідає двом шифрованим блокам. Блок

містить пари через кому ідентифікатор:значення, читання закінчується на останньому символі або якщо зчитано пробіл, тому дані не повинні його містити.

Якщо система модифікується під швидкодію і шифрування залишається важливим, цей алгоритм може бути замінений на асиметричний RSA, що виконується за один раунд, проте з витратою певного часу на підбір параметрів, що задовільнять формули алгоритму.

Отже, основні операції шифрування, що детальніше були описані в розділі 2.3., можуть мати такий вигляд та реалізацію:

- subBytes:

```
for(byte i = 0; i < 16; i++) {
   state[i] = pgm_read_byte(Sbox + state[i]);
}
```

- shiftRows:

```
byte temp;
temp = state[1]; state[1] = state[5]; state[5] = state[9];
state[9] = state[13]; state[13] = temp;
temp = state[10]; state[10] = state[2]; state[2] = temp;
temp = state[14]; state[14] = state[6]; state[6] = temp;
temp = state[3]; state[3] = state[15]; state[15] = state[11];
state[11] = state[7]; state[7] = temp;
```

- mixColumns:

```
byte i, a, b, c, d, e;
for(i = 0; i < 16; i+=4) {
   a = state[i]; b = state[i+1]; c = state[i+2]; d = state[i+3];
   e = a ^ b ^ c ^ d;
   state[i]   ^= e ^ xtime(a^b);
   state[i+1] ^= e ^ xtime(b^c);
   state[i+2] ^= e ^ xtime(c^d);
   state[i+3] ^= e ^ xtime(d^a);
}
```

- addRoundKey:

```
for(byte i=0; i < 16; i++) {
   state[i] ^= key[i];
}
```

Алгоритм також містить інші функції та параметри, зокрема робота з ключем шифрування, статичні таблиці заміни та допоміжні функції для роботи з ними, детальніше в Додатку А.

Збереження даних.

Під збереженням даних розуміється систематизований збір та зберігання переданих параметрів від периферійних пристроїв, інформацію про периферійні пристрої та список інструкцій, що центральний пристрій може надсилати периферійним пристроям.

Мікроконтролер ATmega328 має 3 типи пам'яті, з яких енергонезалежною та призначеною для використання під час виконання програми є EEPROM пам'ять. Вона має розміри в 1 кб, що не надто багато для збереження всіх даних, крім того ця пам'ять має обмежену кількість перезаписів, виробник гарантує 100 тис. циклів, проте за нормальних температур роботи реальна кількість перезаписів перевищує 3 млн. Такого значення вистачить для оновлення даних кожну хвилину протягом більш ніж 5-ти років.

Проте, через відносно малий об'єм пам'яті, EEPROM не підходить для збереження статистичних масивів даних, а оскільки ця функція необхідна для розроблюваної системи, то до її складу було включено модуль для читання та запису SD карт пам'яті.

Таким чином присутні 2 способи збереження даних, хорошим підходом буде розділити дані по типам та зберігати окремо. Оскільки в EEPROM пам'ять запис відбувається побайтово, для читання та запису потрібно вказувати адресу першого байту та розмір змінної, туди зручно записувати визначені змінні, якими є внутрішні налаштування системи, як от час очікування чи дозвіл переходити в режим сну. Тоді на карту пам'яті припадають дані про отримані параметри та інформація про периферійні пристрої.

Для систематизованого збереження даних на SD карті було розроблено наступну систему правил:

- id пристрою, параметра та інструкції повинні мати строго 5-ти байтовий розмір.
- При отримані даних чи при виконані дії, до log файлу вносяться дані про час та тип події (g, get — отримано, s, send — відправлено) і сама подія, в назві вказано дату.
- При отриманні даних вони , окрім log файлу, також записуються в файл з розширенням .prm та назвою відповідною id отриманого параметра, додатково в цей файл вноситься час отримання. Третім необов'язковим рядком вказуються 2 (або 1 за вибором) параметри актуальності, а саме час в хвилинах (позначка t) та значення (позначка v), якщо пройде вказана кількість хвилин або параметр рівний вказаному значенню, то він не буде використовуватись системою, за винятком порівняння з іншими параметрами. Якщо файл відсутній, його буде створено, але без третього рядка.
- Інструкції, що центральний контролер надсилає периферійним пристроям, зберігаються в окремому файлі з розширенням .ins та назвою відповідною id параметра, до якого прив'язана інструкція. Файл містить оператор порівняння (=, !=, >, <, >=, <=), значення, код інструкції та id пристрою, якому вона відправляється. Якщо параметр порівнюється з іншим параметром, математичний оператор починається з латинської літери "p".
- Інформація про пристрої міститься у файлах з розширенням .dev та десятковим номером каналу, на якому ці пристрої працюють, в якості назви. Файл містить ідентифікатор пристрою та його адресу.

