Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

А. В. Нікітчук

ПРОГРАМУВАННЯ ВБУДОВАНИХ СИСТЕМ ІНТЕРНЕТУ РЕЧЕЙ

Лабораторний практикум

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакалавра
за освітньою програмою «Інтелектуальні технології радіоелектронної техніки»
спеціальності 172 Електронні комунікації та радіотехніка

Електронне мережеве навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2024

УДК 004.4’23.031.6:004.77(07)
Н 62

Автор *Нікітчук Артем Валерійович*

Рецензент *Мирончук О. Ю.,* PhD, доцент кафедри РТС РТФ

Відповідальний редактор *Шульга А. В.*, к. т. н., доцент

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 1 від 26.09.2024 р.)
за поданням вченої ради радіотехнічного факультету
(протокол № 6/2024 від 28.06.2024 р.)

Нікітчук А. В.

Н 62 Програмування вбудованих систем Інтернету речей [Електронний ресурс] : лабораторний практикум : навч. посіб. для здобувачів ступеня бакалавра за освіт. програмою «Інтелектуальні технології радіоелектронної техніки» спец. 172 Електронні комунікації та радіотехніка / А. В. Нікітчук ; КПІ ім. Ігоря Сікорського. – Електрон. текст. дані (1 файл). – Київ : КПІ ім. Ігоря Сікорського, 2024. – 88 с.

Лабораторний практикум присвячений програмуванню вбудованих систем, використанню комунікаційних протоколів, підключенню вбудованих систем до мережі Інтернет, обміну даними та командами із хмарною платформою. Останні роботи присвячені паралельному виконанню декількох задач з використанням операційної системи реального часу. Лаконічний виклад теоретичних відомостей та покрокового порядку виконання для кожної роботи дозволяє швидко засвоїти розглянуті теми та застосувати їх на практиці. Видання призначене для студентів технічних спеціальностей, а також інженерів зацікавлених у розширенні знань в сфері вбудованих систем.

УДК 004.4’23.031.6:004.77(07)

Реєстр. № НП 24/25-049. Обсяг 7 авт. арк.
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Берестейський, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

ЗМІСТ

ВСТУП

Вбудовані системи є основою сучасних технологій, забезпечуючи роботу широкого спектра пристроїв – від побутової техніки до промислових автоматизованих систем. Цей посібник створений для студентів, які прагнуть здобути практичні навички програмування мікроконтролерів для роботи з комунікаційними протоколами та периферійними пристроями, а також використання операційних систем та інших корисних засобів.

Навчальний посібник містить 8 лабораторних робіт, тематика яких включає:

- Знайомство з інструментами для симуляції та відлагодження вбудованих систем.
- Вивчення методів управління послідовністю виконання інструкцій, використання операторів, циклів та підпрограм.
- Використання протоколів для комунікації всередині та між вбудованими системами.
- Підключення до Інтернету та налаштування мережевої комунікації.
- Інтеграцію з хмарними платформами, для збору, обробки та візуалізації даних.
- Знайомство з операційною системою реального часу та вивчення методів синхронізації і управління ресурсами у багатозадачних сценаріях використання.

Використання високорівневої мови програмування та доступу до численних бібліотек, дозволить зосередитися на функціональності та логіці програм, не витрачаючи час на розробку низькорівневого коду, що сприятиме швидшому засвоєнню нових концепцій та ефективній реалізації завдань. Також, це надасть можливість легко переносити написаний код між платформами, забезпечуючи високу гнучкість у виборі апаратного забезпечення.

Виконання завдань практикуму надасть студентам важливий досвід, що є необхідним для ефективної розробки сучасних вбудованих систем та пристроїв Інтернету речей.

ЛАБОРАТОРНА РОБОТА № 1

Тема: Онлайн-симуляція в процесі програмування вбудованих систем

Мета: Навчитися використовувати інструменти онлайн-симуляції та документацію до мікроконтролерів для розробки та відлагодження програмного забезпечення.

План проведення лабораторної роботи

- Ознайомлення з теоретичними відомостями.
- Виконання завдання згідно порядку виконання роботи.
- Подання звіту в Moodle (кнопка «Здати роботу»).

Теоретичні відомості

Робоче середовище

Онлайн-інженерія – це концепція, яка охоплює використання інтернет-технологій для виконання різних інженерних завдань, які раніше вимагали фізичної присутності або спеціального обладнання на місці. Вона охоплює широке коло споріднених технологій, наприклад:

- Віддалений доступ до онлайн ресурсів, програмного забезпечення, віртуальних лабораторій і середовищ.
- Спільна робота в реальному часі, дистанційне навчання та тренінги.
- Онлайн симуляції та моделювання.
- Хмарні сервіси.

Застосування онлайн-інженерії значно розширює можливості інженерів, роблячи роботу більш гнучкою, доступною і ефективною.

Симулятори – це програми, які дозволяють імітувати роботу вбудованої системи на комп’ютері. Тобто, виконувати, відлагоджувати, тестувати та оптимізувати програмне забезпечення вбудованої системи без необхідності фізичного доступу до апаратного забезпечення. *Приклади*: Wokwi, Proteus, QEMU, Simulink та інші.

Wokwi [1] – це веб-платформа, онлайн-симулятор електроніки, що дозволяє імітувати та тестувати різні проекти вбудованих систем та Інтернету речей. Wokwi моделює широкий спектр апаратних компонентів, включаючи мікроконтролери, датчики, дисплеї тощо. Заявлена підтримка архітектур: ARM, AVR, RISC-V і Xtensa. Симулятор перетворює вихідний код (ASM, C/C++, MicroPython, Rust) у бінарну мікропрограму, а потім виконує двійкову мікропрограму по одній інструкції за раз, як це робить справжній мікроконтролер.

В документації Wokwi [2] зазначено про наступні переваги даного симулятора:

- Не потрібно замовляти компоненти або завантажувати об’ємне програмне забезпечення, у браузері є все.
- Необмежена кількість електронних компонентів, які неможливо випадково спалити.

- За допомогою посилання на проект можливо легко ним поділитись з іншими розробниками.
- Платформа безкоштовна, а на офіційному сайті спільнота творців постійно розміщує власні цікаві проекти.
- При роботі з реальними пристроями, Wokwi можна застосовувати для перевірки коду і відділення проблем з програмою від апаратних.

Wokwi, як і Arduino IDE, може працювати з файлами .ino. Зазвичай такий файл містить код на мові C++ з використанням функцій Arduino. Переглянути основні функції можна в документації за посиланням [3].

Порти та виводи ATmega328P

Мікроконтролер ATmega328P (рис. 1.1) спілкується із зовнішнім світом через кілька портів вводу/виводу (I/O).

Порти – це групи бітів (ліній/пінів), які можна індивідуально налаштовувати для виведення або зчитування даних.

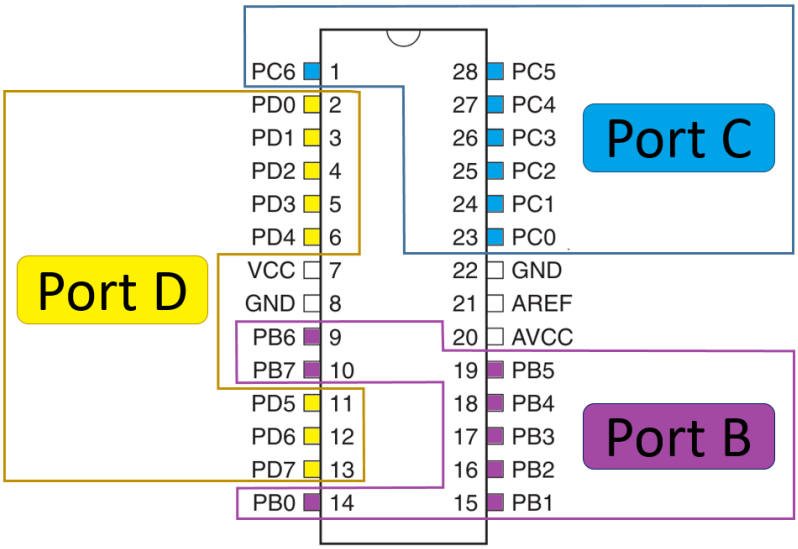


Рис. 1.1. Порти та піни ATmega328P у корпусі DIP-28

Всього таких портів три: PORTB, PORTC та PORTD. Визначити до якого порту відноситься конкретний вивід мікроконтролера можливо з його позначення (рис. 1.1), наприклад:

- вивід 3 має позначення PD1 – це перший пін порту D;
- вивід 27 позначено PC4 – це четвертий пін порту C;
- вивід 10 позначено PB7 це сьомий пін порту B.

Зазвичай плати розробника мають пласні виводи, нумерація яких не співпадає з номерами виводів мікроконтролера. На платі Arduino Uno, яка використовує мікроконтролер ATmega328P:

- **PORTB** має 8 пінів, до PB6 та PB7 підключається кварцовий резонатор, а PB0-PB5 відповідають піни плати 8-13.
- **PORTC** має 7 пінів, PC0-PC6, яким відповідають піни плати A0-A5 та RESET.
- **PORTD** має 8 пінів, PD0-PD7, яким відповідають піни плати 0-7.

Регістри керування портами ATmega328P

Керування портами – це налаштування режиму роботи пінів (вхід або вихід), а також встановлення або зчитування їх значень. Для керування портами ATmega328P використовуються спеціальні регістри:

- **DDR*B*, DDR*C*, DDR*D*** (Data Direction Register, регістри напрямку даних) – використовується для налаштування напрямку (вхід/вихід) портів B, C, D.
- **PORT*B*, PORT*C*, PORT*D*** – використовуються для встановлення значень на пінах портів B, C, D.
- **PIN*B*, PIN*C*, PIN*D*** – використовуються для зчитування значень з пінів портів B, C, D.

Зазначені регістри мають по 8 розрядів (бітів). Кожному біту відповідає певний пін мікроконтролера. Тобто, змінюючи значення певного біту відбувається налаштування певного піна.

Значення бітів регістрів **DDR**:

- «0» – пін налаштований як вхідний;
- «1» – пін налаштований як вихідний.

Значення бітів регістрів **PORT**:

- Якщо пін налаштований як вихідний:
  - «0» – встановлює його в низький стан (LOW);
  - «1» – встановлює його в високий стан (HIGH).
- Якщо пін налаштований як вхідний:
  - «0» – внутрішній підтягувальний резистор відключений.
  - «1» – внутрішній підтягувальний резистор включений.

Значення бітів регістрів **PIN**:

- «0» – логічний стан на піні низький (LOW).
- «1» – логічний стан на піні високий (HIGH).

Наприклад, до пінів 8 та 9, плати Arduino Uno, підключено аноди двох світлодіодів і їх необхідно одночасно засвітити. Встановити високий рівень на пінах 8 та 9 можна дотримуючись наступних кроків:

1. **Налаштування 8-го та 9-го пінів як вихідних.** Цим пінам Arduino Uno відповідають піни ATmega328P – 14 (PB0) та 15 (PB1), які відносяться до порта B, тому, для їх налаштування, необхідно використовувати регістр DDRB. Дізнатись які біти регістра DDRB відповідають за налаштування PB0 та PB1 можна з таблиці 1.1:

Табл. 1.1. Відповідність бітів регістру DDRB

Біти DDRB	DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0
Піни порта B	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
Піни Arduino Uno	-	-	13	12	11	10	9	8

Запис «1» у два наймолодші біти, DDRB0 та DDRB1, налаштує піни PB0 та PB1 як виводи:

```
DDRB = 0b00000011;
```

Префікс **0b** вказує на те, що число записане в двійковому (бінарному) форматі. Тобто після префіксу йде 8 біт, від старшого біту до молодшого.

2. **Встановлення високого рівня на PB0 та PB1.** За встановлення рівня на пінах порта B відповідає регістр PORTB. Відповідність бітів цього регістра наведено у таблиці 1.2:

Табл.1.2. Відповідність бітів регістру PORTB

Біти PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
Піни порта B	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
Піни Arduino Uno	-	-	13	12	11	10	9	8

За піни PB0 та PB1 відповідають біти PORTB0 та PORTB1, тому, в відповідні біти необхідно записати «1»:

```
PORTB = 0b00000011;
```

Після цього, два світлодіоди мають одночасно засвітитися.

Перевірити логічні рівні порта B можливо за допомогою регістра PINB:

```
if (PINB == 0b00101111)
{
    //щось зробити
}
```

Цей код порівнює всі біти в регістрі PINB з шаблоном 0b00101111. Це означає, що умова виконається тільки в тому випадку, якщо всі біти в PINB точно відповідають цьому шаблону, тобто піни PB0-PB3 та PB5 мають високі рівні (HIGH), а піни PB4, PB6, PB7 – низькі (LOW).

Завдання

1. Відвідайте веб-сайт Wokwi та зареєструйтесь. Ознайомтесь основними функціями та можливостями онлайн-середовища для розробки і симуляції вбудованих систем.
2. Створіть проект з платою Arduino Uno та підключіть до неї світлодіоди. За допомогою функцій Arduino напишіть код для блимання ними згідно варіанту.
3. Ознайомтесь з документацією до мікроконтролера ATmega328P та реалізуйте аналогічну програму (для блимання світлодіодами) без використання функцій Arduino.
4. Створіть аналогічний проект для ESP32, STM32 або Raspberry Pi Pico. Спробуйте запустити на даних платформах код написаний під час виконання завдання 2 та 3. Зробіть висновки про переносимість коду.

Таблиця варіантів

Номер варіанту можливо дізнатись в відповідному розділі дистанційного курсу за посиланням [4]. В комірці таблиці 1.3, сума заговків якої відповідає номеру варіанту, знаходиться потрібна комбінація літер. Вони відповідають першим літерам кольорів світлодіодів, якими необхідно блимати: «ч» – червоний; «п» – помаранчевий; «ж» – жовтий; «з» – зелений; «с» – синій; «р» – рожевий. Наприклад, для варіанту 14 – «пср», тобто необхідно одночасно блимати лише помаранчевим, синім та рожевим світлодіодами.

Табл. 1.3. Варіанти завдання

Вар.	0	1	2	3	4	5	6	7	8	9
0	–	чпжзср	чзср	жзр	чжс	чпж	ср	пжзср	чжср	жзс
10	пзр	зр	чжзср	чжзр	пср	чжз	жр	жс	чпзср	чжзс
20	чпзр	пжр	зс	чпжср	чпср	пзс	чжз	жз	пр	чпжзр
30	чпжр	чпжр	пжз	пс	чпжзс	чпзс	пжс	жзср	чпр	пз
40	пж	пжср	чпжз	чзр	чр	пзср	чпжс	чср	чпс	чс
50	чз	чж	пжзр	зср	чзс	чп	пжзс	жср	чжр	чпз

Порядок виконання роботи

Крок 1. Реєстрація та створення проекту:

1. Перейдіть за посиланням (Симулятор Wokwi : сайт URL: https://wokwi.com/) та зареєструйте власний аккаунт (рис. 1.2).

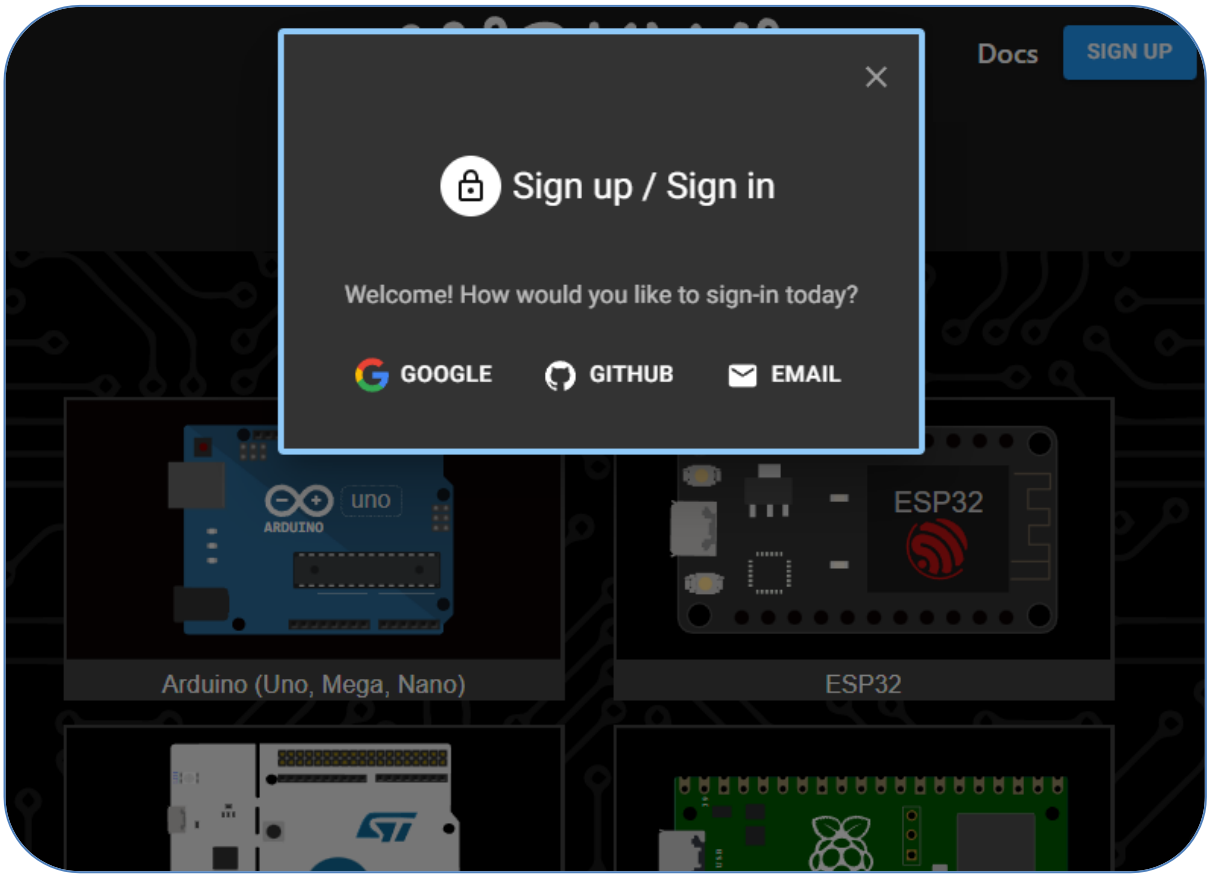


Рис. 1.2. Меню реєстрації на сайті Wokwi

2. Після входу на сайт, відкрийте меню (в лівому верхньому куті) та оберіть «My Projects» (рис. 1.3).

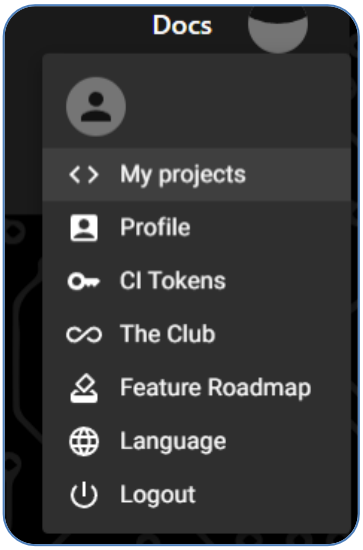


Рис. 1.3. Головне меню Wokwi

3. Натисніть кнопку «NEW PROJECT» (рис. 1.4).

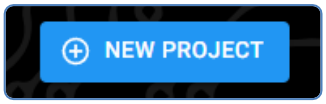


Рис. 1.4. Зовнішній вигляд кнопки для створення нового проекту

4. Оберіть плату розробника «Arduino Uno Rev3» (рис. 1.5).