Слідуючи цим правилам система, що містить в своєму складі сенсор з id sens1 та актуатор з id lght1, які працюють на каналі 5a, буде мати наступні файли:

- log_12.06.2023.txt — дані про події
- Snsr1.prm — файл параметру Snsr1
- Snsr1.ins — файл інструкцій з параметром Snsr1
- 5a.dev — список пристроїв на каналі 5a

Наприклад, якщо сенсор спрацював на щось і передав параметр центральному контролеру, який опрацював його та відправив інструкцію до актуатора, це відобразиться в log файлі за відповідну дату:

```
21:13:37 g Snsr1 = 1
21:13:38 s on_ld to lght1
```

Та в файлі параметру:

```
1
20.06.2023 21:13:37
v0
```

Значення v0 означає, що якщо параметр рівний нулю, то не обробляти його (не демонструвати на екрані чи не відправляти в послідовний порт тощо).

В файлі Snsr1.ins буде вказано куди, яку і при яких значеннях відправити інструкцію, в прикладі це:

```
= 1 on_ld lght1
= 0 offld lght1
```

Що означає при значенні 1 відправити інструкцію on_ld пристрою з id lght1, а при значенні 0 інструкцію offld, тому ж пристрою.

І як, власне, зв’язатись з цими пристроями вказано наявністю файлу 5a.dev та його вмістом, в якому ідентифікатор пристрою пов’язаний з адресою:

```
sens1 node1
act_1 node2
```

Порядок роботи програми, без врахування взаємодії користувача, зображено на блок-схемі на рис. 3.7.

Для реалізації взаємодії з модулем читання та запису SD карт пам’яті з середовищем Arduino IDE постачається бібліотека SD.h. Взаємодія з файлами максимально наближена до подібних операцій на звичайні мові C++, файл так само необхідно спочатку відкрити, а після закінчення роботи закрити, доступно читання порядкове або всього файлу за раз. Єдина серйозна особливість — потрібно слідкувати за витратами оперативної пам’яті, оскільки, як і в операціях мовою C++, дані не можуть читатись напряму з файлу, натомість вони поміщуються до оперативної пам’яті. Для цього і були створені такі розгалужені правила зберігання

інформації, щоб мінімізувати розмір файлів, що читаються в момент. Єдиним великим файлом залишається log файл, проте він, на відміну від інших, не читається, а лише доповнюється.

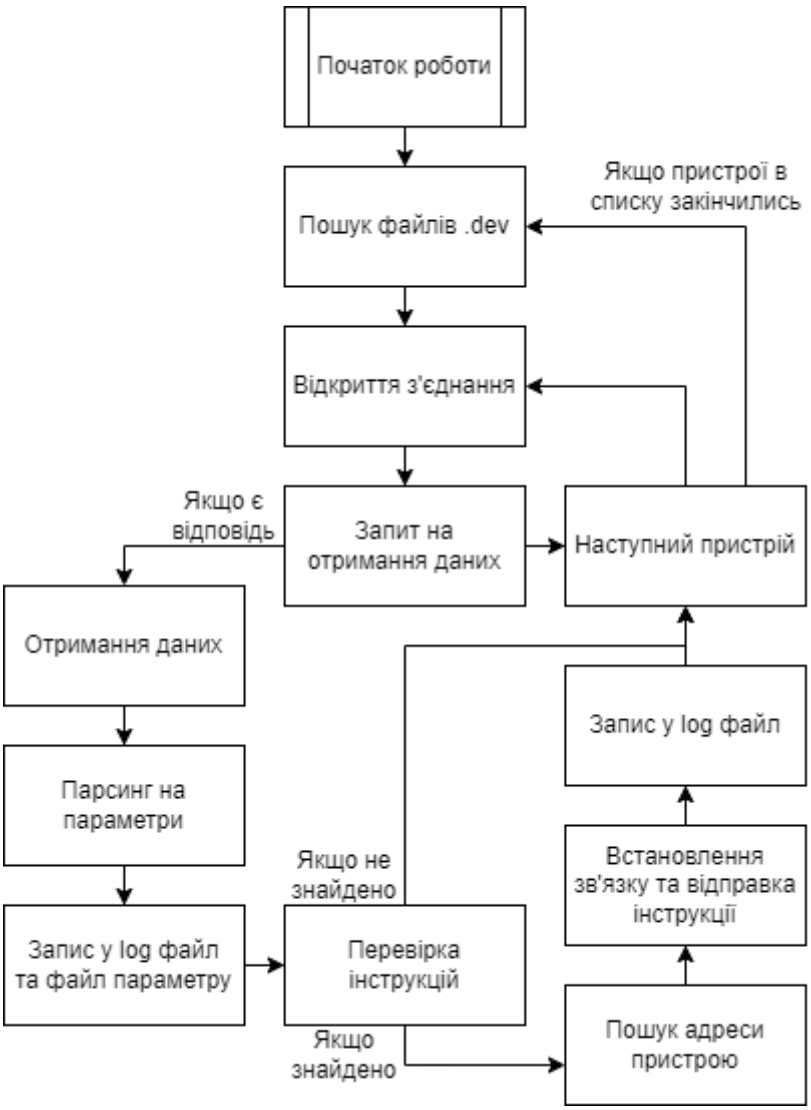

Рис. 3.7. Блок-схема алгоритму опитування пристроїв мережі

EEPROM пам’ять підходить для збереження внутрішніх налаштувань, оскільки змінні, які їх описують, зазвичай мають статичний розмір, що дозволяє компактно помістити велику їх кількість, не розмічаючи пам’ять на сектори з запасом по розміру. Цей тип пам’яті можна уявляти як масив байтів. Таким чином для запису змінної в пам’ять потрібно вказати номер байту в пам’яті та значення, яке потрібно записати, для читання тільки номер. Для запису змінних, що мають розміри більш ніж 1 байт існує функція put, що приймає параметрами номер першого байту та значення до запису, значення займе в пам’яті потрібну для нього

кількість байтів: integer 2 байти, float 4 байти і т. д. Для читання використовується функція get з номером першого байту та змінною, в яку потрібно записати отримане значення.

Система налаштувань.

Останнім важливим компонентом системи є продумана система налаштувань, до якої також можна віднести розв'язані питання по збереженню даних. З нерозв'язаних питань залишаються: перелік параметрів системи та їх налаштування.

Налаштування системи виражаються через ряд змінних, які є чітко визначеними зі сталим розміром, тому можна легко розбити енергонезалежну пам'ять EEPROM на розділи під кожну змінну та при завантаженні пристрою зчитувати з відти їх значення, а при зміні налаштувань оновлювати окремо потрібну секцію в пам'яті. Новий мікроконтролер містить значення 255 для кожного байту цієї пам'яті, тому перед завантаженням основної прошивки потрібно записати початкові налаштування.

Для повного налаштування приладу буде використовуватись послідовний порт пристрою та розроблений додаток для персональних комп'ютерів, що виступає в ролі графічної інтерпретації системи налаштувань. Він виконаний мовою Python, з використанням бібліотеки PyQt5, яка використовується для створення інтерфейсу та роботи з COM портом. Додатки виконані з використанням цієї бібліотеки можуть бути зібрані для використання на персональних комп'ютерах та мобільних пристроях. Детально використання PyQt5 розглядатись не буде, оскільки додаток не є незамінним для розроблюваної системи, і це напряму нестосується теми. Проте варто коротко відмітити, що розмітка додатку створюється по стандартному сценарію "створити віджет з класу віджета – задати характеристики – помістити на поле", а для взаємодії з COM портом використовуються методи модуля PyQt5.QtSerialPort по також алгоритму як і це відбувається в С для ардуїно, лише з поправкою на синтаксис мови.

Для підключення приладу, він має бути переведений через його меню у режим роботи з COM портом.

Додаток має 3 розділи (рис. 3.8): розділ для відкриття порту, розділ параметрів, розділ з інформацією з приладу.

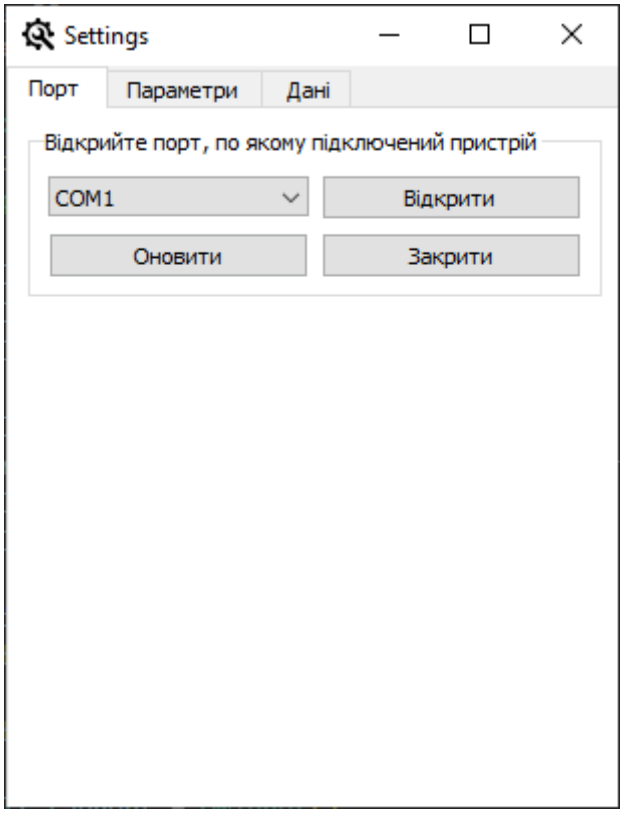

Рис. 3.8. Додаток для налаштування пристрою

Розділ “Порт” використовується для встановлення підключення до пристрою. Необхідно з випадаючого списку обрати порт на який підключено пристрій, оновити список портів можна відповідною кнопкою. На те чи з’єднання відкрите вказує напис зверху розділу. Підключення до мікроконтролера через послідовний порт повинно перезавантажувати пристрій, проте при використанні інструментів PyQt це відбувається не завжди, на роботу не впливає.