Рис. 1.5. Плата розробника Arduino Uno Rev3

Крок 2. Ознайомлення з документацією Wokwi:

1. Підтримуване апаратне забезпечення: https://docs.wokwi.com/getting-started/supported-hardware
2. Інтерактивний редактор діаграм (додавання, редагування та видалення компонентів та з'єднань) - https://docs.wokwi.com/guides/diagram-editor
3. Комбінації клавіш у редакторі коду: https://docs.wokwi.com/keyboard-shortcuts
4. Підключення сторонніх бібліотек: https://docs.wokwi.com/guides/libraries

Додатково можна ознайомитись з популярними проектами, що збережені на сайті іншими користувачами:

- Arduino: https://wokwi.com/arduino
- ESP32: https://wokwi.com/esp32
- STM32: https://wokwi.com/stm32
- Raspberry Pi Pico: https://wokwi.com/pi-pico

Крок 3. Підключення світлодіода:

1. У вікні «Simulation» знайдіть та натисніть кнопку позначену «+» (рис. 1.6).

Рис. 1.6. Кнопки керування симуляцією

2. Оберіть звичайний світлодіод «LED» (рис. 1.7).

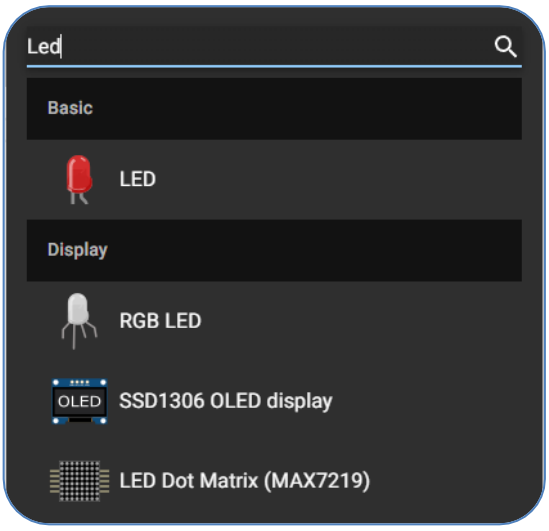


Рис. 1.7. Меню вибору компонентів

Після цього світлодіод буде додано в робочу область. Повернути елемент або змінити його колір можливо виділивши його мишкою.

3. Зберіть схему відповідно до рисунку 1.8 (в симуляторі можна знехтувати обмежувальними резисторами).

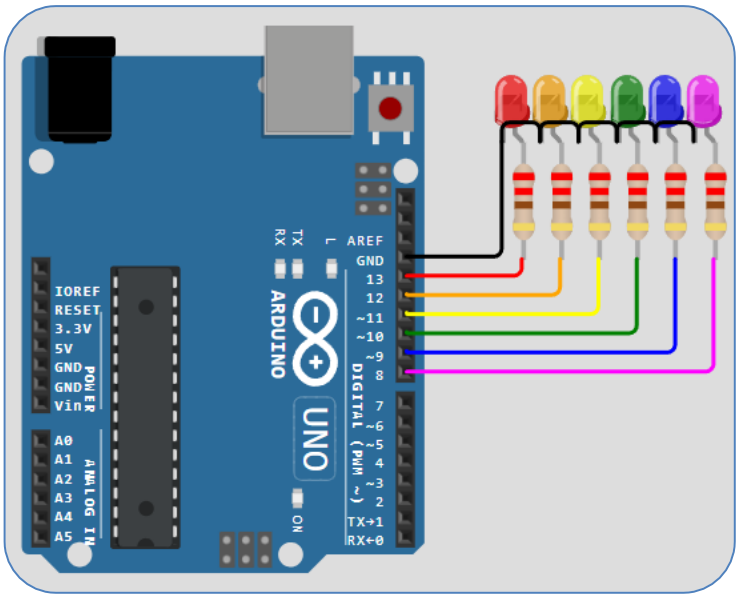


Рис. 1.8. Вигляд зібраної схеми

Щоб не збирати кожного разу схему «вручну», в проекті Wokwi існує файл **`diagram.json`**, в якому описуються всі з'єднання між електронними компонентами, їх розташування та інші

властивості. Для отримання готової схеми (рис. 1.8) замініть початковий зміст файлу **diagram.json** наступним текстом:

```
{"version": 1, "author": "", "editor": "wokwi", "parts": [{"type": "wokwi-arduino-uno", "id": "uno", "top": 25.8, "left": -190.6, "rotate": 90, "attrs": {}}, {"type": "wokwi-led", "id": "led1", "top": 6, "left": 71, "attrs": {"color": "red"}}, {"type": "wokwi-led", "id": "led2", "top": 6, "left": 90.2, "attrs": {"color": "orange"}}, {"type": "wokwi-led", "id": "led3", "top": 6, "left": 109.4, "attrs": {"color": "yellow"}}, {"type": "wokwi-led", "id": "led4", "top": 6, "left": 128.6, "attrs": {"color": "green"}}, {"type": "wokwi-led", "id": "led5", "top": 6, "left": 147.8, "attrs": {"color": "blue"}}, {"type": "wokwi-led", "id": "led6", "top": 6, "left": 167, "attrs": {"color": "magenta"}}, {"type": "wokwi-resistor", "id": "r1", "top": 72, "left": 66.65, "rotate": 90, "attrs": {"value": "220"}}, {"type": "wokwi-resistor", "id": "r2", "top": 72, "left": 85.85, "rotate": 90, "attrs": {"value": "220"}}, {"type": "wokwi-resistor", "id": "r3", "top": 72, "left": 105.05, "rotate": 90, "attrs": {"value": "220"}}, {"type": "wokwi-resistor", "id": "r4", "top": 72, "left": 124.25, "rotate": 90, "attrs": {"value": "220"}}, {"type": "wokwi-resistor", "id": "r5", "top": 72, "left": 143.45, "rotate": 90, "attrs": {"value": "220"}}, {"type": "wokwi-resistor", "id": "r6", "top": 72, "left": 162.65, "rotate": 90, "attrs": {"value": "220"}}], "connections": [["led1:C", "uno:GND.1", "black", ["v57.6", "h-47.6"]], ["led6:C", "led5:C", "black", ["v-9.6", "h-18.8"]], ["led5:C", "led4:C", "black", ["v-9.6", "h-18.8"]], ["led4:C", "led3:C", "black", ["v-9.6", "h-18.8"]], ["led3:C", "led2:C", "black", ["v-9.6", "h-18.8"]], ["led2:C", "led1:C", "black", ["v-9.6", "h-18.8"]], ["led1:A", "r1:1", "red", ["v0"]], ["r1:2", "uno:13", "red", ["v0", "v8.4", "h-57.6"]], ["led2:A", "r2:1", "orange", ["v0"]], ["r2:2", "uno:12", "orange", ["v18", "h-76.8"]], ["led3:A", "r3:1", "yellow", ["v0"]], ["r3:2", "uno:11", "yellow", ["v27.6", "h-96"]], ["led4:A", "r4:1", "green", ["v0"]], ["r4:2", "uno:10", "green", ["v37.2", "h-115.2"]], ["led5:A", "r5:1", "blue", ["v0"]], ["r5:2", "uno:9", "blue", ["v46.8", "h-134.4"]], ["led6:A", "r6:1", "magenta", ["v0"]], ["r6:2", "uno:8", "magenta", ["v56.4", "h-153.6"]]], "dependencies": {}}
```

Крок 4. Написання скетчу, для блимання світлодіодами, за допомогою функцій Arduino:

1. Вставте наступний код у вікно редактора:

```
void setup()
{
    // Налаштування піну 12 як вихідного
    pinMode(12, OUTPUT);
}

void loop()
{
    // Вмикаємо світлодіод. Напруга на виводі відповідає логічному рівню “1”.
    digitalWrite(12, HIGH);
    delay(1000); // Затримка 1 секунда

    //Вимикаємо світлодіод. Напруга на виводі відповідає логічному рівню “0”.
    digitalWrite(12, LOW);
    delay(1000); // Затримка 1 секунда
}
```

2. Розпочніть симуляцію за допомогою зеленої кнопки зі стрілочкою (рис. 1.5).
3. Перевірте роботу програми (блимання світлодіоду).
4. Відредагуйте код, для реалізації завдання №2.
5. Збережіть проект з виконаним завданням №2.

Крок 5. Написання коду без використання функцій Arduino:

1. Створіть новий проект (копію попереднього).
2. Arduino Uno оснащена мікроконтролером ATmega328P. Дослідіть теоретичні відомості та документацію. Визначте з якими регістрами необхідно працювати для встановлення режимів роботи пінів і «блимання» потрібними світлодіодами (Datasheet доступний за посиланням: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf).
3. Замініть функції **`pinMode()`** та **`digitalWrite()`** на безпосередню роботу регістрами керування портами.
4. Встановіть затримку, щоб світлодіоди блимали, без використання функції delay від Arduino.
5. Застосуйте класичний підхід, використавши функцію **`int main()`** замість функцій **`void setup()`** та **`void loop()`**.
6. Виконайте завдання №3.
7. Збережіть проект з виконаним завданням №3.

Крок 6. Ознайомлення корисними функціями Wokwi:

1. Завантаження файлу з машинним кодом (для прошивки мікроконтролера). Для цього:
   - активуйте вікно редактора коду та натисніть F1;
   - у меню команд (рис. 1.9) оберіть «Download Compiled Firmware».

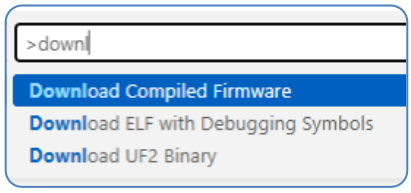


Рис. 1.9. Меню команд Wokwi

2. Перегляд асемблерного коду програми:
   - активуйте вікно редактора коду та натисніть F1;
   - у меню команд оберіть «View Compiled Assembly Code Listing» (рис. 1.10).

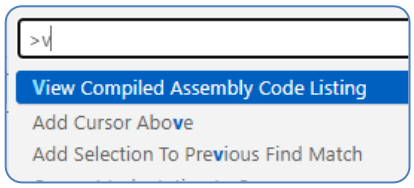


Рис. 1.10. Меню команд Wokwi

Крок 7. Перенесення коду на іншу платформу:

1. Створіть проект для будь-якої іншої платформи (рис. 1.11).

Рис. 1.11. Інші підтримувані платформи симулятором Wokwi

2. Використайте готову схему з’єднання компонентів для обраної платформи замінивши зміст файлу **`diagram.json`**:
   - Для ESP32 (рис. 1.12):

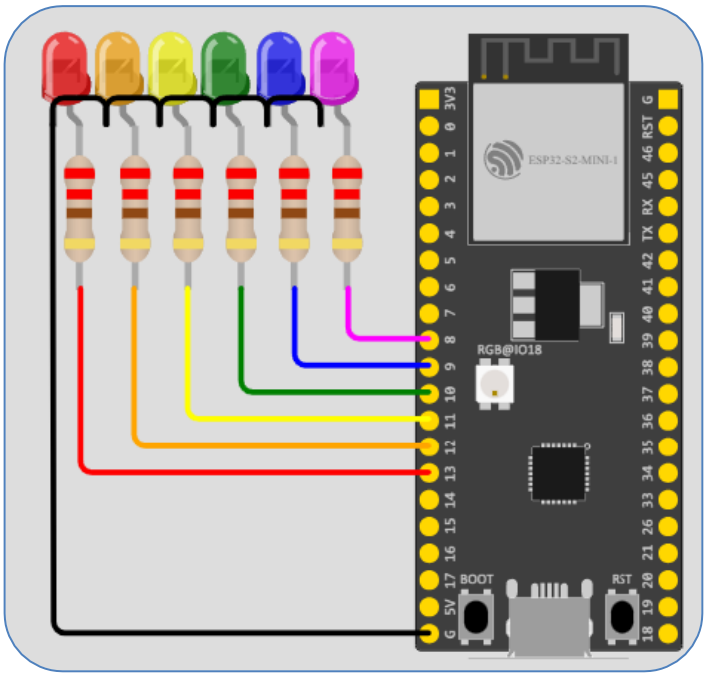


Рис. 1.12. Схема з ESP32

```
{"version": 1, "author": "", "editor": "wokwi", "parts": [{"type": "board-esp32-s2-devkitm-1", "id": "esp", "top": -33.11, "left": 196.57, "attrs": {}}, {"type": "wokwi-led", "id": "led1", "top": -42, "left": 51.8, "attrs": {"color": "red"}}, {"type": "wokwi-led", "id": "led2", "top": -42, "left": 71, "attrs": {"color": "orange"}}, {"type": "wokwi-led", "id": "led3", "top": -42, "left": 90.2, "attrs": {"color": "yellow"}}, {"type": "wokwi-led", "id": "led4", "top": -42, "left": 109.4, "attrs": {"color": "green"}}, {"type": "wokwi-led", "id": "led5", "top": -42, "left": 128.6, "attrs": {"color": "blue"}}, {"type": "wokwi-led", "id": "led6", "top": -42, "left": 147.8, "attrs": {"color": "magenta"}}, {"type": "wokwi-resistor", "id": "r1", "top": 24, "left": 47.45, "rotate": 90, "attrs": {"value": "220"}}, {"type": "wokwi-resistor", "id": "r2", "top": 24, "left": 66.65, "rotate": 90, "attrs": {"value": "220"}}, {"type": "wokwi-resistor", "id": "r3", "top": 24, "left": 85.85, "rotate": 90, "attrs": {"value": "220"}}, {"type": "wokwi-resistor", "id": "r4", "top": 24, "left": 105.05, "rotate": 90, "attrs": {"value": "220"}}, {"type": "wokwi-resistor", "id": "r5", "top": 24, "left": 124.25, "rotate": 90, "attrs": {"value": "220"}}, {"type": "wokwi-resistor", "id": "r6", "top": 24, "left": 143.45, "rotate": 90, "attrs": {"value": "220"}}], "connections": [["esp:TX", "$serialMonitor:RX", "", []], ["esp:RX", "$serialMonitor:TX", "", []], ["led6:A", "uno:8", "green", ["v38.4", "h-19.6"]], ["led3:A", "uno:11", "green", ["v19.2", "h28.2"]], ["uno:12", "led2:A", "green", ["v-28.8", "h-18.7", "v-28.8"]], ["led1:A", "uno:13", "green", ["v38.4", "h28.8", "v19.2"]], ["led1:C", "uno:GND.1", "black", ["v48", "h48.1"]], ["led6:C", "led5:C", "black", ["v-9.6", "h-18.8"]], ["led5:C", "led4:C", "black", ["v-9.6", "h-18.8"]], ["led4:C", "led3:C", "black", ["v-9.6", "h-18.8"]], ["led3:C", "led2:C", "black", ["v-9.6", "h-18.8"]], ["led2:C", "led1:C", "black", ["v-9.6", "h-18.8"]], ["led5:A", "uno:9", "green", ["v28.8", "h-9.6", "v28.8"]], ["led4:A", "uno:10", "green", ["v0"]], ["led1:C", "esp:GND.1", "black", ["h0.4", "v182.4"]], ["esp:8", "r6:2", "magenta", ["h0"]], ["r6:1", "led6:A", "magenta", ["h0"]], ["esp:9", "r5:2", "blue", ["h0"]], ["r5:1", "led5:A", "blue", ["h0"]], ["esp:10", "r4:2", "green", ["h0"]], ["r4:1", "led4:A", "green", ["h0"]], ["esp:11", "r3:2", "yellow", ["h0"]], ["r3:1", "led3:A", "yellow", ["h0"]], ["esp:12", "r2:2", "orange", ["h0"]], ["r2:1", "led2:A", "orange", ["h0"]], ["esp:13", "r1:2", "red", ["h0"]], ["r1:1", "led1:A", "red", ["h0"]]], "dependencies": {}}
```

- Для STM32 (рис. 1.13):

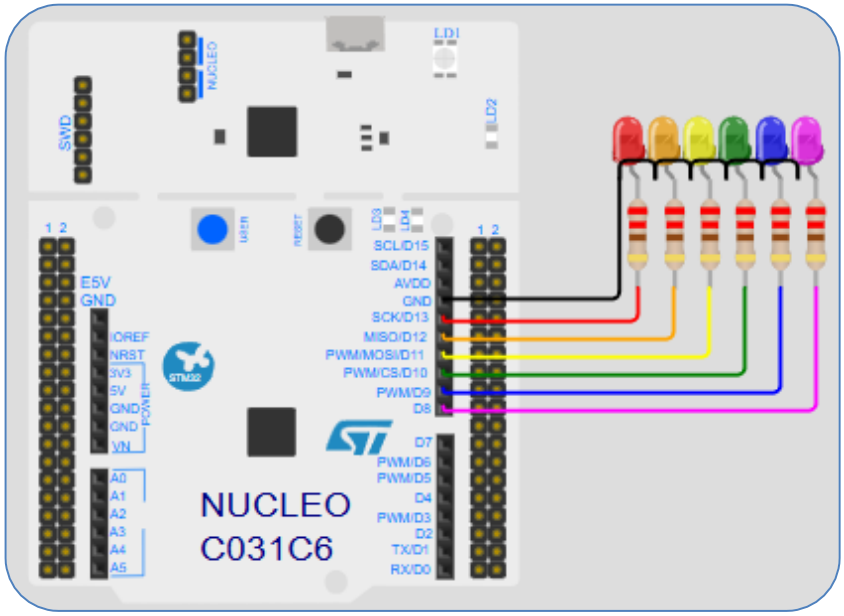


Рис. 1.13. Схема з STM32


```
{"version": 1, "author": "", "editor": "wokwi", "parts": [{"type": "board-st-nucleo-c031c6", "id": "nucleo", "top": 17.68, "left": -60.07, "attrs": {}}, {"type": "wokwi-led", "id": "led1", "top": 63.6, "left": 243.8, "attrs": {"color": "red"}}, {"type": "wokwi-led", "id": "led2", "top": 63.6, "left": 263, "attrs": {"color": "orange"}}, {"type": "wokwi-led", "id": "led3", "top": 63.6, "left": 282.2, "attrs": {"color": "yellow"}}, {"type": "wokwi-led", "id": "led4", "top": 63.6, "left": 301.4, "attrs": {"color": "green"}}, {"type": "wokwi-led", "id": "led5", "top": 63.6, "left": 320.6, "attrs": {"color": "blue"}}, {"type": "wokwi-led", "id": "led6", "top": 63.6, "left": 339.8, "attrs": {"color": "magenta"}}, {"type": "wokwi-resistor", "id": "r1", "top": 129.6, "left": 335.45, "rotate": 90, "attrs": {"value": "220"}}, {"type": "wokwi-resistor", "id": "r2", "top": 129.6, "left": 239.45, "rotate": 90, "attrs": {"value": "220"}}, {"type": "wokwi-resistor", "id": "r3", "top": 129.6, "left": 258.65, "rotate": 90, "attrs": {"value": "220"}}, {"type": "wokwi-resistor", "id": "r4", "top": 129.6, "left": 277.85, "rotate": 90, "attrs": {"value": "220"}}, {"type": "wokwi-resistor", "id": "r5", "top": 129.6, "left": 297.05, "rotate": 90, "attrs": {"value": "220"}}, {"type": "wokwi-resistor", "id": "r6", "top": 129.6, "left": 316.25, "rotate": 90, "attrs": {"value": "220"}}], "connections": [["$serialMonitor:TX", "nucleo:PA3", "", []], ["$serialMonitor:RX", "nucleo:PA2", "", []], ["led6:A", "uno:8", "green", ["v38.4", "h-19.6"]], ["led3:A", "uno:11", "green", ["v19.2", "h28.2"]], ["uno:12", "led2:A", "green", ["v-28.8", "h-18.7", "v-28.8"]], ["led1:A", "uno:13", "green", ["v38.4", "h28.8", "v19.2"]], ["led1:C", "uno:GND.1", "black", ["v48", "h48.1"]], ["led6:C", "led5:C", "black", ["v-9.6", "h-18.8"]], ["led5:C", "led4:C", "black", ["v-9.6", "h-18.8"]], ["led4:C", "led3:C", "black", ["v-9.6", "h-18.8"]], ["led3:C", "led2:C", "black", ["v-9.6", "h-18.8"]], ["led2:C", "led1:C", "black", ["v-9.6", "h-18.8"]], ["led5:A", "uno:9", "green", ["v28.8", "h-9.6", "v28.8"]], ["led4:A", "uno:10", "green", ["v0"]], ["led1:C", "esp:GND.1", "black", ["h-47.6", "v48"]], ["led1:A", "esp:13", "red", ["v0"]], ["led2:A", "esp:12", "orange", ["v9.6", "h-9.6"]], ["led3:A", "esp:11", "yellow", ["v9.6", "h-19.2"]], ["led4:A", "esp:10", "green", ["v19.2", "h-28.8"]], ["led5:A", "esp:9", "blue", ["v28.8", "h-38.4"]], ["led6:A", "esp:8", "purple", ["v38.4", "h-48"]], ["led1:C", "nucleo:GND.9", "black", ["v0"]], ["r2:2", "nucleo:D13", "red", ["v18", "h-104.74"]], ["r3:2", "nucleo:D12", "orange", ["v27.6", "h-123.94"]], ["r4:2", "nucleo:D11", "yellow", ["v37.2", "h-143.14"]], ["r5:2", "nucleo:D10", "green", ["v46.8", "h-162.34"]], ["r6:2", "nucleo:D9", "blue", ["v56.4", "h-181.54"]], ["r1:2", "nucleo:D8", "magenta", ["v66", "h-200.74"]], ["r2:1", "led1:A", "red", ["h0"]], ["led2:A", "r3:1", "orange", ["v0"]], ["led3:A", "r4:1", "yellow", ["v48"]], ["led4:A", "r5:1", "green", ["v0"]], ["led5:A", "r6:1", "blue", ["v0"]], ["led6:A", "r1:1", "magenta", ["v0"]]], "dependencies": {}}
```