Розділ “Параметри” містить випадаючий список з варіантом, що налаштовувати, доступні загальні налаштування та керування об’єктами периферійних пристроїв, параметрів та інструкцій. Для отримання актуальних параметрів пристрою по відкритій вкладці призначена відповідна кнопка. Дані відправляються рядком та обробляються пристроєм, після чого отримавши

актуальні параметри з пристрою можна переглянути чи вірно вони були введені та застосовані.

Розділ “Дані” для перегляду log файлу за вказану дату або для перегляду даних про події в реальному часі по мірі їх появи. Щоб відкрити потрібний файл необхідно вказати дату його створення, рис. 23.

Інтерфейс приладу, що реалізований за допомогою екрану та енкодера в якості елемента керування, містить можливість налаштовувати частину параметрів, наприклад режим роботи з COM портом, відправка в COM порт актуальних збережених параметрів, дозвіл вимикати підсвітку дисплею та час для цього тощо.

Інші компоненти та функції.

З нерозглянутих компонентів залишились: модуль реального часу, дисплей та енкодер.

Для взаємодії з RTC модулем DS3231 використовується однойменна бібліотека. Принцип роботи з об’єктом класу аналогічний до вже розглянутих. Дані від цього модуля записуються в об’єкт типу Time, що містить поля sec, min, hour, date, mon, year, та dow (номер). Для встановлення та отримання окремих значень використовуються сетери і гетери. Функціонал модуля не обмежується підрахунком часу, проте в роботі інші функції не використовуються.

Щоб використання модуля не зводилось отримання часу для файлу історії подій, до системи даних внесено спеціальний параметр _TIME, що не містить файлу .prm та зберігається виключно в оперативній пам’яті, проте містить інструкції для часу у форматі hh:mm. Цей параметр, на відміну від інших, оновлюється всередині системи, при зміні часу.

Стосовно інтерфейсу, то для центрального контролера використовується рідкокристалічний символьний дисплей на 4 рядки по 20 символів з підключенням по i2c протоколу. Інтерфейс розроблений під ці розміри, тому дисплей може бути легко заміненим на інший за умови, що він не менших розмірів, має розмічену бібліотекою область та використовує протокол i2c.

Для цього дисплею використовується бібліотека LiquidCrystal_I2C. Для створення об’єкту класу вказується його адреса та розміри. Для друку використовуються методи clear, setCursor та print, може керувати підсвіткою методами backlight / noBacklight. Доступне створення 8-ми власних символів методом createChar з номером символу (0–7) та масивом, що описує символ (рис. 3.11), друк символу відбувається методом write. В розробленій модульній системі на екран виводяться значення з динамічного списку параметрів, що формується при отриманні даних, перший рядок містить дату, час та номер сторінки, інші 3 почергово розміщують ту частину списку параметрів, що відповідає номеру сторінки. Список внутрішніх налаштувань виводиться на екран при вході в меню та має строго визначений вигляд з прив’язкою відповідних параметрів до рядка меню.

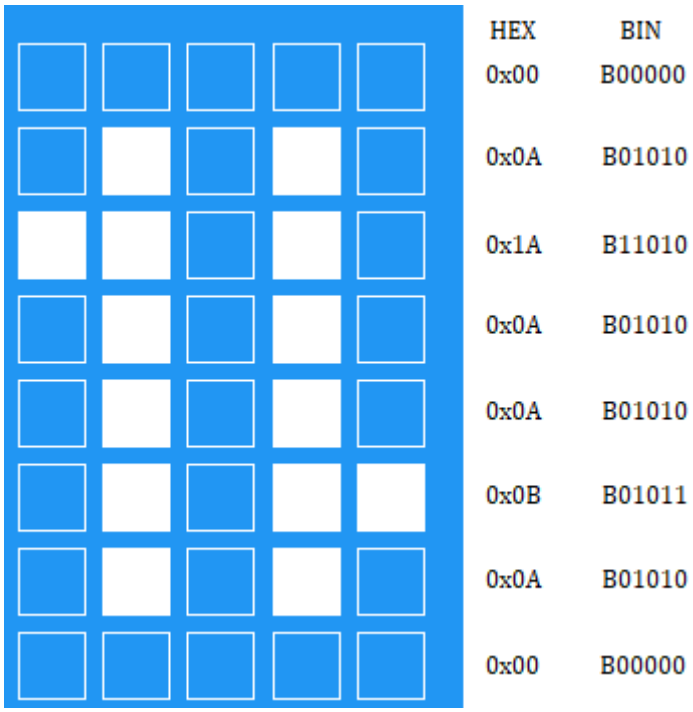

Рис. 3.9. Опис символу масивом

Принцип роботи енкодера достатньо простий: використовуються виходи A та B, при прокручуванні назад їх значення змінюються (1 або 0) та співпадають, при прокручуванні вперед теж змінюються, але відрізняються між собою. В розробленій модульній системі між запитами до периферійних пристроїв