- Для Raspberry Pi Pico (рис. 1.14):

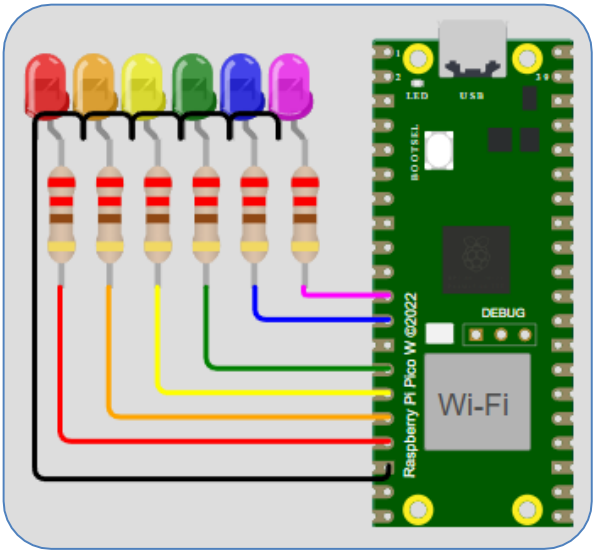


Рис. 1.14. Схема з Raspberry Pi Pico


```
{"version": 1, "author": "", "editor": "wokwi", "parts": [{"type": "board-pi-pico-w", "id": "pico",
"top": 1.15, "left": 75.45, "attrs": {}}, {"type": "wokwi-led", "id": "led1", "top": 6, "left": -73,
"attrs": {"color": "red"}}, {"type": "wokwi-led", "id": "led2", "top": 6, "left": -53.8, "attrs":
{"color": "orange"}}, {"type": "wokwi-led", "id": "led3", "top": 6, "left": -34.6, "attrs": {"color":
"yellow"}}, {"type": "wokwi-led", "id": "led4", "top": 6, "left": -15.4, "attrs": {"color": "green"}},
{"type": "wokwi-led", "id": "led5", "top": 6, "left": 3.8, "attrs": {"color": "blue"}}, {"type":
"wokwi-led", "id": "led6", "top": 6, "left": 23, "attrs": {"color": "magenta"}}, {"type":
"wokwi-resistor", "id": "r1", "top": 72, "left": 18.65, "rotate": 90, "attrs": {"value": "220"}},
{"type": "wokwi-resistor", "id": "r2", "top": 72, "left": -0.55, "rotate": 90, "attrs": {"value":
"220"}}, {"type": "wokwi-resistor", "id": "r3", "top": 72, "left": -19.75, "rotate": 90, "attrs":
{"value": "220"}}, {"type": "wokwi-resistor", "id": "r4", "top": 72, "left": -38.95, "rotate": 90,
"attrs": {"value": "220"}}, {"type": "wokwi-resistor", "id": "r5", "top": 72, "left": -58.15,
"rotate": 90, "attrs": {"value": "220"}}, {"type": "wokwi-resistor", "id": "r6", "top": 72, "left":
-77.35, "rotate": 90, "attrs": {"value": "220"}}], "connections": [["pico:GP0", "$serialMonitor:RX",
"", []], ["pico:GP1", "$serialMonitor:TX", "", []], ["led6:A", "uno:8", "green", ["v38.4", "h-19.6"]],
["led3:A", "uno:11", "green", ["v19.2", "h28.2"]], ["uno:12", "led2:A", "green", ["v-28.8", "h-18.7",
"v-28.8"]], ["led1:A", "uno:13", "green", ["v38.4", "h28.8", "v19.2"]], ["led1:C", "uno:GND.1",
"black", ["v48", "h48.1"]], ["led6:C", "led5:C", "black", ["v-9.6", "h-18.8"]], ["led5:C", "led4:C",
"black", ["v-9.6", "h-18.8"]], ["led3:C", "led2:C", "black", ["v-9.6", "h-18.8"]], ["led2:C",
"led1:C", "black", ["v-9.6", "h-18.8"]], ["led5:A", "uno:9", "green", ["v28.8", "h-9.6", "v28.8"]],
["led4:A", "uno:10", "green", ["v0"]], ["led1:C", "esp:GND.1", "black", ["h-47.6", "v48"]], ["led1:A",
"esp:13", "red", ["v0"]], ["led2:A", "esp:12", "orange", ["v9.6", "h-9.6"]], ["led3:A", "esp:11",
"yellow", ["v9.6", "h-19.2"]], ["led4:A", "esp:10", "green", ["v19.2", "h-28.8"]], ["led5:A", "esp:9",
"blue", ["v28.8", "h-38.4"]], ["led6:A", "esp:8", "purple", ["v38.4", "h-48"]], ["pico:GND.4",
"led1:C", "black", ["v5.24", "h-139.5"]], ["led4:C", "led3:C", "black", ["v-9.6", "h-19.2"]],
["pico:GP8", "r1:2", "magenta", ["h0"]], ["pico:GP9", "r2:2", "blue", ["h0"]], ["pico:GP10", "r3:2",
"green", ["h0"]], ["pico:GP11", "r4:2", "yellow", ["h0"]], ["pico:GP12", "r5:2", "orange", ["h0"]],
["pico:GP13", "r6:2", "red", ["h0"]], ["r6:1", "led1:A", "red", ["h0"]], ["led2:A", "r5:1", "orange",
["v0"]], ["led3:A", "r4:1", "yellow", ["v0"]], ["led4:A", "r3:1", "green", ["v0"]], ["led5:A", "r2:1",
"blue", ["v0"]], ["led6:A", "r1:1", "magenta", ["v0"]]], "dependencies": {}}
```


3. Виконайте завдання №4.

Оформлення звіту та порядок його подання

1. Увійдіть в дистанційний курс Moodle з власного аккаунту.
2. Відкрийте активність присвячену даній лабораторній роботі.
3. Натисніть кнопку «Здати роботу».
4. У текстове поле завантажте:
   - посилання на проекти Wokwi з виконаними завданнями;
   - висновки до лабораторної роботи.
5. Натисніть кнопку «Зберегти».

Контрольні запитання

1. Що таке онлайн-симуляція та які її переваги при розробці вбудованих систем?
2. Які стандартні функції Arduino ви використовували у програмі і для чого вони призначені?
3. Які регістри мікроконтролера Atmega328 використовуються для реалізації функцій, аналогічних функціям Arduino?
4. Які основні відмінності між програмуванням з використанням готових наборів функцій та взаємодією безпосередньо з регістрами?
5. Що таке переносимість коду?

ЛАБОРАТОРНА РОБОТА № 2

Тема: Контроль потоку програми

Мета: Використовуючи оператори, функції та механізми контролю потоку програми, навчитися писати структурований код для динамічної взаємодії з апаратними компонентами.

План проведення лабораторної роботи

- Ознайомлення з короткими теоретичними відомостями.
- Виконання завдання згідно порядку виконання роботи.
- Подання звіту в Moodle (кнопка «Здати роботу»).

Короткі теоретичні відомості

Потік програми – це послідовність інструкцій, які виконуються під час роботи програми.

Контроль потоку програми – це механізм, який визначає порядок виконання інструкцій у програмі. Інакше кажучи, він керує тим, як і коли окремі частини коду будуть виконуватися під час роботи програми. Це досягається за допомогою різних конструкцій програмування, таких як оператори умови, цикли, функції.

Оператори умови (`if`, `else`, `switch`) – дозволяють змінювати порядок виконання інструкцій залежно від певних умов.

Оператор `if`

Оператор **`if`** перевіряє умову, і тільки якщо вона істинна, виконується певний блок коду.

Синтаксис:

```
if (Умова_для_перевірки)
{
    код_який_буде_виконуватись;
}
```

- ***`Умова_для_перевірки`*** – це вираз, який містить логічні або ж арифметичні оператори.

Приклад:

```
int temperature = 20;

if (temperature > 25)
{
    turnOnConditioner();       // Якщо температура більше 30, то виконується цей код
}
```

В даному прикладі умова – це **`temperature > 25`**. Тільки якщо значення змінної **`temperature`** буде більше за **`25`** (коли умова є істинною та має значення **`true`**) – виконається код у фігурних дужках. Якщо умова хибна (**`false`**), блок коду у фігурних дужках пропускається.

Оператор **`if`** є одним з основних інструментів для контролю потоку програми, дозволяючи приймати рішення на основі даних і змінних.

Оператор else

Оператор **else** використовується разом з оператором **if** для виконання альтернативного блоку коду, якщо умова в **if** виявилася хибною (**false**).

Основна структура **if-else**:

```
if (Умова_для_перевірки)
{
    Обчислення_1;
}
else
{
    Обчислення_2;
}
```

- ***Обчислення_1*** – фрагмент програми який виконається тільки в тому випадку, коли ***Умова_для_перевірки*** поверне значення **true**.
- ***Обчислення_2*** – фрагмент програми який виконається тільки в тому випадку, коли результатом ***умова_для_перевірки*** буде **false**.

Приклад:

```
int temperature = 20;

if (temperature > 25)
{
    turnOnConditioner();       // Якщо температура більше 30, то виконується цей код
}
else
{
    turnOffConditioner();      // Якщо температура менше 25, виконується цей код
}
```

В даному прикладі буде виконуватись лише код, який знаходиться у фігурних дужках після **else**, оскільки значення змінної **temperature** менше 25.

Тернарний оператор

Використовується з тією ж метою, що й **if**, але з ним код буде займати всього один рядок.

Синтаксис:

```
Вираз ? Дія_1 : Дія_2;
```

Приклад:

```
(temperature > 25) ? Serial.print("Спекотно") : Serial.print("Нормально");
```

Якщо **temperature** більше 25, виведеться «**Спекотно**», якщо ні – «**Нормально**».

Оператор switch

Оператор **switch** використовується для вибору одного з декількох варіантів дій на основі значення певного виразу. Він працює з цілими числами або символами і дозволяє уникнути написання великої кількості умов **if-else**.

Структура **switch** включає кілька випадків (**case**), кожен з яких відповідає певному значенню виразу. Коли значення виразу співпадає з одним із випадків, виконується відповідний блок коду. Оператор **break** використовується для виходу з **switch** після виконання потрібного

коду, щоб запобігти виконанню наступних випадків. Якщо жоден випадок не підходить, можна використовувати блок **`default`**, який виконається за замовчуванням.

Синтаксис:

```
switch (Вираз)
{
  case Значення_1:
    Дія_1;              // Код, який виконається, якщо Вираз == Значення_1
    break;
  case Значення_2:
    Дія_2;              // Код, який виконається, якщо Вираз == Значення_2
    break;
  default:
    Дія_3;              // Код, який виконається, якщо жоден з випадків не підходить
    break;
}
```

Приклад:

```
int number = 2;

switch (number) {
  case 1:
    Serial.println("One");
    break;
  case 2:
    Serial.println("Two");
    break;
  case 3:
    Serial.println("Three");
    break;
  default:
    Serial.println("Unknown");
    break;
}
```

В даному прикладі, змінна **`number`** дорівнює 2, тому, в результаті виконання **`switch`**, виведеться **`Two`**, а для всіх інших значень number – виведеться **`Unknown`**.

Використання підпрограм

Підпрограма – це блок коду, який виконує певну задачу і може бути викликаний з різних частин програми. Підпрограми використовуються для розділення великого коду на менші, більш керовані частини, що полегшує читання і повторне використання коду.

Типи підпрограм:

- **Функції** – підпрограми, які повертають значення.
- **Процедури** – підпрограми, які не повертають значення, а просто виконують певні дії.
- **Методи** – підпрограмами у об'єктно-орієнтованих мовах програмування, що визначаються в межах класів і можуть працювати з даними об'єкта.

Оголошення (опис) та **виклик** підпрограми виконуються в різних частинах програми. Оголошення зазвичай розташоване у заголовкових файлах (наприклад, .h файли) або на початку коду, а виклик підпрограми відбувається в тілі основної програми або в інших підпрограмах.

Синтаксис:

```
// Реалізація підпрограми

return_type name(parameter_list)
{
    // Тіло підпрограми
    // ...
    return return_value;          // (не використовується, якщо тип повернення void)
}
```

- Тип повернення (**return_type**) – вказує тип значення, яке підпрограма поверне (може бути **void**, якщо нічого не повертає).
- Назва підпрограми (**name**) – ім'я, яке буде використовуватись для виклику підпрограми.
- Параметри (**parameter_list**) – список змінних, які передаються підпрограмі. Їх може не бути.
- Тіло – код, який виконується при виклику підпрограми.
- Оператор повернення (**return**) – вказує значення (**return_value**), яке повертається функцією або методом. Значення **return_value** має бути типу **return_type**. Якщо типом повернення є **void** – оператор **return** не використовується.

Приклад:

```
const int led1 = 8;
const int led2 = 9;


// Реалізація підпрограми Arduino - setup
void setup()
{
  pinMode(led1, OUTPUT);    // Виклик підпрограми для налаштування піну 8
  pinMode(led2, OUTPUT);    // Виклик підпрограми для налаштування піну 9
}

// Реалізація підпрограми Arduino - loop
void loop()
{
  blink(led1);              // Виклик підпрограми для блимання першим світлодіодом
  blink(led2);              // Виклик підпрограми для блимання другим світлодіодом
}


// Реалізація підпрограми для блимання світлодіода
void blink(int led)
{
  digitalWrite(led, HIGH);  // Виклик підпрограми для включення світлодіода
  delay(500);               // Виклик підпрограми для затримки

  digitalWrite(led, LOW);   // Виклик підпрограми для вимикання світлодіода
  delay(500);               // Виклик підпрограми для затримки
}
```

У програмуванні для Arduino-сумісних систем підпрограми **setup** та **loop** мають особливу роль і їх виклик виконується системою Arduino. При запуску системи або її перезавантаженні, спочатку (один раз) виконується підпрограма **setup**, а далі, безперервно виконується підпрограма **loop** (і весь код розміщений в ній).

В наведеному прикладі в підпрограмі **loop** двічі виконується виклик підпрограми **blink**, яка реалізована нижче в коді. У наведеному прикладі підпрограма **blink** має такі характеристики:

- Тип повернення – **void** (жодне значення не повертається).
- Назва – **blink**.
- Параметри – **int led** (один параметр типу **int**).
- Тіло підпрограми містить 4 рядки коду з викликом інших підпрограм Arduino.

Підпрограма **digitalWrite** [5] у середовищі програмування Arduino використовується для виведення цифрових даних на пін мікроконтролера. Вона дозволяє встановити рівень логічного сигналу (високий або низький) на конкретному піні.

Синтаксис:

```
digitalWrite(pin, value);
```

- **pin** – номер піна, на якому буде встановлено значення.
- **value** – значення, яке потрібно записати на пін (**HIGH** або **LOW**). **HIGH** – встановлює високий рівень (зазвичай 5 V), **LOW** – встановлює низький рівень (0 V).

Цикли

Цикли – це конструкції, які дозволяють виконувати один і той самий блок коду кілька разів. Вони є потужним інструментом для оптимізації коду і повторювальних завдань, дозволяючи програмістам ефективно управляти процесами, які потрібно повторювати.

Основні типи циклів:

1) **for** – використовується, коли кількість повторень відома заздалегідь.

   Синтаксис:

   ```
   for (ініціалізація; умова; оновлення)
   {
       // код для виконання
   }
   ```

   Приклад:

   ```
   for (int i = 0; i < 5; i++)           // Цикл, що виводить числа від 0 до 4
   {
       Serial.println(i);
   }
   ```

2) **while** – виконує код поки умова залишається істинною (**true**).

   Синтаксис:

   ```
   while (умова)
   {
       // код для виконання
   }
   ```

Приклад:

```
int i = 0;
while (i < 5)                          // Виводить числа від 0 до 4
{
    Serial.println(i);
    i++;
}
```

3) **do-while** – подібний до **while**, але спочатку виконується блок коду, а потім перевіряється умова. Гарантується виконання блоку коду принаймні один раз.

Синтаксис:

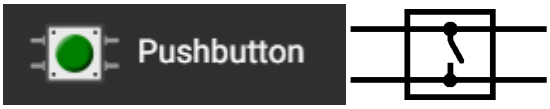
```
do
{
    // код для виконання
}
while (умова);
```

Приклад:

```
int i = 0;
do                                    // Виводить числа від 0 до 4
{
    Serial.println(i);
    i++;
}
while (i < 5);
```

Робота з компонентами

Кнопка – це простий механічний компонент (рис. 2.1), який використовується для вводу сигналів або команд в електронні системи. Кнопки зазвичай мають два стани: натиснутий і не натиснутий.



а б

Рис. 2.1. Зовнішній вигляд кнопки (а) та її внутрішня схема (б)

Приклад програми для роботи з кнопкою:

```
const int buttonPin = 2;    // Пін, до якого підключена кнопка
const int ledPin = 13;      // Пін, до якого підключений світлодіод

int buttonState = 0;        // Змінна для зберігання стану кнопки


void setup()
{
  pinMode(buttonPin, INPUT);    // Встановлення піна кнопки як вхідного
  pinMode(ledPin, OUTPUT);      // Встановлення піна світлодіода як вихідного
}
```

```
void loop()
{
  buttonState = digitalRead(buttonPin); // Зчитування стану кнопки
  if (buttonState == HIGH)      // Перевірка стану кнопка
  {
    digitalWrite(ledPin, HIGH); // Увімкнути світлодіод
  }
  else
  {
    digitalWrite(ledPin, LOW);  // Вимкнути світлодіод
  }
  delay (200);                  // Затримка для усування вібрації контактів
}
```

При використанні механічних кнопок існує такий ефект, як «**вібрація контактів**» (англ. contact bounce). При одному фізичному натисканні на кнопку електричний контакт не стабільно замикається чи розмикається і це може бути сприйнято як кілька натискань. Існують різні варіанти для уникнення вібрації контактів, одним з яких є додавання затримки в коді між зчитуваннями стану кнопки. Тому в наведеному вище прикладі, в підпрограму **`loop`** доцільно додати невелику затримку (**`delay`**).