перевіряється актуальне значення виходу A та порівнюється зі збереженим, і згідно з простою логікою робиться висновок чи крутився енкодер і в якому напрямку:

```
aState = digitalRead(outputA);
if (aState != aLastState) {
  if (digitalRead(outputB) != aState)
    counter ++;
  else
    counter --;
}
```

На цій основі виконано зміну різних параметрів інтерфейсу, як от номер сторінки чи зміна параметру меню. Енкодер також має третій вихід, що фіксує натиснення, це використовується для входу в меню та вибору потрібного його пункту. Варто враховувати можливий брязкіт контактів кнопки, в пристрої це вирішено за допомогою затримки обробки натиснення: сигнал рахується логічним натисненням тільки якщо триває не менше 100 мс.

З нерозглянутих функцій лишилась тільки функція енергозбереження через сон мікроконтролера. Центральний мікроконтролер функціонує неперервно, проте передбачена можливість модифікації його для використання сну, для цього до виходів зовнішніх переривань підключено вихід енкодера, що відповідає за кнопку та вихід IRQ радіомодулю, що може генерувати сигнал при отриманні даних. Але сенсори в системі не мають вимоги до постійної активності, тому можуть між вимірюваннями використовувати сон, що дозволить економити значну частину енергії.

Можна виділити кілька методів енергозбереження: вимкнення чи демонтаж елементів, що не використовуються на платах розробників, збір пристрою на чистому мікроконтролері, додаючи тільки необхідне, зниження частоти роботи мікроконтролера та режими сну. Хороший пристрій повинен мати оптимізовану плату, наскільки це можливо, тому щодо перших пунктів можна відзначити, що базова версія arduino не є енергоефективною, оскільки це насамперед плата розробника, що має великий спектр компонентів, тому після завершення розробки потрібно або модифікувати плату, або збирати власну, без зайвих деталей. А щодо частоти, то це може бути корисним для певних типів живлення, проте для сучасних

акумуляторів ефективніше на максимальній швидкості виконати необхідне і перейти до сну, тим не менше, зниження частоти може використовуватись у випадках, коли час активності прив'язаний до чогось, що не залежить від швидкодії пристрою, як наприклад аналогові датчики. Сон мікроконтролера є найбільш ефективним з поміж інших методів [6].

Для сну, контролю живлення елементів плат та частоти використовується бібліотека LowPower [14]. В розділі 2.2 було розглянуто режими сну мікроконтролерів, що можуть бути використані для побудови сенсорів мережі, зокрема в табл. 2.2. було порівняно підтримувані режими для кількох з них. Кількість режимів і на що вони впливають потрібно уточнювати в документації до мікроконтролера, проте зазвичай мікроконтролери містять режими легкий режим — IDLE, вимикає тільки cpu та flash, дозволяє миттєво продовжити роботу після пробудження, та режим глибокого сну — POWERDOWN, що вимикає все окрім зовнішніх переривань та вартового таймера (WDT), пробудження відбувається за 22 такти, що дорівнює ≈1.3 мкс. Сон може тривати певний проміжок часу, що складається з закладених з виробництва в мікроконтролер проміжків сну або не мати таймеру та пробуджуватись тільки по зовнішнім перериванням [7][14].

Таким чином робота сенсора має такий алгоритм: сон – пробудження по таймеру чи перериванню – проведення вимірювань – підготовка пакету до відправлення – очікування запиту від центрального контролера – відправка по запиту – очікування підтвердження успішного отримання – сон.

3.6. Огляд результату

Під поставленні завдання було створено центральний контролер та додаток для налаштування під ОС Windows 10 та вище.

Для використання центрального контролеру в складі модульної системи спостереження за параметрами середовища необхідно через додаток налаштування додати до системи периферійні пристрої, задати список параметрів, що будуть отримуватись від них та інструкції, що будуть надіслані центральним контролером.

Файли налаштувань також можна створити власноруч та помістити на карту пам’яті центрального контролера.

Після ввімкнення всіх модулів система почне працювати використовуючи радіочастоту 2.4 ГГц з захистом даних алгоритмом AES-128, що відповідає шифруванню аналогічних популярних пристроїв.