Сегментний індикатор

Сегментний індикатор – це електронний дисплей, який використовується для відображення чисел або символів. Він складається з декількох сегментів (напр. світлодіодів), зазвичай розташованих у формі цифри «8». Детальніше про семисегментні індикаторами в симуляторі (рис. 2.2) можливо дізнатись з документації Wokwi [6].

Рис. 2.2 — Варіанти сегментних індикаторів у симуляторі Wokwi

Основними типами світлодіодних сегментних індикаторів є індикатори з загальним катодом та загальним анодом (рис. 2.3). Вони мають різну конфігурацію підключення світлодіодів, що впливає на спосіб керування ними.

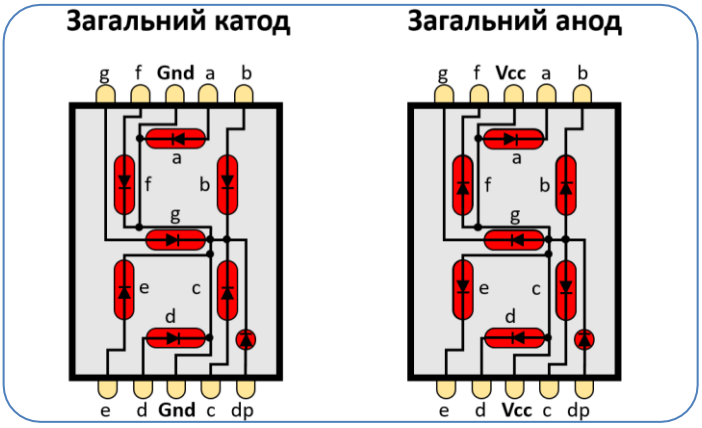


Рис. 2.3 — Схеми світлодіодних індикаторів з загальним катодом та анодом

У цій роботі необхідно налаштувати динамічну індикацію за допомогою 7-ми сегментного дисплею з 4 цифрами. Зазвичай такий дисплей має 12 пінів та не потребує підключення землі, 5 В чи 3,3 В. Для формування будь-якої цифри використовуються 7 пінів, ще один — для встановлення десяткової крапки. Інші 4 з 12 пінів керують кожною з 4-х цифр на дисплеї (їх необхідно підключати через резистори).

Динамічна індикація — це метод керування сегментними індикаторами, коли кілька цифр (або символів) по черзі швидко включаються і вимикаються. Завдяки цьому створюється ілюзія, що всі цифри світяться одночасно, хоча насправді в кожен момент часу активна лише одна. Це дозволяє економити ресурси та зменшувати кількість задіяних виводів.

Динамік

П'єзодинамік – це пристрій, що використовує п'єзоелемент для створення звуку. Коли на п'єзоелемент подається змінна електрична напруга, він починає вібрувати, створюючи звукові хвилі. Це простий і ефективний спосіб створювати звукові сигнали в електронних пристроях.

Функція **tone** від Arduino використовується для генерації квадратної хвилі на певному піні, що може створювати звуковий сигнал на підключеному до цього піна п'єзодинаміку.

Синтаксис:

```
tone(pin, frequency);
tone(pin, frequency, duration);

//pin - номер порту входу/виходу, на якому буде генеруватися сигнал
//frequency - частота сигналу в Герцах
//duration - тривалість сигналу в мілісекундах
```

Приклад програми для генерації звукового сигналу:

```
#define piezoPin 3      //номер піна до якого підключено п'єзодинамік

void setup()
{
   pinMode(buzzer, OUTPUT);
}

void loop()
{
  tone(piezoPin, 1000); // Генерується сигнал з частотою 1000 Гц
  delay(1000);          // Тривалість звучання 500 мс

  noTone(piezoPin);     // Генерація звуку зупиняється
  delay(1000);
}
```

Завдання

1. У симуляторі Wokwi реалізуйте наведений в порядку виконання роботи приклад.
2. Модифікуйте приклад для створення програми, яка буде:
   - ✓ генерувати цифри від 0 до 9 на кожному з знаків 4-значного індикатора (одночасно або послідовно);
   - ✓ для отримання чотиризначного коду послідовно зупиняти генерацію на кожному з розрядів індикатора одразу після натискання кнопки;
   - ✓ отриманий код перевіряти на відповідність паролю (заданого в програмі константою, напр. «`const int password = 1234;`»), якщо пароль правильний – програти мелодію та відобразити пароль на еркані. Якщо пароль не правильний – перезапустити програму.

Примітка. Програма повинна моментально реагувати на натискання кнопки!

Додаткове завдання 1. Генерувати різні короткі звукові сигнали для правильного/неправильного вибору кожної цифри.

Додаткове завдання 2. Реалізувати завдання без використання функцій Arduino.

Порядок виконання роботи

Крок 1. Зібрати схему (рис. 2.4) або скопіювати наведений нижче текст у файл **`diagram.json`**.

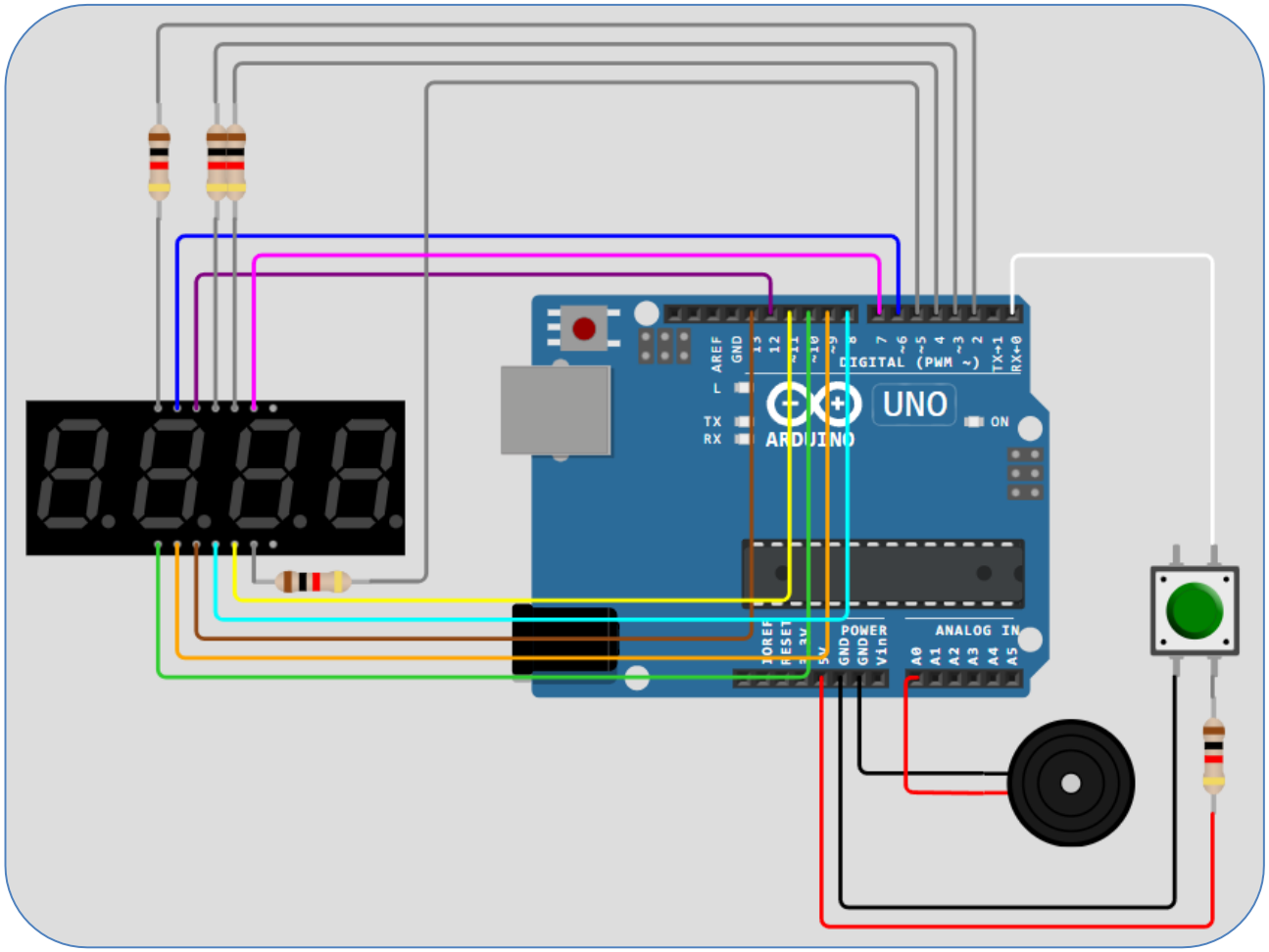


Рис. 2.4. Схема

```
{"version": 1, "author": "", "editor": "wokwi", "parts": [{"type": "wokwi-arduino-uno", "id": "uno", "top": 10.2, "left": 143.4, "attrs": {}}, {"type": "wokwi-7segment", "id": "sevseg1", "top": 63.42, "left": -94.88, "attrs": {"digits": "4", "colon": "", "common": "cathode"}}, {"type": "wokwi-resistor", "id": "r1", "top": 147.95, "left": 19.2, "attrs": {"value": "1000"}}, {"type": "wokwi-resistor", "id": "r2", "top": -62.4, "left": -19.75, "rotate": 90, "attrs": {"value": "1000"}}, {"type": "wokwi-resistor", "id": "r3", "top": -62.4, "left": -58.15, "rotate": 90, "attrs": {"value": "1000"}}, {"type": "wokwi-resistor", "id": "r4", "top": -62.4, "left": -29.35, "rotate": 90, "attrs": {"value": "1000"}}, {"type": "wokwi-buzzer", "id": "bz1", "top": 218.1, "left": 388.5, "rotate": 90, "attrs": {"volume": "0.1"}}, {"type": "wokwi-pushbutton", "id": "btn1", "top": 143.9, "left": 454.7, "rotate": 90, "attrs": {"color": "green"}}, {"type": "wokwi-resistor", "id": "r5", "top": 235.2, "left": 469.85, "rotate": 90, "attrs": {"value": "1000"}}], "connections": [["uno:6", "sevseg1:A", "blue", ["v-38.4", "h-361.1"]], ["uno:12", "sevseg1:F", "purple", ["v-19.2", "h-287.5"]], ["uno:10", "sevseg1:E", "limegreen", ["v182.4", "h-325.71"]], ["sevseg1:D", "uno:9", "orange", ["v56.76", "h325.6"]], ["sevseg1:DP", "uno:13", "#8f4814", ["v47.16", "h278"]], ["sevseg1:C", "uno:8", "cyan", ["v37.56", "h316.4"]], ["sevseg1:G", "uno:11", "yellow", ["v27.96", "h277.8"]], ["sevseg1:DIG4", "r1:1", "gray", ["v0"]], ["r1:2", "uno:5", "gray", ["h27.6", "v-249.6", "h245.8"]], ["sevseg1:DIG2", "r4:2", "gray", ["v0"]], ["sevseg1:DIG3", "r2:2", "gray", ["v0"]], ["r3:2", "sevseg1:DIG1", "gray", ["h0"]], ["uno:7", "sevseg1:B", "magenta", ["v-28.8", "h-313.2"]], ["r2:1", "uno:4", "gray", ["v-19.2", "h351.3"]], ["r4:1", "uno:3", "gray", ["v-28.8", "h370.4"]], ["r3:1", "uno:2", "gray", ["v-38.4", "h408.7"]], ["uno:GND.3", "bz1:1", "black", ["v0"]], ["bz1:2", "uno:A0", "red", ["v-0.4", "h-38.4", "v-57.5"]], ["btn1:1.r", "r5:1", "gray", ["v0"]], ["uno:GND.2", "btn1:2.r", "black", ["v115.1", "h-120.7"]], ["r5:2", "uno:5V", "red", ["v56.4", "h-195.8"]], ["btn1:1.l", "uno:0", "white", ["v-144", "h-100.3"]]], "dependencies": {}}
```


Крок 2. Дослідити приклад:

1. Завантажити зміст файлу Example2.cpp [4] у редактор коду.
2. Перевірити працездатність програми, а саме: після натискання кнопки повинен запускатися таймер, а динамік кожну секунду повинен видавати звуковий сигнал зі зростаючою частотою. Після повторного натискання кнопки таймер повинен вимикатися, а динамік програвати мелодію.
3. Дослідити реалізацію потоку програми на основі алгоритму зображеного на рисунку 2.5.

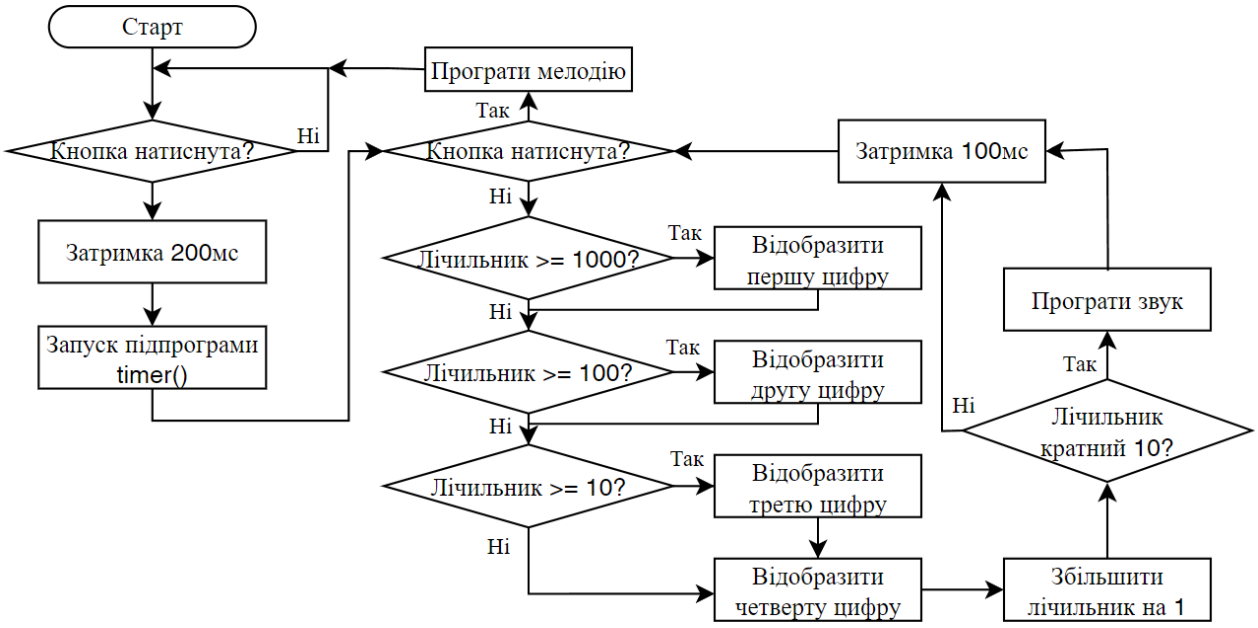


Рис. 2.5. Алгоритм програми з прикладу

Крок 3. Виконання завдання № 2 згідно алгоритму на рисунку 2.6.

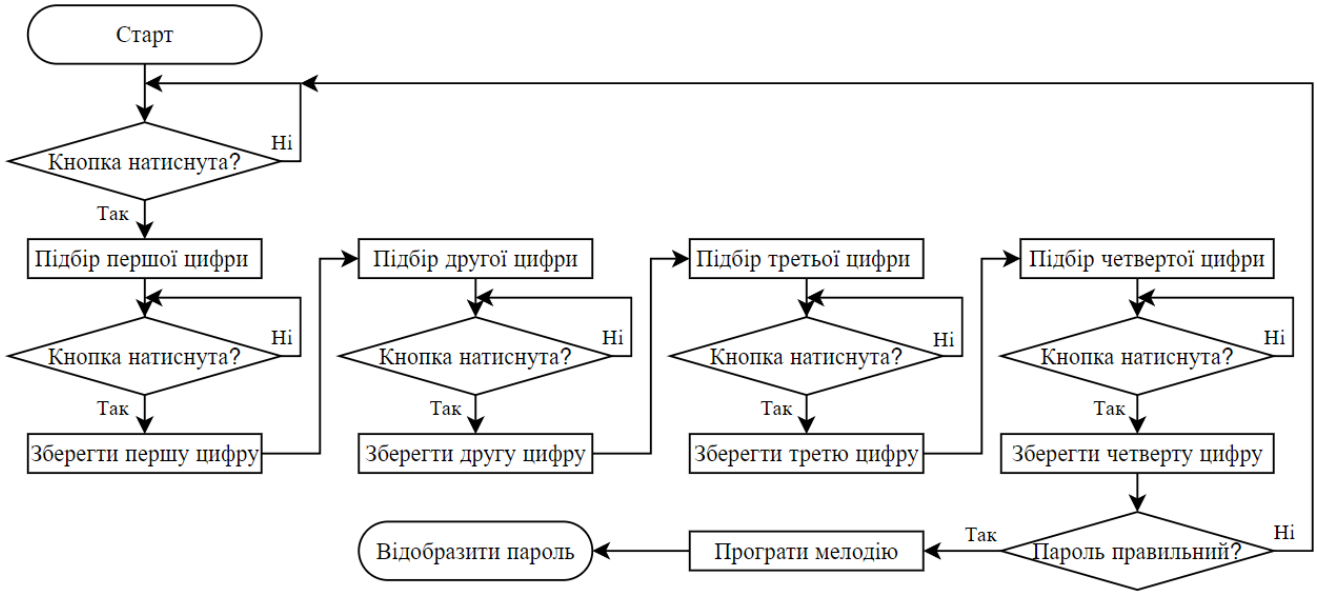


Рис. 2.6. Алгоритм програми з завдання

Оформлення звіту та порядок його подання

1. Увійдіть в дистанційний курс Moodle з власного аккаунту.
2. Відкрийте активність присвячену даній лабораторній роботі.
3. Натисніть кнопку «Здати роботу».
4. У текстове поле завантажте:
   - посилання на створений проект в симуляторі Wokwi з виконаними завданнями;
   - висновки до лабораторної роботи.
5. Натисніть кнопку «Зберегти».

Контрольні запитання

1. Що таке підпрограми і які переваги надає їх використання?
2. Які засоби використовуються для контролю потку програми?
3. У чому відмінність статичної та динамічної індикації?
4. Як реалізувати декілька завдань уникаючи зайвих затримок? Як в процесі динамічної індикації миттєво реагувати на будь-яку іншу подію (натискання кнопки)?

ЛАБОРАТОРНА РОБОТА № 3

Тема: Внутрішньосистемні та міжсистемні комунікаційні протоколи

Мета: Навчитися створювати програми, які використовують протоколи UART, SPI та I2C, для передачі даних між мікроконтролером та периферійними пристроями.

План проведення лабораторної роботи

- Ознайомлення з короткими теоретичними відомостями.
- Виконання завдання згідно порядку виконання роботи.
- Подання звіту в Moodle (кнопка «Здати роботу»).

Короткі теоретичні відомості

Univsersal Asynchronos Reciever-Transmitter (UART)

Універсальний асинхронний приймач-передавач (UART, Universal Asynchronous Receiver-Transmitter) — це апаратний модуль, що використовується для асинхронної серійної передачі та прийому даних між двома пристроями (мікроконтролерами, комп'ютерами, або іншими пристроями з низькою швидкістю передачі даних).

Для кожного напрямку комунікації передбачено окремий вивід: TX – для передачі даних; RX – для прийому. Під час використання UART відповідні виводи, TX (1) та RX (0) (рис. 3.1), не доцільно використовувати для інших задач.

Рис. 3.1. Виводи UART на платі Arduino UNO та ESP32

Як працює UART

Передача даних здійснюється послідовно, без синхронізації (без використання спільного тактового сигналу) між передавачем та приймачем. Замість цього, обидва пристрої повинні бути налаштовані на однакову швидкість передачі даних, кількість біт даних (в одному «пакеті»), кількість стоп-бітів, наявність або відсутність біта парності.

Дані відправляються по одному біту за раз (рис. 3.2). Спочатку йде стартовий біт, потім біти даних, біт парності (якщо використовується) та стоп-біти в кінці.

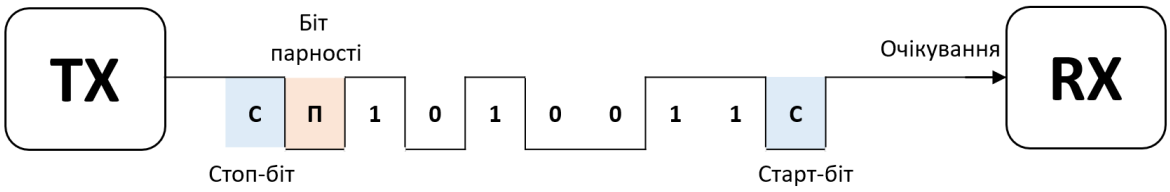


Рис. 3.2. Послідовність передачі даних з використанням UART

Клас Serial

Комунікацію з використанням UART можливо налаштувати за допомогою класу Serial [7], який включає відповідні методи, основні з яких:

- **`Serial.begin(`***`speed`***`)`** або **`Serial.begin(`***`speed, config`***`)`**, де:
  - *`speed`* – швидкість в бітах на секунду (baudrate);
  - *`config`* – це опціональний параметр, що дозволяє налаштувати кількість біт даних, перевірку парності і стопові біти. За замовчуванням, кадр складається з 8 біт даних, без перевірки парності та з одним стоповим бітом.

  Метод **`begin`** задає швидкість передачі даних по послідовному інтерфейсу. Для взаємодії з ПК рекомендується використовувати одну з попередньо встановлених швидкостей: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115200.

- **`Serial.print(`***`val`***`)`** та **`Serial.print(`***`val, format`***`)`**, де:
  - *`val`* – значення, яке необхідно вивести (число, символ, рядок або будь-які інші дані);
  - *`format`* – це опціональний параметр, який визначає формат виводу, тобто, систему числення (DEC, HEX, OCT, BIN, BYTE) для цілочисельних типів, а також кількість десяткових знаків після коми для чисел з плаваючою комою.

  Метод **`print`** використовується для виведення інформації в послідовний порт. Це означає, що дані, передані за допомогою нього, будуть відправлені через UART на інший пристрій або в серійний монітор на комп'ютері.
  Метод **`println`** додає до даних символ переносу рядка, щоб кожне повідомлення відображалося в окремому рядку.

- **`Serial.available()`** – перевіряє, чи є вхідні дані. Цей метод не зчитує дані, а тільки повертає кількість байтів доступних для читання, які вже були прийняті і зберігаються в буфері послідовного порту.
- **`Serial.read()`** – використовується для зчитування першого байта даних, доступного в буфері послідовного порту. Якщо даних для зчитування немає, метод повертає -1.
- **`Serial.readStringUntil('\n')`** – дозволяє зчитувати дані з послідовного порту до появи певного роздільного символу (delimiter).
- **`Serial.write(`***`val`***`)`** – використовується для відправки даних через послідовний порт у вигляді байтів, на відміну від **`print`** і **`println`**, які призначені для виведення даних у форматі тексту.

Приклад:

```
byte inputByte = 0;

void setup()
{
  Serial.begin(4800);
  Serial.println("Введіть дані в серійний монітор та натисніть Enter!");
}

void loop()
{
  if (Serial.available() > 0)
  {
    inputByte = Serial.read();
    Serial.print("З буфера отримано: ");
    Serial.write(inputByte);
    Serial.println("");
  }
}
```

Цей приклад демонструє просту програму, яка здійснює обмін даними з комп'ютером через послідовний порт. Програма спочатку ініціалізує послідовний зв'язок, а потім у циклі **`loop`** перевіряє, чи є вхідні дані в серійному порту. Якщо є, то вона зчитує один байт, відправляє повідомлення про отримання цього байта і виводить його у серійний порт. Таким чином, кожного разу, при вводі будь-якого символу у серійний монітор, його буде прочитано та відображено.

Онлайн-симулятор Wokwi не підтримує одночасну симуляцію декількох мікроконтролерів (МК), але приклади коду для такої комунікації, з використанням UART, можливо знайти в офіційній документації [8].

Serial Peripheral Interface (SPI)

Послідовний периферійний інтерфейс (SPI, Serial Peripheral Interface) – це високошвидкісний послідовний (серійний) протокол зв'язку, який використовується для обміну даними між мікроконтролерами та периферійними пристроями (наприклад, датчиками, дисплеями, пам'яттю і т. д.).

Пристрої, що беруть участь у комунікації по SPI, поділяються на дві основні категорії:

- Master (Майстер) – головний пристрій, який ініціює комунікацію з іншими пристроями. Він контролює тактовий сигнал (SCLK) і визначає, коли саме відбувається передача даних та з яким підлеглим пристроєм (Slave) буде взаємодіяти.
- Slave (Підлеглий) – це пристрій, який реагує на ініціативу майстра. Він отримує тактовий сигнал від майстра і передає або отримує дані лише тоді, коли майстер активує його за допомогою лінії SS/CS (Slave Select або Chip Select). Підлеглих пристроїв може бути декілька, кожен з них активується окремою лінією SS/CS.

Лінії які застосовуються при комунікації за допомогою SPI:

- MISO (Master In Slave Out) – лінія для передачі даних від підлеглого пристрою до майстра.
- MOSI (Master Out Slave In) – лінія для передачі даних від майстра до підлеглого пристрою.
- SCK (Serial Clock) – тактова лінія, яка синхронізує обмін даними.
- SS (Slave Select) – лінія вибору підлеглого пристрою. Майстер активує її для вибору конкретного підлеглого для передачі даних.

Дізнатись які виводи мікроконтролера відповідають за ці лінії можливо з документації або просто вивести інформацію про це в серійний монітор за допомогою наступного коду:

```
Serial.print("MOSI: ");
Serial.println(MOSI);
Serial.print("MISO: ");
Serial.println(MISO);
Serial.print("SCK: ");
Serial.println(SCK);
Serial.print("SS: ");
Serial.println(SS);
```

Для Arduino UNO та ESP32 відповідні піни відмічені на рисунку 3.3.

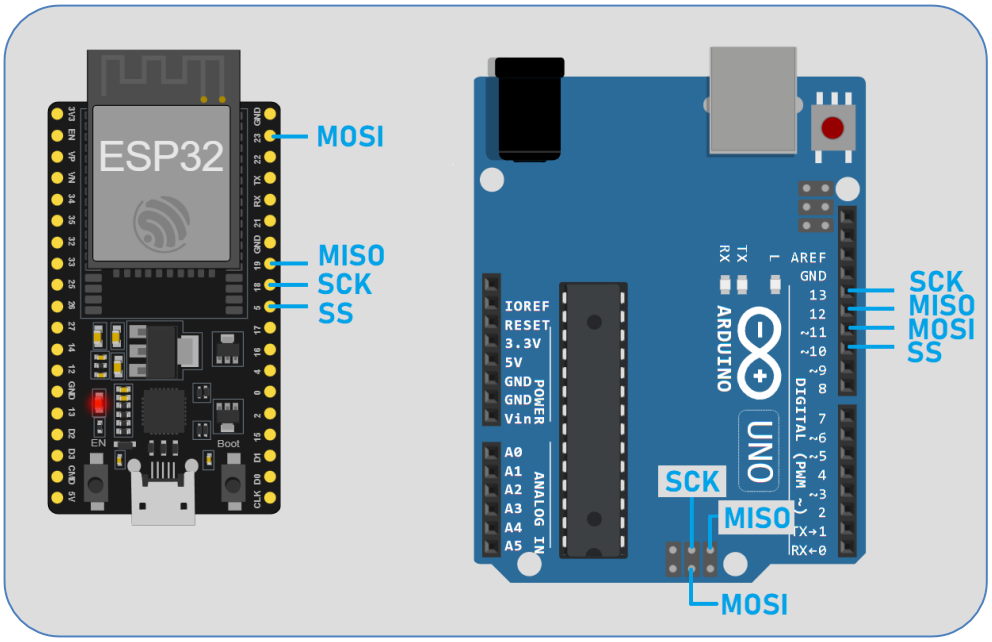


Рис. 3.3. Лінії передачі даних через SPI

Щоб почати взаємодіяти з пристроєм по SPI потрібно виконати кілька основних кроків:

- Підключити пристрій до відповідних пінів (MOSI, MISO, SCK, SS).
- Імпортувати бібліотеку SPI.h, яка значно полегшує роботу по SPI:

```
#include <SPI.h>
```

- Налаштувати пін **SS** як вихідний:

```
pinMode(SS, OUTPUT);
```

- Налаштувати такі параметри, як: тактова частота, режим SPI, порядок бітів:

```
SPI.beginTransaction(SPISettings(14000000, MSBFIRST, SPI_MODE0));
```

  - Перший параметр (частота) – це частота тактового сигналу SPI, яка буде використовуватись для комунікації. Вона вимірюється Гц та зазвичай обирається в залежності від характеристик підлеглого пристрою, наприклад, якщо він має номінальну частоту 14 МГц, то – перший параметр необхідно встановити 14 000 000.
  - Другий параметр (порядок бітів) – визначає, яким чином передаються дані: старший (**`MSBFIRST`**) або молодший (**`LSBFIRST`**) біт першим. Вибір залежить від того, як підлеглий пристрій очікує отримувати дані.
  - Третій параметр (режим SPI) – визначає яку полярність і фазу має тактовий сигнал SPI. Всього є чотири режими **`SPI_MODE`**. Детальніше про це можливо дізнатись в відповідній документації [9].
    - В основному циклі **`loop`** (або в окремих функціях) написати код для обміну даними з підлеглим пристроєм. Щоб активувати підлеглий пристрій, майстер встановлює пін SS у низький рівень (**`digitalWrite(SS, LOW)`**), що сигналізує підлеглому про початок передачі даних. Дані передаються за допомогою методу **`SPI.transfer()`**. Кожен виклик **`SPI.transfer()`** передає один байт даних від майстра до підлеглого і одночасно отримує один байт даних від підлеглого. Після завершення обміну даними майстер деактивує підлеглого, встановлюючи пін SS у високий рівень (**`digitalWrite(SS, HIGH)`**).
- **`SPI.endTransaction()`** – завершує транзакцію і дозволяє іншим пристроям, , якщо вони є, використовувати SPI.
- Якщо більше не потрібно працювати з SPI, можна викликати **`SPI.end()`**, щоб звільнити пін SPI і відключити шину.

Приклад: RGB-світлодіод, через драйвер NLFS595, підключено до мікроконтролера (рис. 3.4).

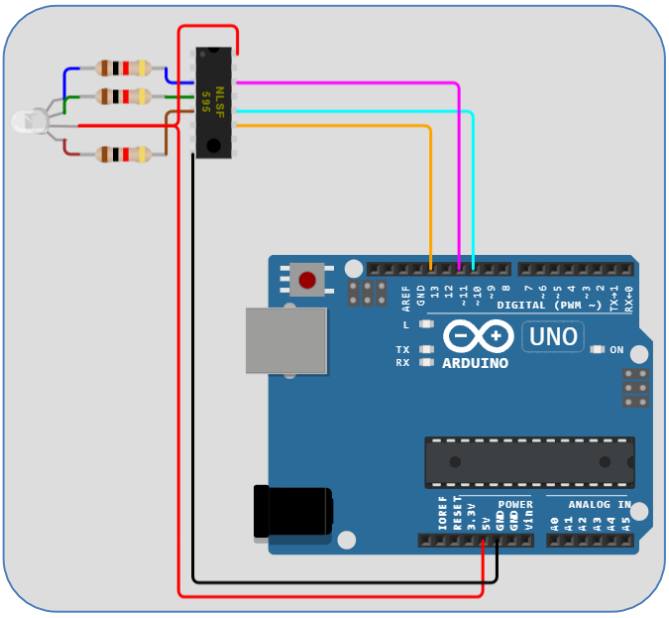


Рис. 3.4. Схема підключення NLFS595 до Arduino UNO

Щоб реалізувати блимання RGB-світлодіода кольором Aqua, до NLFS595, по SPI, надсилається значення 0010 0100, а через секунду – значення 1111 1111:

```
#include <SPI.h>


#define chip1 10  // NLSF595


/*  NLSF595 MOSI - SI
            SCLK - SCK
            SS   - RCK  */


void setup()
{
  pinMode(chip1, OUTPUT);
  SPI.begin();
  SPI.beginTransaction(SPISettings(14000000, LSBFIRST, SPI_MODE0));
}

void loop()
{
  // Aqua Color
  digitalWrite(chip1, LOW);
  SPI.transfer(0b00100100);
  digitalWrite(chip1, HIGH);
  delay(1000);

  // LED Off
  digitalWrite(chip1, LOW);
  SPI.transfer(0b11111111);
  digitalWrite(chip1, HIGH);
  delay(1000);
}
```

Детальніше про SPI можливо дізнатись зі статті за посиланням [10]. Також, в документації можливо знайти значну кількість прикладів з використанням SPI, наприклад:

- Зчитування тиску та температури з давача SCP1000 [11].
- Управління цифровим потенціометром AD5206 [12].

Inter-Integrated Circuit (I2C)

I2C (Inter-Integrated Circuit) — це двопровідний серійний інтерфейс, який використовує лінії SDA (Serial Data Line) і SCL (Serial Clock Line) (рис. 3.5):

- SDA – використовується для передачі даних між Master і Slave пристроями.
- SCL – відповідає за передачу тактового сигналу, для синхронізацію обміну даними.

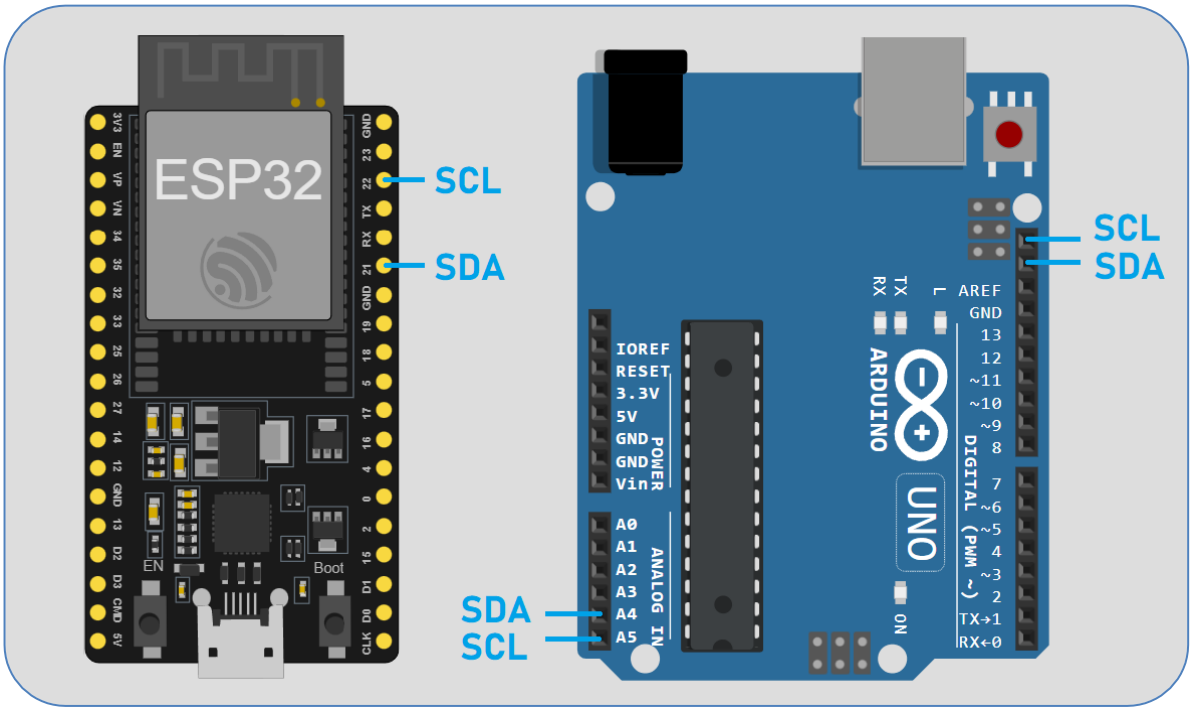


Рис. 3.5. Піни ESP32 та Arduino UNO для підключення ліній I2C

Кожен пристрій на шині I2C має унікальну адресу, що дозволяє головному пристрою (Master) керувати кількома підлеглими пристроями (Slave). Стандартна послідовність бітів при передачі даних за допомогою I2C (рис. 3.6) виглядає наступним чином:

1) **Start Condition**. Передача починається з генерації стартової умови (start condition). Для цього лінія SDA переходить з високого рівня на низький, поки SCL залишається високою, що сигналізує всім пристроям на шині, що розпочинається передача даних.
2) **7-бітна адреса** (інколи застосовується 10-біт). Master-пристрій надсилає адресу Slave-пристрою, з яким він хоче взаємодіяти.
3) **R/W біт** – визначає, чи буде Master читати (R = 1) або записувати (W = 0) дані.
4) **Біт підтвердження** (ACK, Acknowledge) від Slave. Якщо, в отриманій адресі, Slave-пристрій розпізнав свою адресу, він відповідає сигналом підтвердження (ACK). Для цього Slave встановлює лінію SDA на низький рівень (логічний нуль) протягом одного такту SCL.
5) **Передача даних** в 8-бітових блоках (байтах). Після кожного байта отримувач повинен надіслати підтвердження (ACK).
6) **Stop Condition**. Після завершення передачі даних Master генерує умови для зупинки встановлюючи лінію SDA з низького рівня на високий, поки SCL залишається високим, що сигналізує про завершення передачі.

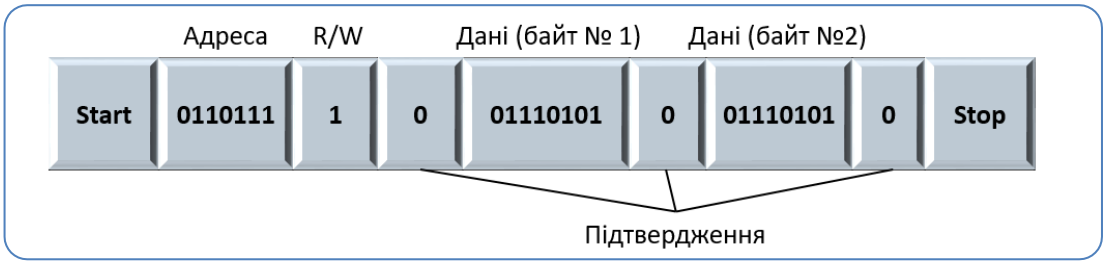


Рис. 3.6. Послідовність комунікації за допомогою I2C

Для роботи з I2C доцільно використовувати бібліотеку Wire [13], основні функції якої включають:

- **`Wire.begin()`** – Ініціалізує шину I2C. Використовується як для Master, так і для Slave-пристроїв.
- **`Wire.beginTransmission(`*`address`*`)`** – починає передачу даних до Slave-пристрою з вказаною I2C-адресою.
- **`Wire.write(`*`value`*`)`** – записує дані в буфер для передачі по I2C. Дані не відправляються одразу, а накопичуються в буфері.
- **`Wire.endTransmission()`** – завершує передачу даних і відправляє їх до Slave-пристрою.
- **`Wire.requestFrom(`*`address, quantity`*`)`** – запитує дані від Slave-пристрою з вказаною I2C-адресою. Після виклику цієї функції можна читати отримані дані.
- **`Wire.read()`** – зчитує один байт даних, отриманий через I2C. Використовується після функції Wire.requestFrom().
- **`Wire.onReceive(`*`function`*`)`** – встановлює функцію обробки даних, яка буде викликатися, коли Slave-пристрій отримує дані від Master.
- **`Wire.onRequest(`*`function`*`)`** – встановлює функцію обробки даних, яка буде викликатися, коли Master запитує дані від Slave.