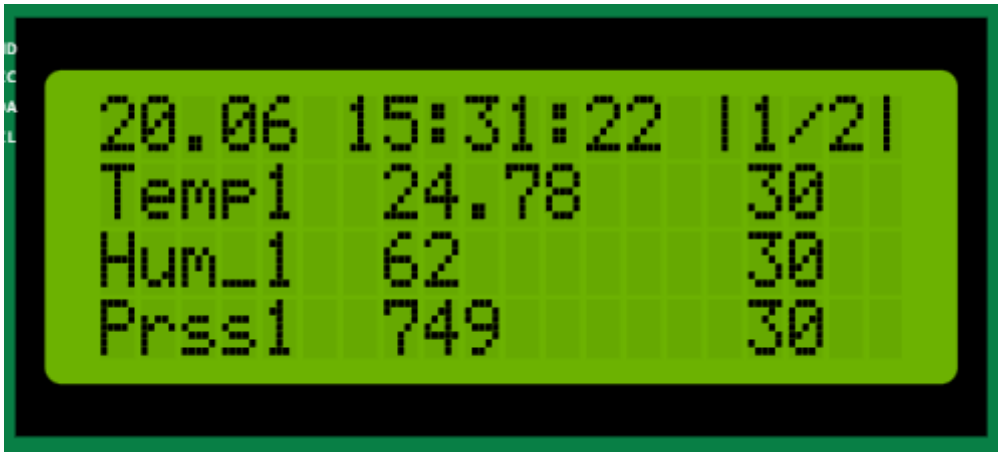

Рис. 3.10. Візуалізація головної сторінки центрального контролера

Параметри середовища, отримані від сенсорів, будуть зберігатись в файлах параметрів на карті пам’яті та друкуватись на екран (з часом скільки хв тому були отримані) і в COM порт, якщо ввімкнено відповідний режим в налаштуваннях. Залежно від отриманих даних та збережених інструкцій, пристрій буде відправляти ідентифікатор певної інструкції вказаному, при налаштуванні, приладу. Всі події будуть зберігатись в log файл. Log файли поділенні по даті, переглянути їх можна при перегляді карти пам’яті пристрою або з додатку вказавши потрібну дату.

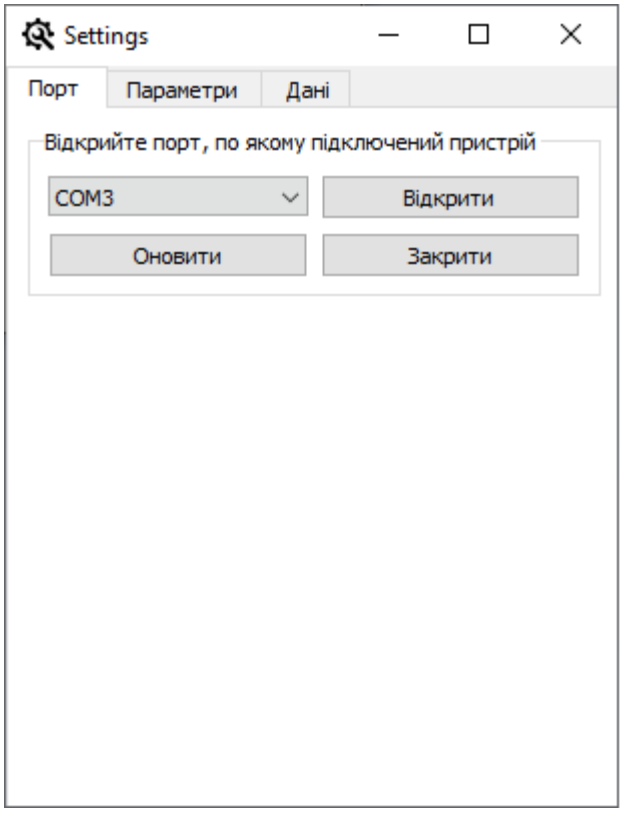

Рис. 3.11. Дані з log файлу за 20.06

Натиснувши на головному меню енкодер, можна відкрити меню пристрою, що містить можливість надіслати одну зі збережених інструкцій, надіслати актуальні параметри до послідовного порту та налаштувати час вимкнення дисплею, в хв, якщо значення 0, дисплей вимикатись не буде.