Симулятор Wokwi підтримує декілька компонентів, які взаємодіють по I2C, наприклад:

- **MPU6050** – інерційний вимірювальний модуль, що об'єднує в собі трьохосьовий акселерометр і трьохосьовий гіроскоп. Мапа регістрів міститься в документі за посиланням [14].
- **LCD-дисплей.** Для взаємодії з LCD 1602/2004 по шині I2C необхідно як мінімум дві бібліотеки:
  - Wire – для роботи з I2C.
  - LiquidCrystal_I2C [15] – включає велику різноманітність методів для управління дисплеєм і дозволяє зробити код простішим і коротшим.

  Після підключення даних бібліотек необхідно створити об'єкт **`LiquidCrystal_I2C lcd(0x48, 16, 2);`** та використовувати його методи:

  - **`begin()`** – ініціалізує дисплей з вказаною кількістю стовпців і рядків. **`lcd.begin(16, 2);`** – ініціалізує дисплей з 16 стовпцями і 2 рядками.
  - **`setCursor(`*`col, row`*`)`** – встановлює позицію курсора в зазначеній колонці (col) і рядку (row).
  - **`print(`*`text`*`)`** – виводить текст на дисплей, починаючи з поточного положення курсора.
  - **`home()`** і **`clear()`** – перша функція дозволяє повернути курсор в початок екрану, друга теж, але при цьому видаляє все, що було на моніторі до цього.

Завдання

1. Реалізувати керування світлодіодом (червоним) за допомогою вводу команд у серійний монітор («on» - для ввімкнення та «off» - для вимкнення).
2. Підключити до мікроконтролера три чіпи *NLFS595 Serial Tri-Color LED Driver* та RGB-світлодіоди. До кожного чіпу підключити лінію CS (Chip Select) та реалізувати можливість увімкнення кожного з трьох світлодіодів за допомогою команд у серійному моніторі (наприклад: «RGB1» – світиться лише перший RGB світлодіод, «RGB2» – другий, «RGB3» – третій).
3. Додати до ліній I2C другий LCD-дисплей 16x2, для відображення температури з MPU6050.

Додаткове завдання. Реалізувати завдання без використання функцій Arduino.

Порядок виконання роботи

Крок 1. Зібрати схему зображену на рисунку 3.7 або замінити вміст **`diagram.json`** відповідним текстом:

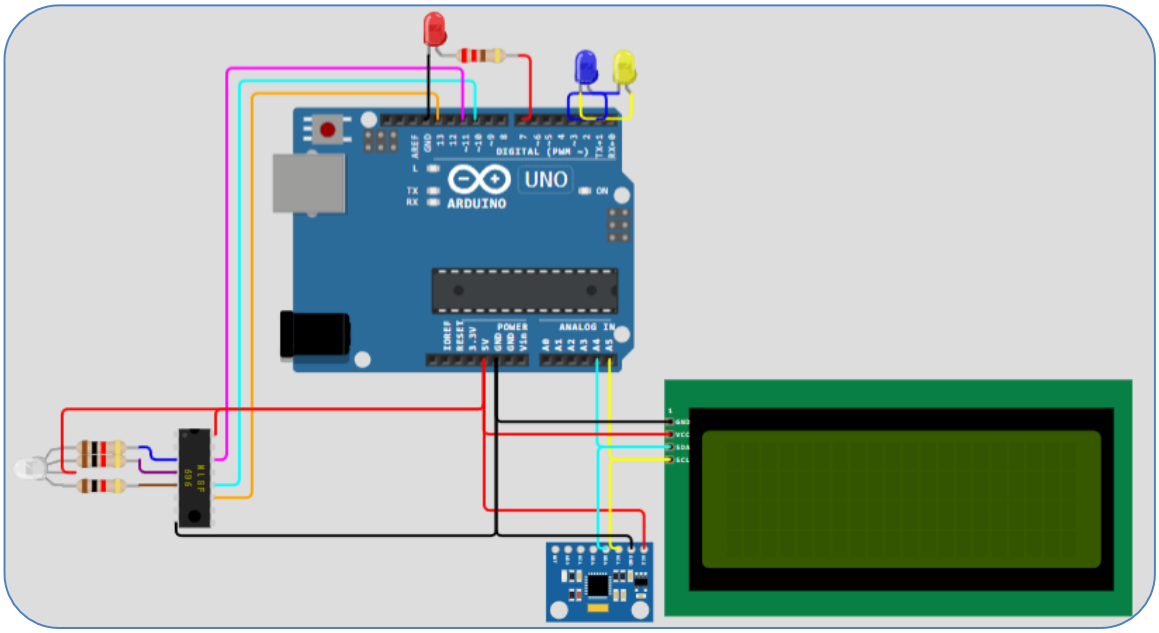


Рис. 3.7. Схема

```
{"version": 1, "author": "", "editor": "wokwi", "parts": [{"type": "wokwi-arduino-uno", "id": "uno",
"top": 183.33, "left": -51.33, "attrs": {}}, {"type": "wokwi-nlsf595", "id": "sr1", "top": 449.34,
"left": -148.3, "rotate": 90, "attrs": {}}, {"type": "wokwi-rgb-led", "id": "rgb1", "top": 421.4,
"left": -239.7, "rotate": 270, "attrs": {}}, {"type": "wokwi-resistor", "id": "r1", "top": 464.75,
"left": -211.2, "attrs": {"value": "1000"}}, {"type": "wokwi-resistor", "id": "r2", "top": 445.55,
"left": -211.2, "attrs": {"value": "1000"}}, {"type": "wokwi-resistor", "id": "r3", "top": 435.95,
"left": -211.2, "attrs": {"value": "1000"}}, {"type": "wokwi-mpu6050", "id": "imu1", "top": 512.62,
"left": 155.92, "attrs": {}}, {"type": "wokwi-led", "id": "led1", "top": 130.8, "left": 195.8, "attrs":
{"color": "yellow"}}, {"type": "wokwi-led", "id": "led2", "top": 130.8, "left": 167, "attrs":
{"color": "blue"}}, {"type": "wokwi-led", "id": "led3", "top": 102, "left": 51.8, "attrs": {"color":
"red"}}, {"type": "wokwi-lcd2004", "id": "lcd2", "top": 390.4, "left": 245.6, "attrs": {"pins": "i2c",
"i2cAddress": "0x48"}}, {"type": "wokwi-resistor", "id": "r4", "top": 138.35, "left": 76.8, "attrs":
```

```
{"value": "220"}}], "connections": [["r3:1", "rgb1:B", "blue", ["h0"]], ["r2:1", "rgb1:G", "green",
["h0"]], ["rgb1:R", "r1:1", "brown", ["v0"]], ["uno:5V", "rgb1:COM", "red", ["v37.97", "h-319.87",
"v48.1"]], ["r3:2", "sr1:QD", "blue", ["h8.4", "v9.6"]], ["r2:2", "sr1:QE", "purple", ["v0"]],
["sr1:QF", "r1:2", "#8f4814", ["h0"]], ["sr1:SCK", "uno:13", "orange", ["h27.6", "v-307.2",
"h140.87"]], ["sr1:RCK", "uno:10", "cyan", ["h18", "v-307.2", "h178.97"]], ["uno:11", "sr1:SI",
"magenta", ["v-38.73", "h-179.07", "v297.6"]], ["sr1:GND", "uno:GND.2", "black", ["v9.6", "h242.97"]],
["imu1:VCC", "uno:5V", "red", ["v-28.8", "h-121.23", "v-210.77"]], ["sr1:VCC", "uno:5V", "red",
["v-19.2", "h203.47"]], ["led1:C", "uno:1", "blue", ["h-38", "v48.33"]], ["led1:A", "uno:0", "yellow",
["v19.2", "h-35.83"]], ["led2:C", "uno:0", "yellow", ["h0.4", "v19.2", "h21.77"]], ["led2:A", "uno:1",
"blue", ["h9.6", "v48.33"]], ["led3:C", "uno:GND.1", "black", ["v48", "h-2.63"]], ["uno:A5",
"lcd2:SCL", "yellow", ["v0"]], ["uno:A4", "lcd2:SDA", "cyan", ["v0"]], ["lcd2:GND", "uno:GND.2",
"black", ["h0"]], ["lcd2:VCC", "uno:5V", "red", ["h0"]], ["lcd2:SCL", "imu1:SCL", "yellow",
["h-45.04", "v125.1"]], ["lcd2:SDA", "imu1:SDA", "cyan", ["h-54.29", "v134.6"]], ["uno:GND.2",
"imu1:GND", "black", ["v133.97", "h102.55"]], ["uno:7", "r4:2", "red", ["h6.33", "v-57.93"]],
["led3:A", "r4:1", "red", ["v0"]]], "dependencies": {}}
```


Крок 2. Завантаження шаблону коду з файлу Example3.cpp [4] у редактор коду Wokwi.

Крок 3. Перевірка працездатності програми та дослідження реалізації прикладів функцій для взаємодії по UART, SPI та I2C.

Крок 4. Реалізація керування червоним світлодіодом за допомогою команд від UART (завдання №1).

- Налаштуйте пін до якого підключено світлодіод.
- Модифікуйте функцію **`UART_example`**, а саме: після перевірки наявності вхідних даних, зчитайте повний вміст введеної команди.
- Реалізуйте логіку для ввімкнення та вимкнення світлодіода на основі отриманої команди «**`on`**» чи «**`off`**».

Крок 5. Підключення, через SPI, додаткових чіпів NLFS595, для керування трьома RGB-світлодіодами (завдання №2).

- Підключіть два додаткових чіпи NLFS595 до Arduino через інтерфейс SPI (MOSI, SCK, CS). Лінію CS кожного чіпу підключіть до окремого (вільного) цифрового піну на платі.
- До кожного з чіпів підключіть по одному RGB-світлодіоду.
- Модифікуйте функцію **`SPI_example`**, для реалізації обробки команд («RGB1», «RGB2», «RGB3») отриманих по UART.
- В залежності від отриманої команди налаштуйте активацію обраного RGB-світлодіода та деактивацію решти RGB-світлодіодів.

Крок 6. Додавання другого LCD-дисплея 16x2 до шини I2C, для відображення температури з MPU6050.

- Підключіть другий LCD-дисплей до ліній SDA і SCL.
- Переконайтесь, що адреси дисплеїв відрізняються. Для цього в файлі diagram.json виконайте пошук за словом «i2cAddress» та, за потреби, змініть вказані в лапках значення (для цього модулю адреса за замовчуванням «0x27»)
- Створіть новий об’єкт для роботи з другим дисплеєм.
- Модифікуйте функцію **`I2C_example`**, для зчитування з MPU6050 значень температури та відображення їх на дисплеї.

Оформлення звіту та порядок його подання

1. Увійдіть в дистанційний курс Moodle з власного аккаунту.
2. Відкрийте активність присвячену даній лабораторній роботі.
3. Натисніть кнопку «Здати роботу».
4. У текстове поле завантажте:
   - посилання на створений проект в симуляторі Wokwi з виконаними завданнями;
   - висновки до лабораторної роботи.
5. Натисніть кнопку «Зберегти».

Контрольні запитання

1. Що таке UART і які особливості його застосування?
2. Як відбувається взаємодія по SPI? Чи можливо налаштувати комунікацію з декількома мікросхемами?
3. В чому його основні відмінності I2C від UART та SPI?
4. Які переваги застосування кожного з розглянутих протоколів? Який з них доцільніше обрати, наприклад, при необхідності підключення великої кількості переферійних пристроїв і чому?

ЛАБОРАТОРНА РОБОТА № 4

Тема: Підключення вбудованої системи до мережі Інтернет та інтеграція мережевих функцій

Мета: Навчитися підключатись до мережі Інтернет та розвинути навички інтеграції мережевих функцій у вбудовані системи.

План проведення лабораторної роботи

- Ознайомлення з короткими теоретичними відомостями.
- Виконання завдання згідно порядку виконання роботи.
- Подання звіту в Moodle (кнопка «Здати роботу»).

Короткі теоретичні відомості

Wi-Fi

Wi-Fi (Wireless Fidelity) – це бездротова технологія, яка дозволяє пристроям підключатися до Інтернету або створювати бездротові локальні мережі. Wi-Fi є однією з основних технологій для підключення IoT-пристроїв до мережі Інтернет, що дозволяє їм взаємодіяти між собою і з іншими системами, забезпечуючи віддалений моніторинг і керування.

Wi-Fi має кілька ключових особливостей:

- **Бездротове підключення**.
- **Широкий діапазон**. Wi-Fi працює на частотах 2,4 ГГц та 5 ГГц, забезпечуючи різний діапазон покриття та швидкість передачі даних.
- **Швидкість**. Стандарти 802.11n, 802.11ac, 802.11ax та інші забезпечують різну швидкість передачі даних, від десятків до тисяч мегабіт за секунду.
- **Захист даних**. Рівні безпеки, включаючи WPA2 і WPA3, забезпечують шифрування даних і захист від несанкціонованого доступу.
- **Сумісність.** Wi-Fi підтримується більшістю сучасних пристроїв, таких як смартфони, ноутбуки, розумні домашні пристрої, IoT-гаджети.
- **Роумінг**. Підтримується плавний перехід між різними точками доступу в межах однієї мережі, що дозволяє безперервно користуватися підключенням на великих просторах.

Для підключення вбудованої системи до Wi-Fi потрібні наступні компоненти та кроки:

- Wi-Fi модуль або мікроконтролер з вбудованим Wi-Fi. Якщо мікроконтролер не має вбудованого Wi-Fi, знадобиться зовнішній модуль, наприклад ESP8266 або ESP32. Підключити який можливо за допомогою UART або SPI.
- Використовуючи бібліотеку, наприклад, WiFi.h для ESP8266/ESP32, в коді необхідно задати облікові дані для підключення до точки доступу Wi-Fi, а саме –вказати SSID (назву мережі) і пароль до неї.
- Програмування мікроконтролера. Програма повинна ініціалізувати Wi-Fi підключення, перевіряти наявність підключення і виконувати всі необхідні дії для роботи в мережі.

Приклад:

Використання бібліотеки WiFi.h [16] для підключення мікроконтролера ESP32 до Wi-Fi мережі включає наступні кроки:

1) Спочатку потрібно завантажити бібліотеку WiFi.h та підключити її:

```
#include <WiFi.h>
```

2) Далі потрібно визначити облікові дані Wi-Fi мережі, до якої треба підключитися (для симулятора Wokwi мережа має назву Wokwi-GUEST, а пароль – відсутній):

```
const char* ssid = "Wokwi-GUEST";   //Назва мережі
const char* password = "";          //Пароль
```

3) Перед підключенням до мережі можна задати режим роботи Wi-Fi:
   - режим роботи в якості станції (пристрій підключається до точки доступу Wi-Fi):

     ```
     WiFi.mode(WIFI_STA);
     ```

   - режим точки доступу (коли пристрій стає точкою доступу Wi-Fi):

     ```
     WiFi.mode(WIFI_AP);
     ```

   - змішаний режим (коли пристрій може одночасно підключатися до іншої точки доступу Wi-Fi та бути точкою доступу).

     ```
     WiFi.mode(WIFI_AP_STA);
     ```

4) Почати процес підключення мережі:

```
WiFi.begin(ssid, password);
```

5) Зачекати поки підключення буде встановлено.
6) Перевірку встановлення підключення до Wi-Fi можливо реалізувати за допомогою методу **`WiFi.status()`** [17]. Він повертає поточний статус Wi-Fi з'єднання, наприклад:
   - **`WL_CONNECTED`** – пристрій успішно підключений до Wi-Fi мережі.
   - **`WL_IDLE_STATUS`** – пристрій знаходиться в стані очікування (поки не намагався підключитися).
   - **`WL_NO_SSID_AVAIL`** – SSID (назва мережі) не знайдено.
   - **`WL_CONNECT_FAILED`** – помилка при спробі підключення.
   - **`WL_CONNECTION_LOST`** – з'єднання з мережею було втрачено.
   - **`WL_WRONG_PASSWORD`** – введений неправильний пароль.
   - **`WL_DISCONNECTED`** – пристрій відключено від мережі.

   Зазвичай, перевірка здійснюється в циклі, поки статус не зміниться на **`WL_CONNECTED`**:

```
while (WiFi.status() != WL_CONNECTED)
{
  delay(1000);         //Затримка 1 секунду
  Serial.print(".");   //Вивід "." для відображення процесу підключення
}
```

7) Написання програми для роботи з мережею. Для отримання MAC- та IP-адреси пристрою можливо застосувати методи **`WiFi.macAddress()`** та **`WiFi.localIP()`** відповідно:

```
Serial.print("IP Address: ");
Serial.println(WiFi.localIP());

Serial.print("MAC Address: ");
Serial.println(WiFi.macAddress());
```

Примітка. Детальніше про можливості, що надає бібліотека WiFi.h можна дізнатись за посиланням [16].

Отримання точної дати та часу з мережі Інтернет

NTP (Network Time Protocol) – це протокол для синхронізації годинника системи через мережу з високою точністю. Він дозволяє пристроям отримувати точний час з серверів NTP, які розповсюджують час, синхронізований з атомними годинниками або іншими надійними джерелами. Точність NTP зазвичай є вищою, ніж у мікросхем RTC (Real-Time Clock), де похибка може складати 20-30 секунд на добу.

Застосування протоколу NTP доцільне в проєктах, де важлива точна синхронізація часу, наприклад:

- Розподілені системи, де різні пристрої повинні працювати синхронно.
- Системи моніторингу, безпеки або збору даних, де важливо мати точний час для запису подій.
- Проекти IoT, в яких дані передаються на сервери або в хмару.
- Розумні будинки та інші системи автоматизації, де точний час може бути важливим для керування освітлення, опаленням і т.д.
- Фінансові та торгові системи, де транзакції повинні бути точно синхронізовані.
- Системи відтворення потокового відео та аудіо.

NTPClient [18] – бібліотека, яка спрощує отримання часу з NTP-серверів через протокол UDP. Вона дозволяє легко синхронізувати годинник мікроконтролера з інтернет-часом.

Основні методи бібліотеки NTPClient:

- **`begin()`** – ініціалізує NTPClient і запускає процес синхронізації часу.
- **`update()`** – оновлює час, отриманий від NTP сервера.
- **`getFormattedTime()`** – повертає відформатований час у форматі hh:mm:ss.
- **`getEpochTime()`** – повертає час у Unix-форматі (кількість секунд з 1 січня 1970 року).
- **`setTimeOffset(`*`offset`*`)`** – встановлює зсув часу для локального часового поясу в секундах.
- **`setUpdateInterval(`*`interval`*`)`** – встановлює інтервал між оновленнями часу від NTP сервера.
- **`getHours(), getMinutes(), getSeconds(), getDay()`** – ці методи повертають відповідно години, хвилини, секунди та день поточного часу.

Приклад:

Використання бібліотеки NTPClient [18] для отримання мережевого часу може включати наступні кроки:

1) Встановлення та підключення бібліотек **`NTPClient.h`** (для роботи з NTP) та **`WiFiUdp.h`** (для комунікації по UDP):

```
#include <NTPClient.h>
#include <WiFiUdp.h>
```

2) Підключення до мережі.
3) Створення об'єктів WiFiUDP та NTPClient:

```
WiFiUDP udp;
NTPClient ntpClient(udp, "pool.ntp.org", 3600);
```

4) Початок роботи з сервером NTP за допомогою методу **`begin()`**:

```
void setup()
{
    //... підключення до мережі

    ntpClient.begin();
}
```

5) Оновлення часу та виведення відформатованого часу в серійний монітор:

```
ntpClient.update();
Serial.print("Поточний час: ");
Serial.println(timeClient.getFormattedTime());
```

Завдання

1. Підключити до мікроконтролера дисплей та реалізувати відображення поточного статусу підключення до мережі (Offline/Online).
2. Після успішного підключення до Wi-Fi, крім статусу, відобразити на дисплеї присвоєну пристрою IP-адресу.
3. За допомогою NTP-сервера визначити поточний київський час (враховуючи зсув часового поясу) та відобразити його на дисплеї (після статусу та IP-адреси).
4. Кожні 60 секунд виконувати синхронізацію з NTP-сервером, а в інтервалах між оновленнями, реалізувати лічильник для відображення кількості секунд в реальному часі.

Додаткове завдання 1. Додайте на схему кнопку для відключення/підключення мережі Wi-Fi.

Додаткове завдання 2. За допомогою кнопок реалізувати можливість переведення годинника на одну годину вперед та назад (зміни часового поясу).

Додаткове завдання 3. Реалізувати завдання без використання бібліотек та функцій Arduino ☺.

Порядок виконання роботи

Крок 1. Підключення до ESP32 дисплея 20x4 по протоколу I2C. Піни для підключення ліній SCL та SDA зображені на рисунку 4.1.

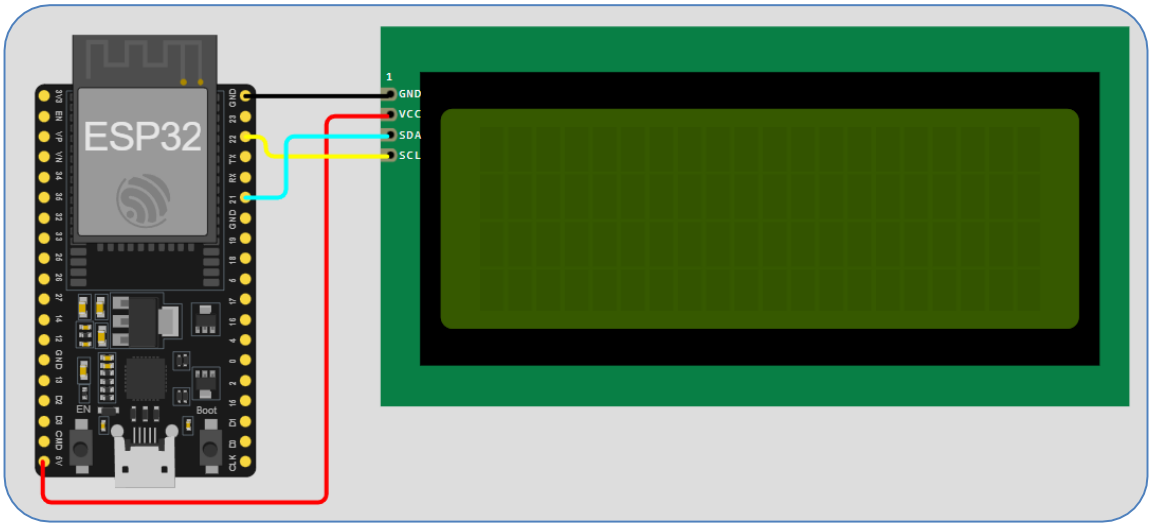


Рис. 4.1. Підключення дисплея по I2C

Крок 2. Ініціалізація дисплея у коді.

Крок 3. Налаштування підключення до Wi-Fi, використовуючи бібліотеку WiFi.

Крок 4. Перевірка статусу підключення (Online/Offline) до мережі та відображення його на дисплеї.

Крок 5. Отримання IP-адреси та відображення її на дисплеї.

Крок 6. Визначення та відображення часу:

- Налаштуйте NTP-клієнт.
- За допомогою NTP-клієнта отримайте поточний час (з урахуванням зсуву для Києва).

Крок 7. Синхронізація та відображення реального часу:

- Реалізуйте механізм для синхронізації з NTP-сервером кожні 60 секунд.
- Реалізуйте лічильник, який після синхронізації буде продовжувати підрахунок часу (без додаткових запитів до NTP) та відображення поточної кількості секунд на дисплеї.

Оформлення звіту та порядок його подання

- Увійдіть в дистанційний курс Moodle з власного аккаунту.
- Відкрийте активність присвячену даній лабораторній роботі.
- Натисніть кнопку «Здати роботу».
- У текстове поле завантажте:
  - посилання на створений проект в симуляторі Wokwi з виконаними завданнями;
  - висновки до лабораторної роботи.
- Натисніть кнопку «Зберегти».

Контрольні запитання

1. Які основні кроки потрібно виконати для підключення вбудованої системи до мережі?
2. Які протоколи використовуються для отримання точного часу з Інтернет сервера?
3. Як можна перевірити поточний статус підключення до мережі Wi-Fi?
4. Які основні переваги використання NTP для синхронізації часу у вбудованих системах?

ЛАБОРАТОРНА РОБОТА № 5

Тема: Протоколи прикладного рівня в системах Інтернету речей

Мета: Навчитись використовувати протоколи прикладного рівня (HTTP та MQTT), а також серіалізувати/десеріалізувати дані, для забезпечення взаємодії з сервісами та пристроями в мережі Інтернет.

План проведення лабораторної роботи

- Ознайомлення з короткими теоретичними відомостями.
- Виконання завдання згідно порядку виконання роботи.
- Подання звіту в Moodle (кнопка «Здати роботу»).

Короткі теоретичні відомості

HTTP

Hypertext Transfer Protocol (HTTP) – це протокол прикладного рівня мережевої моделі, який використовується для передачі гіпертексту (наприклад, HTML-документів) у всесвітній павутині (World Wide Web). HTTP забезпечує взаємодію між веб-браузерами або іншими клієнтами (наприклад, мобільними додатками, програмами IoT-пристроїв) та веб-серверами. Це основний протокол, на якому базується робота Інтернету.

Основні принципи роботи HTTP:

- Архітектура «клієнт-сервер». HTTP працює за принципом "запит-відповідь". Сервер зберігає ресурси (HTML-сторінки, зображення, відео тощо). Клієнт (наприклад, браузер чи будь-яка інша програма) ініціює запит до сервера, сервер обробляє запит і надсилає відповідь клієнту.
- HTTP є безстановим протоколом, тобто кожен запит обробляється незалежно від попередніх запитів. Сервер не зберігає інформацію про попередні запити одного клієнта. Для збереження стану (наприклад, під час роботи з сесіями) використовуються механізми, такі як cookies, сесії та токени.
- HTTP-заголовки містять метадані про запит або відповідь. Вони можуть включати інформацію про тип вмісту (Content-Type), код відповіді (Status Code), налаштування кешування, аутентифікацію, та багато іншого.
- Методи HTTP – це команди, які клієнт використовує для взаємодії з сервером:
  - GET – запит на отримання даних з сервера.
  - POST – використовується для надсилання даних на сервер.
  - PUT – зазвичай використовується для оновлення наявного ресурсу на сервері.
  - DELETE – видалення ресурсу.
  - HEAD – запит схожий на GET, але отримує тільки заголовки відповіді, без тіла.
  - PATCH – використовується для часткового оновлення ресурсу.

- HTTP відповіді містять коди статусу, які вказують на результат запиту, наприклад:
  - 200 OK – запит успішно виконано.
  - 301 Moved Permanently – ресурс переміщений на нову адресу.
  - 404 Not Found – ресурс не знайдено.
  - 500 Internal Server Error – внутрішня помилка сервера.

Існує кілька версій HTTP, кожна з яких вносить удосконалення для підвищення ефективності, безпеки та зручності роботи з мережею. Більшість веб-сайтів досі працюють лише з HTTP/1.1.

HTTPS (HTTP Secure) – це розширення HTTP, яке додає шифрування за допомогою протоколу TLS (Transport Layer Security). HTTPS забезпечує захист даних під час передачі між клієнтом і сервером, що особливо важливо для конфіденційної інформації.

У системі для вимірювання температури та вологості HTTP-методи можуть бути використані для надсилання даних від давача на сервер наступним чином:

- Кожен раз, коли система вимірює температуру та вологість вона направляє запит до сервера за допомогою метода POST, наприклад такий:

```
POST /temperature.php HTTP/1.1
Host: example.com
Content-Type: application/json; charset=utf-8
Content-Length: 36

{
  "temperature": 25.5,
  "humidity": 50.2
}
```

Він включає:

  - Перший рядок «**`POST /temperature.php HTTP/1.1`**», де:
    - ✓ **`POST`** – метод, який вказує, що дані будуть передані на сервер для обробки.
    - ✓ **`/temperature.php`** – це шлях до ресурсу на сервері, який буде обробляти отримані дані.
    - ✓ **`HTTP/1.1`** – використовувана версія протоколу.
  - Заголовки запиту:
    - ✓ **`Host: example.com`** – вказує доменне ім'я сервера, до якого відправляється запит.
    - ✓ **`Content-Type: application/json; charset=utf-8`** – вказує, що передані дані мають формат JSON і використовують кодування символів UTF-8.
    - ✓ **`Content-Length: 36`** – вказує розмір тіла запиту в байтах (36 байт).
  - Тіло запиту – це корисне навантаження запиту, текст в форматі JSON, який містить два поля: «**`temperature`**» та «**`humidity`**».

Для отримання останніх виміряних даних з сервера можна звернутись до нього за допомогою метода GET, наприклад:

```
GET /temperature.php HTTP/1.1
Host: example.com
```

Він включає:

- Перший рядок, в якому вказано, що саме (**GET**) необхідно зробити з певним ресурсом на сервері (**temperature.php**).
- Заголовок запиту, де вказується доменне ім’я сервера до якого направляється запит.

Відповідь сервера залежить від його налаштувать і її мінімалістичний варіант може виглядати наступним чином:

```
HTTP/1.1 200 OK
Content-Type: text/plain; charset=utf-8

Hello! Current temperature = 25, humidity = 50!
```

Відповідь включає:

- Перший рядок «**HTTP/1.1 200 OK**»:
  - ✓ **HTTP/1.1** – версія протоколу, яка використовується для передачі відповіді.
  - ✓ **200 OK** – це код статусу, який означає, що запит був успішно оброблений сервером.
- Заголовок відповіді «**Content-Type: text/plain; charset=utf-8**», який вказує на тип контенту у відповіді (звичайним текст у кодуванні UTF-8).
- Тіло відповіді – текст, що інформує клієнта про поточну температуру та вологість.

Вцілому, HTTP є широко розповсюдженим і стандартизованим протоколом, який просто реалізувати та почати використовувати. Недоліками HTTP є його «важкість», через використання текстових заголовків і безперервних TCP-з'єднань, що може бути надмірним для енергообмежених і низькобітних IoT-пристроїв. Також, HTTP не оптимізований для малих пакетів даних, що характерно для IoT-систем, де часто передаються невеликі обсяги інформації. У випадках, коли IoT-пристрої розгортаються в умовах з ненадійним зв’язком (наприклад, у віддалених регіонах або з використанням мобільних мереж), HTTP може не забезпечити необхідної стійкості і надійності передачі даних. Інші протоколи, такі як MQTT, мають вбудовані механізми для відновлення з'єднання і повторної передачі даних у разі втрат, що робить їх більш придатними для таких умов.

MQTT

MQTT (Message Queuing Telemetry Transport) — це легковаговий протокол обміну повідомленнями, розроблений для пристроїв з обмеженими ресурсами і в ненадійних мережах. MQTT використовує модель «publish-subscribe» (pub-sub), яка дозволяє ефективно передавати дані між пристроями, мінімізуючи навантаження на мережу.

До переваг MQTT над HTTP, особливо в контексті IoT-систем, можливо віднести:

- Низьке енергоспоживання.
- Невеликий заголовок.
- Підтримка активного з'єднання.
- Підтримка асинхронного зв'язку.
- Механізм гарантування доставки повідомлень.

MQTT підтримує **три рівні якості обслуговування (QoS)**. Клієнт, при відправці повідомлення, вибирає рівень QoS залежно від того, який рівень надійності доставки він бажає забезпечити:

- **QoS 0** (максимум один раз) – повідомлення доставляється один раз, але без підтвердження та повторної передачі. TCP гарантує доставку до конкретної машини, але не до застосунку (MQTT-клієнта чи брокера).
- **QoS 1** (принаймні, один раз) – гарантується, що повідомлення буде доставлено принаймні один раз, але можлива подвійна доставка, тобто може бути отримано декілька копій повідомлення.
- **QoS 2** (точно один раз) – гарантується доставка повідомлення лише один раз. Точність доставки може вимагати більше ресурсів мережі.

Модель pub-sub (publish-subscribe) – це архітектурний патерн, що використовується для організації обміну повідомленнями між різними компонентами системи. Вона дозволяє клієнтам обмінюватися повідомленнями через посередника, не знаючи про існування один одного. Ця модель є асинхронною, що робить її зручною для систем з великою кількістю пристроїв.

Основні компоненти моделі pub-sub:

- **Publisher** (Видавець) – відправляє (публікує) повідомлення у певну тему (канал повідомлень – topic).
- **Subscriber** (Підписник) – підписується на одну або більше тем, щоб отримувати повідомлення, які там публікуються.
- **Broker** (Посередник) – отримує повідомлення від видавця та пересилає їх усім підписникам, які підписалися на відповідну тему.

Переваги моделі pub-sub:

- Видавці й підписники не потребують інформації один про одного.
- Модель добре підходить для великих систем з великою кількістю учасників.
- Підписники можуть підписуватись на різні теми й отримувати тільки ті повідомлення, які їм потрібні.

Особливості тем (топіків):

- Теми можуть мати ієрархічну структуру, подібно до файлової системи. Наприклад, «`sensors/temperature/living_room`» або «`sensors/humidity/outside`».
- Підписники можуть підписуватись на конкретні теми або використовувати шаблони для отримання повідомлень з кількох тем одночасно. Наприклад, підписка на «`sensors/temperature/#`» дозволить отримувати всі повідомлення з тем, що починаються на «`sensors/temperature/`».
- Захист тем відбувається через налаштування політик доступу на MQTT-брокері. Вони визначають, які користувачі чи пристрої мають право публікувати або підписуватись на певні теми.

Серіалізація даних

Серіалізація даних – це процес перетворення структурованих даних (наприклад, об'єктів у програмі) у формат, який може бути легко збережений або переданий, а потім десеріалізований (відновлений) у вихідну структуру. Одним із таких форматів є JSON.

JSON (JavaScript Object Notation [19]) – це текстовий формат, який використовується для зберігання і передачі структурованих даних. JSON легко читається людьми та обробляється машинами. Дані представляються у вигляді об’єктів з ключами та значеннями, а також масивів.

Наприклад, в програмі на C++ є декілька змінних, значення яких необхідно передати мережею:

```
int deviceId = 10;
double temperature = 25.5;
double humidity = 50;
String city = "Kyiv";
double coordinates[] = {50.450001, 30.523333};
```

Наведену інформацію можливо серіалізувати у звичайний рядок тексту:

```
string data = "1025.550Kyiv50.45000130.523333";
```

З таким текстом може бути складно працювати тому, використавши формат JSON, текст буде мати наступний вигляд:

```
{
  "deviceId": 10,
  "temperature": 25.5,
  "humidity": 50,
  "city": "Kyiv",
  "coordinates": [50.450001, 30.523333]
}
```

Десеріалізація

Десеріалізація – це зворотний серіалізації процес. Дані з формату для передачі чи збереження перетворюються у програмний об'єкт.

Бібліотека ArduinoJson [20] дозволяє легко десеріалізовувати дані у форматі JSON та зручно отримувати доступ до значень окремих полів. Основні кроки включають:

- Підключення бібліотеки до проекту:

```
#include <ArduinoJson.h>
```

- Підготовка тексту у форматі JSON, який необхідно десеріалізувати:

```
String jsonText = "{\"deviceId\":44, \"temperature\":25.5, \"humidity\":50,
\"city\":\"Kyiv\", \"coordinates\":[50.450001, 30.523333]}";
```

- Створення об'єкта **doc** для зберігання десеріалізованих даних:

```
JsonDocument doc;
```

- Десеріалізація (JSON-рядок **jsonText** перетворюється на об'єкт **doc**):

```
deserializeJson(doc, jsonText);
```

- Отримання значень:

```
int deviceId = doc["deviceId"];
double temperature = doc["temperature"];
double humidity = doc["humidity"];
const char* city = doc["city"];
double lat = doc["coordinates"][0];
double lon = doc["coordinates"][1];
```

Крім цього працювати з текстовими даними можливо за допомогою вбудованих методів класу **`String`**, що дозволяє виконувати різні операції, такі як пошук підрядків, виділення частин рядка, слів, чисел і так далі:

- **`substring(`*`beginIndex`*`, `*`endIndex`*`)`** [21] – використовується для виділення частини рядка. Він має два параметри: початковий індекс і кінцевий індекс. Якщо кінцевий індекс не вказаний, метод повертає підрядок від початкового індексу до кінця рядка. Наприклад:

```
String data = "Temperature: 25.5C";
String temperature = data.substring(13, 17);  // Виділяє підрядок "25.5"
```

- **`indexOf(`*`ch`*`)`** – використовується для пошуку індексу певного символу або підрядка в рядку, наприклад:

```
String data = "Temperature: 25.5C";
int colonIndex = data.indexOf(':');  // Знайде індекс символу ':'
```

Завдання

1. Здійснити HTTPS запит до «https://ua.sinoptik.ua/погода-київ». Вивести відповідь у серійний монітор та ознайомитись з нею.
2. Здійснити HTTPS запит (з використанням безкоштовних open-source API) до «**`https://api.open-meteo.com/v1/forecast?latitude=50.4547&longitude=30.5238&hourly=temperature_2m&timezone=auto&forecast_days=1`**». Вивести відповідь у серійний монітор та ознайомитись з нею.
   Примітка. Більш детально сформувати запит можливо онлайн за посиланням [22].
3. З отриманої відповіді (прогнозу погоди) виділити значення температури, у вказаний за варіантом час (табл. 5.1), та відобразити його на дисплеї I2C 20x4 у форматі «T(18:00) = 3.2 *C» *[Можна замінити на додаткове завдання 1]*.
4. Щосекунди знімати показник температури з давача DHT22 та відправляти його за допомогою протоколу MQTT у топік «GroupRE/Temp/**N**», де **N** – номер варіанту згідно (використовуючи публічний брокер [23]).
5. Підписатися на топіки «GroupRE/Temp/**N**» та «GroupRE/#».
6. При отриманні повідомлення «**`on`**» в топік «GroupRE/Temp/**N**» вмикати синій світлодіод, а при «**`off`**» - вимикати.

Додаткове завдання 1. Обробити отриманий в першому завданні рядок (**`payload`**), виділивши з нього поточне значення температури. Відобразити його на дисплеї у вигляді напису, наприклад: «Sinoptik.ua: -1 °C».

Додаткове завдання 2. Підключити додатковий дисплей для відображення на ньому прогнозу температури на ніч (00:00), ранок (06:00), день (12:00) та вечір (18:00) (за даними отриманими в другому завданні).