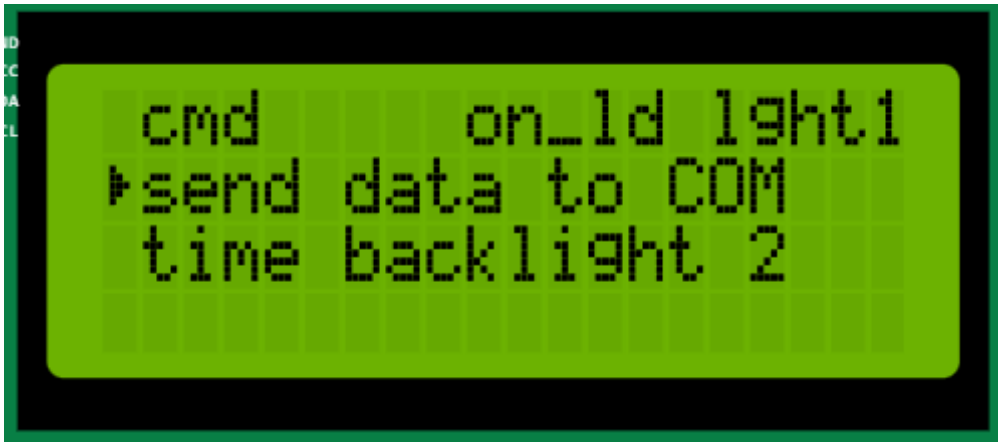

Рис. 3.12. Візуалізація сторінки меню

ВИСНОВКИ

В ході роботи було розглянуто основні поняття галузі розумних будинків та домашньої автоматизації, типові складові таких систем, історію їх розвитку, актуальні рішення та тенденцію очікуваного розвитку галузі. Було детально розглянуто існуючі мережеві топології та архітектури, виділено їх переваги та недоліки в порівнянні між собою. Досліджено популярні мікроконтролери та основні протоколи бездротової комунікації, описано їх переваги, недоліки, характеристики та ключові моменти, що повинні бути реалізовані в якісних проектах, незалежно від використаного в них стеку технологій.

Були набуті навички постановки вимог до систем домашньої автоматизації згідно з поставленими вимогами, навички роботи з мікроконтролерами в середовищі Arduino IDE та навички проектування схем електричних приладів.

В ході роботи було досягнуто поставлену мету та виконано поставлені завдання, а саме дослідження протоколів передачі даних, методів шифрування даних та енергозбереження, розробка центрального контролера для модульної системи домашньої автоматизації.

Практичне значення отриманого результату полягає у можливості значно пришвидшити та зробити доступнішим розгортання індивідуальними розробниками власних систем домашньої автоматизації на основі створеного центрального контролера мережі та шаблонів розробки периферійних пристроїв.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Gerhart James Home automation and wiring: McGraw-Hill, 1999 y., 340 p.

2. Anthony Velte Build Your Own Smart Home (Build Your Own): McGraw-Hill Osborne Media, 2003 y., 256 p.

3. James Kurose, Keith Ross Computer Networking: A Top-Down Approach: 2016 y, 864 p.

4. Dwight Spivey Home Automation For Dummies: Dummies Tech, 2015 y., 360 p.

5. Muhammad Ali Mazidi, Sarmad Naimi AVR Microcontroller and Embedded Systems: Using Assembly and C 1st Edition: Pearson India Education, 2012 y., 95 p.

6. Simon Monk Programming Arduino: Getting Started With Sketches: Tab Books, 2011 y., 162 p.

7. Elliot Williams AVR Programming: Learning to Write Software for Hardware: Make Community, LLC, 2014 y., 474 p.

8. Theodore S. Rappaport, Jeffrey G. Andrews Wireless Communications: Principles and Practice: Prentice Hall, 2002 y., 707 p.

9. Albert S. Huang Bluetooth Essentials for Programmers: Cambridge University Press, 2007 y., 210 p.

10. Drew Gislason Zigbee Wireless Networking: Newnes, 2008 y., 448 p.

11. Dr. Christian Paetz Z-Wave Basics: Remote Control in Smart Homes: CreateSpace Independent Publishing Platform, 2013 y., 300 p.

12. Nano | Arduino Documentation [Електронний ресурс] – режим доступу до ресурсу: https://docs.arduino.cc/hardware/nano

13. In-Depth: How nRF24L01 Wireless Module Works & Interface with Arduino [Електронний ресурс] – режим доступу до ресурсу: https://lastminuteengineers.com/nrf24l01-arduino-wireless-communication/

14. The Arduino Guide to Low Power Design [Електронний ресурс] – режим доступу до ресурсу: https://docs.arduino.cc/learn/electronics/low-power

ДОДАТКИ

Додаток А

Код основного файлу бібліотеки шифрування

```
#include <avr/pgmspace.h>
#include "AES128.h"

PROGMEM const byte AES128::Sbox[256] = {
   0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5,
   0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
   0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0,
   0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
   0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc,
   0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
   0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a,
   0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
   0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0,
   0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
   0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b,
   0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
   0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85,
   0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
   0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5,
   0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
   0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17,
   0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
   0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88,
   0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
   0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c,
   0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
   0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9,
   0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
   0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6,
   0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
   0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e,
   0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
   0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94,
   0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
   0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68,
   0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16
};

PROGMEM const byte AES128::inv_Sbox[256] = {
   0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38,
   0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb,
```

```
    0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87,
    0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb,
    0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d,
    0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e,
    0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2,
    0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25,
    0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16,
    0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92,
    0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda,
    0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84,
    0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a,
    0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06,
    0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02,
    0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b,
    0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea,
    0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73,
    0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85,
    0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e,
    0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89,
    0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b,
    0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20,
    0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4,
    0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31,
    0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f,
    0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d,
    0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef,
    0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0,
    0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61,
    0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26,
    0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d
};


PROGMEM const byte AES128::rcon [] = {
        0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab
};

AES128::AES128(const byte * skey) {
        memcpy((void*)codekey, (const void*)skey, 16);
}

void AES128::initKey() {
        memcpy((void*)key, (const void*)codekey, 16);
}
```

```
byte * AES128::encrypt(byte *message) {
	int i;

	memcpy((void*)state, (const void*)message,16);

	initKey();
	addRoundKey();

	for(i = 0; i < 9; i++) {
		subBytes();
		shiftRows();
		mixColumns();
		computeKey(pgm_read_byte(rcon + i));
		addRoundKey();
	}

	subBytes();
	shiftRows();
	computeKey(pgm_read_byte(rcon + i));
	addRoundKey();

	memcpy((void*)message,(const void*)state, 16);
	return message;
}

byte * AES128::decrypt(byte *message) {
	int i;

	memcpy((void*)state, (const void*)message, 16);

	initKey();
	inv_addRoundKey(10);
	inv_shiftRows();
	inv_subBytes();

	for(i = 0; i < 9; i++) {
		inv_addRoundKey(9-i);
		inv_mixColumns();
		inv_shiftRows();
		inv_subBytes();
	}

	inv_addRoundKey(0);
```

```
        memcpy((void*)message,(const void*)state,16);

        return message;
}


void AES128::addRoundKey() {
        int i;
        for(i=0; i < 16; i++) {
                state[i] ^= key[i];
        }
}


void AES128::inv_addRoundKey(int i) {
        computeKeyRound(i);
        addRoundKey();
}


void AES128::subBytes() {
        int i;

        for(i = 0; i < 16; i++) {
                state[i] = pgm_read_byte(Sbox + state[i]);
        }
}

void AES128::shiftRows() {
        byte temp;

        temp = state[1]; state[1] = state[5]; state[5] = state[9];
   state[9] = state[13]; state[13] = temp;
        temp = state[10]; state[10] = state[2]; state[2] = temp;
        temp = state[14]; state[14] = state[6]; state[6] = temp;
        temp = state[3]; state[3] = state[15]; state[15] = state[11];
   state[11] = state[7]; state[7] = temp;
}

void AES128::inv_subBytes() {
        int i;

        for(i = 0; i < 16; i++) {
                state[i] = pgm_read_byte(inv_Sbox + state[i]);
```

```
        }
}

void AES128::inv_shiftRows() {
        byte temp;
        temp = state[1]; state[1] = state[13]; state[13] = state[9];
        state[9] = state[5]; state[5] = temp;
        temp = state[10]; state[10] = state[2]; state[2] = temp;
        temp = state[14]; state[14] = state[6]; state[6] = temp;
        temp = state[3]; state[3] = state[7]; state[7] = state[11];
        state[11] = state[15]; state[15] = temp;
}

byte AES128::xtime(byte x) {
        return (x & 0x80) ? ((x << 1) ^ 0x1b) : (x<<1);
}

void AES128::mixColumns() {
        byte i, a, b, c, d, e;
        for(i = 0; i < 16; i+=4) {
                a = state[i]; b = state[i+1]; c = state[i+2]; d = state[i+3];
                e = a ^ b ^ c ^ d;
                state[i]   ^= e ^ xtime(a^b);
                state[i+1] ^= e ^ xtime(b^c);
                state[i+2] ^= e ^ xtime(c^d);
                state[i+3] ^= e ^ xtime(d^a);
        }
}

void AES128::inv_mixColumns() {
        byte i, a, b, c, d, e, x, y, z;
        for(i = 0; i < 16; i+=4)
        {
                a = state[i]; b = state[i+1]; c = state[i+2]; d = state[i+3];
                e = a ^ b ^ c ^ d;
                //Inverses
                z = xtime(e);
                x = e ^ xtime(xtime(z^a^c) );
                y = e ^ xtime(xtime(z^b^d) );
                state[i]   ^= x ^ xtime(a^b);
                state[i+1] ^= y ^ xtime(b^c);
                state[i+2] ^= x ^ xtime(c^d);
                state[i+3] ^= y ^ xtime(d^a);
        }
```

```
}

void AES128::computeKeyRound(int round) {
        int i;
        initKey();
        for (i=0; i < round; i++ ) {
                computeKey(pgm_read_byte(rcon+i));
        }
}

void AES128::computeKey(byte r) {
        byte buf0, buf1, buf2, buf3;
        buf0 = pgm_read_byte(Sbox + key[13]);
        buf1 = pgm_read_byte(Sbox + key[14]);
        buf2 = pgm_read_byte(Sbox + key[15]);
        buf3 = pgm_read_byte(Sbox + key[12]);

        key[0] ^= buf0 ^ r;
   key[1] ^= buf1;
   key[2] ^= buf2;
   key[3] ^= buf3;

        key[4] ^= key[0];
        key[5] ^= key[1];
        key[6] ^= key[2];
        key[7] ^= key[3];

        key[8]  ^= key[4];
        key[9]  ^= key[5];
        key[10] ^= key[6];
        key[11] ^= key[7];

        key[12] ^= key[8];
        key[13] ^= key[9];
        key[14] ^= key[10];
        key[15] ^= key[11];
}
```