Таблиця варіантів

Табл. 5.1. Варіанти завдання

Варіант	1	2	3	4	5	6	7	8	9	10
Час	00:00	01:00	02:00	03:00	04:00	05:00	06:00	07:00	08:00	09:00

Варіант	11	12	13	14	15	16	17	18	19	20
Час	10:00	11:00	12:00	13:00	14:00	15:00	16:00	17:00	18:00	19:00

Варіант	21	22	23	24	25	26	27	28	29	30
Час	20:00	21:00	22:00	23:00	00:00	01:00	02:00	03:00	04:00	05:00

Варіант	31	32	33	34	35	36	37	38	39	40
Час	06:00	07:00	08:00	09:00	10:00	11:00	12:00	13:00	14:00	15:00

Варіант	41	42	43	44	45	46	47	48	49	50
Час	16:00	17:00	18:00	19:00	20:00	21:00	22:00	23:00	00:00	01:00

Порядок виконання роботи

Крок 1. Під’єднання до ESP32 дисплея 20x4 (по протоколу I2C), давача DHT22 та світлодіода згідно схеми (рис. 5.1).

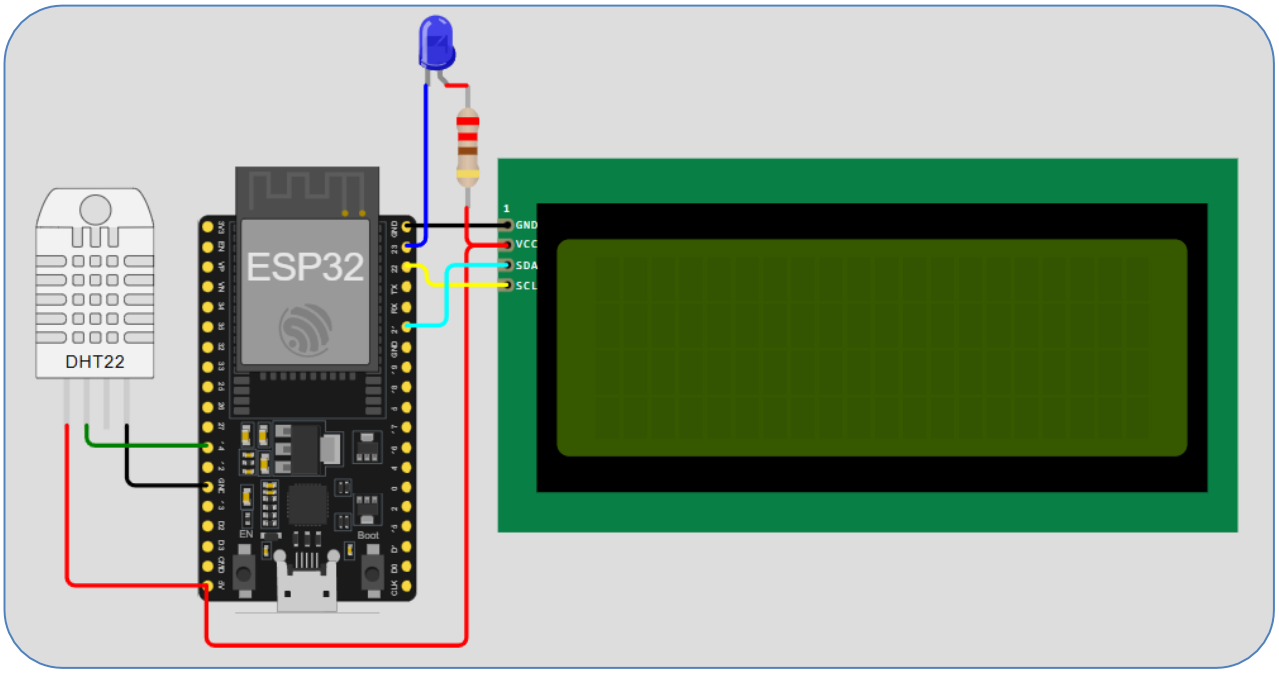


Рис. 5.1. Схема підключення модулів

Крок 2. Написання коду для підключення до Wi-Fi, виведення статусу підключення на дисплей та здійснення запиту до сервера в Інтернеті «https://ua.sinoptik.ua/погода-київ», для чого:

1) Підключити бібліотеки WiFi.h – для роботи в Wi-Fi мережі, HTTPClient – для роботи за протоколом HTTP, LiquidCrystal_I2C – для роботи з дисплеєм:

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <LiquidCrystal_I2C.h>
```

2) Вказати SSID та пароль Wi-Fi мережі:

```
const char* ssid = "Wokwi-GUEST";
const char* password = "";
```

3) Визначити URL для подальшого виконання GET-запиту:

```
const String url1 = "https://ua.sinoptik.ua/погода-київ";
```

4) Ініціалізувати дисплей:

```
LiquidCrystal_I2C lcd(0x27, 20, 4);
```

5) Встановити підключення до мережі Wi-Fi в функції setup:

```
void setup()
{
  // Ініціалізація дисплею
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0,0);

  // Ініціалізація порту для виведення даних у консоль
  Serial.begin(115200);
  // Підключення до Wi-Fi
  WiFi.begin(ssid, password);
  lcd.print("Wi-Fi: Connecting");
  while(WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    lcd.print(".");
  }
  lcd.clear();
  lcd.println("Wi-Fi: Online");
  loadPage1();
}
```

6) Перед виконанням запитів перевірити наявність підключення до Wi-Fi:

```
void loadPage1()
{
    //Перевірка наявності Wi-Fi підключення
    if(WiFi.status()== WL_CONNECTED)
    {
      //Код для роботи по HTTP, описаний в пунктах 7-10
    }
```

```
    else
    {
      Serial.println("З'єднання Wi-Fi втрачено");
    }
}
```

7) Створити об’єкт класу **`HTTPClient`**. В якості аргументу методу begin використовувати URL до якого буде виконуватись запит. За допомогою **`http.GET()`** виконати GET-запиту. В змінну типу **`int`** записати код відповіді:

```
HTTPClient http;
http.begin(url1);
int httpResponseCode = http.GET();
```

8) Якщо код відповіді присутній (>0) – вивести його, після чого викликати метод **`getString()`**, для повернення вмісту відповіді (**`payload`**):

```
if (httpResponseCode > 0)
{
  Serial.print("HTTP код відповіді: ");
  Serial.println(httpResponseCode);
  String payload = http.getString();
  Serial.println(payload);
}
```

9) Якщо код відповіді менше або дорівнює 0, вивести повідомлення про помилку:

```
else
{
  Serial.print("Error code: ");
  Serial.println(httpResponseCode);
}
```

10) Завершити роботу по HTTP:

```
http.end();
```

Крок 3. Після запиту до **`url1`**, здійснення аналогічних дій для **`url2`** (із завдання № 2).

Крок 4. Ознайомлення з можливостями бібліотеки ArduinoJson [20]. Виділення, з отриманих даних, температури у вказаний за варіантом час та відображення її на дисплеї у форматі «T(18:00) = 3.2 *C».

Крок 5. Ознайомлення з матеріалами присвяченими роботі з давачем DHT22, наприклад за посиланням [24]. Перевірити роботу давача вивівши температуру у серійний монітор. Приклад коду для роботи з DHT:

```
#include "DHT.h"

#define DHTPIN 14        // Визначаємо пін, до якого підключено датчик DHT
#define DHTTYPE DHT22    // Визначаємо тип датчика DHT (DHT22)

DHT dht(DHTPIN, DHTTYPE); // Створюємо об'єкт класу DHT з вказаним піном і типом
```

```
void setup()
{
  Serial.begin(115200); // Ініціалізуємо з'єднання з Serial монітором
  dht.begin();          // Ініціалізуємо датчик DHT
}

void loop()
{
  float temperature = dht.readTemperature(); // Отримуємо значення температури
                                             // з датчика DHT
  Serial.print("Temperature: "); // Виводимо повідомлення "Temperature: "
  Serial.print(temperature);     // Виводимо значення температури
  Serial.println(" °C");         // Виводимо символ градуса і одиницю виміру

  delay(1000);
}
```

Крок 6. Ознайомлення з функціями бібліотеки PubSubClient [25, 26] для комунікації по MQTT та налаштуваннями публічного MQTT брокера від HiveMQ [23] (рис. 5.2).

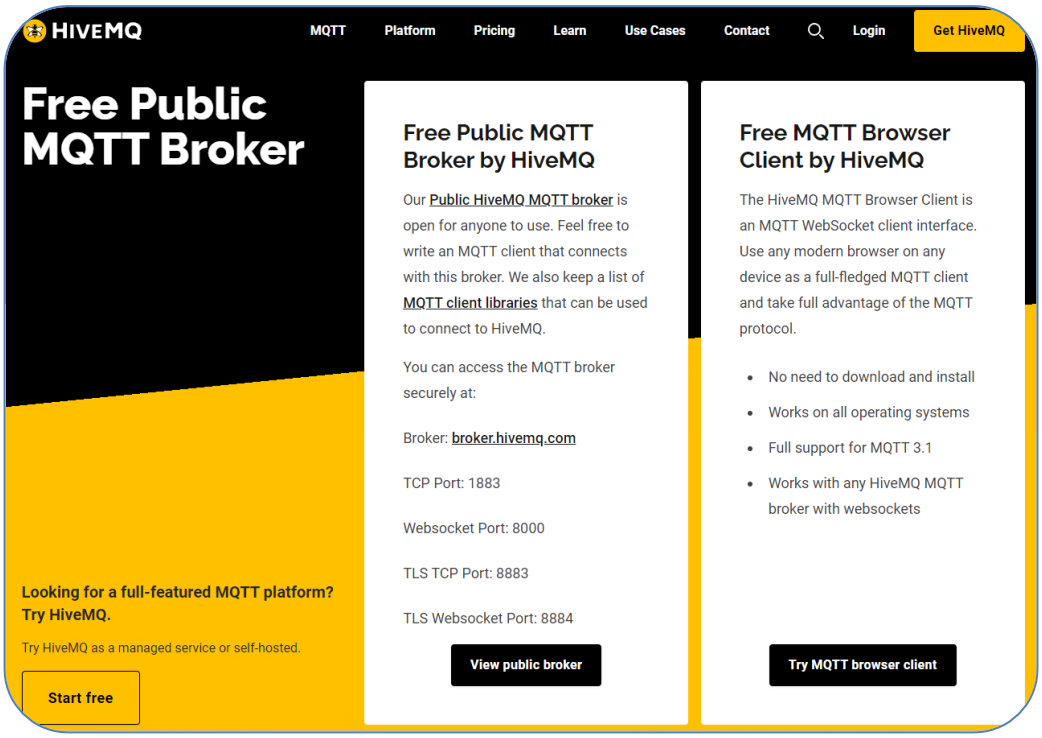


Рис. 5.2. Вигляд сторінки веб-сайту HiveMQ присвяченої MQTT брокеру

Налаштування виглядають наступним чином:

- Server: broker.hivemq.com
- TCP Port: 1883 (Websocket Port: 8000; TLS TCP Port: 8883; TLS Websocket Port: 8884)

Основні кроки, необхідні для налаштування плати-видавця (Publisher):

1) Починаємо з підключення, до проекту розробленому на попередніх кроках, бібліотеки **`PubSubClient.h`** для роботи з MQTT:

```
#include <PubSubClient.h>
```

2) Налаштування параметрів мережі та MQTT. Задаються параметри, такі як: адреса брокера, порт, топік та ідентифікатор клієнта (для публічного брокера логін та пароль не використовуються). **clientId** використовується для ідентифікації конкретного клієнта при підключенні до MQTT-брокера. Кожен клієнт повинен мати унікальний **clientId**, це дозволяє брокеру відрізняти одного клієнта від іншого та визначати на які топіки вони підписані:

```
const char* mqttServer = "broker.hivemq.com";
const char* myTopic = "GroupRE/Temp/N"; // N - замінити на ваш номер варіанту
const char* groupTopic = "GroupRE/#";
int port = 1883;
const char* clientId = "UID"; // Придумати власний Id
```

3) Ініціалізація об'єктів WiFi та MQTT. Створюються об'єкти WiFiClient та PubSubClient, які використовуються для взаємодії з Wi-Fi та MQTT відповідно:

```
WiFiClient wifiClient;
PubSubClient client(wifiClient);
```

4) У функції **setup** виконується підключення до Wi-Fi, а також, встановлюється адреса та порт MQTT-брокера для клієнта PubSubClient:

```
void setup()
{
  // Підключення до Wi-Fi та інший код

  client.setServer(mqttServer, port);
}
```

5) Основний цикл програми (**loop**). У ньому перевіряється, чи встановлено з'єднання з MQTT-брокером. Якщо ні (або воно зникло), викликається функція **reconnect**, що спробує підключитися знову. Якщо з’єднання є – у консоль виводиться інформація щодо топіку, який буде використовуватись для публікації повідомлення та відправляється значення температури з DHT22 у вказаний топік MQTT-брокера.

```
void loop()
{
  delay(1000);

  // Перевірка підключення по MQTT
  if (!client.connected())
  {
    reconnect();
  }

  // Відправка повідомлення в топік
  String temperature = String(dht.readTemperature());
  client.publish(myTopic, temperature.c_str());
}
```

6) Функція **reconnect** встановлює з'єднання з MQTT-брокером та підписується на вказаний раніше топік. Якщо з'єднання невдале, у консоль виводиться помилка та виконується очікування перед спробою повторного підключення:

```
void reconnect()
{
  while (!client.connected())
  {
    Serial.println("Підключення по MQTT...");
    if (client.connect(clientId))
    {
      client.subscribe(myTopic);
      client.subscribe(groupTopic);
      Serial.print("Клієнт: ");
      Serial.print(clientId);
      Serial.println(" підключений.");
    }
    else
    {
      Serial.print("помилка: ");
      Serial.print(client.state());
      Serial.println(" повторна спроба через 3 секунди");
      delay(3000);
    }
  }
}
```

Крок 7. Перевірка надходження повідомлень у топік за допомогою онлайн клієнта HiveMQ [27] (рис. 5.3).

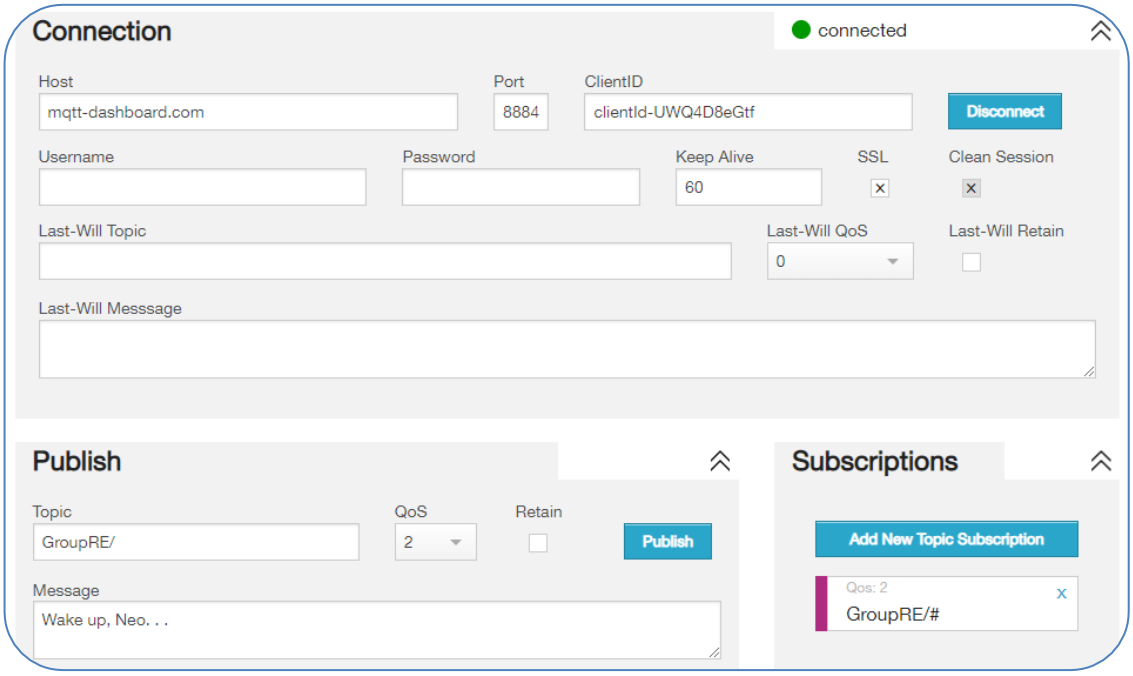


Рис 5.3. Вигляд сторінки онлайн клієнта HiveMQ

Крок 8. Налаштування отримання та обробки повідомлень від брокера. Основні кроки, необхідні для налаштування підписки:

1) До попереднього варіанту **`setup()`** необхідно додати виклик **`client.setCallback(callback)`**, що вказує клієнту PubSubClient використовувати функцію **`callback`** для обробки повідомлень:

```
client.setServer(mqttServer, port);
client.setCallback(callback);
```

2) В кінець основного циклу програми (**`loop`**) необхідно додати виклик методу **`client.loop()`**, для отримання повідомлень і виклику функції **`callback`**:

```
void loop()
{
  // . . . код з попередніх кроків
  // Отримання повідомлень
  client.loop();
}
```

3) Функція **`callback`** викликається при отриманні нового повідомлення від MQTT-брокера. Вона: зчитує байти повідомлення та конвертує їх в рядок (**`String`**), який потім виводиться у консоль та на дисплей; крім цього перевіряється, чи топік повідомлення не співпадає з вказаним топіком (**`myTopic`**), щоб не дублювати власні повідомлення в консоль:

```
void callback(char* topic, byte* message, unsigned int length)
{

  String stMessage;

  for (int i = 0; i < length; i++)
  {
    stMessage += (char)message[i];
  }

  //Вивід всіх повідомлень крім власних
  if (String(topic) != myTopic)
  {
    Serial.print("Отримано повідомлення з топіка ");
    Serial.print(topic);
    Serial.print(": ");
    Serial.println(stMessage);

    lcd.setCursor(0,2);
    lcd.print("Last msg:");
    lcd.setCursor(0,3);
    lcd.print(stMessage);
  }
}
```

Крок 9. Реалізувати ввімкнення та вимкнення світлодіоду після отримання повідомлення «on»/«off» з топіка за варіантом.

Оформлення звіту та порядок його подання

1. Увійдіть в дистанційний курс Moodle з власного аккаунту.
2. Відкрийте активність присвячену даній лабораторній роботі.
3. Натисніть кнопку «Здати роботу».
4. У текстове поле завантажте:
   - посилання на створений проект в симуляторі Wokwi з виконаними завданнями;
   - висновки до лабораторної роботи.
5. Натисніть кнопку «Зберегти».

Контрольні запитання

1. Як надсилати запити за протоколом HTTP? З чого вони складаються?
2. Що таке серіалізація даних?
3. Як обробляти відповіді від сервера, які засоби можливо застосовувати для цього? Що таке десеріалізація?
4. Чим відрізняється налаштування комунікації за протоколами HTTP та MQTT?
5. Які переваги використання моделі «видавець-підписник» в IoT-системах?

ЛАБОРАТОРНА РОБОТА № 6

Тема: Основи роботи з хмарною платформою IoT

Мета: Оволодіти основами налаштування хмарної платформи для Інтернету речей та навчитись ефективно взаємодіяти з нею, надсилати/збирати дані та обробляти команди.

План проведення лабораторної роботи

- Ознайомлення з короткими теоретичними відомостями.
- Виконання завдання згідно порядку виконання роботи.
- Подання звіту в Moodle (кнопка «Здати роботу»).

Короткі теоретичні відомості

Хмарні обчислення – це модель надання обчислювальних ресурсів, таких як сервери, сховища, бази даних, мережеві ресурси, програмне забезпечення та аналітичні засоби, через Інтернет ("хмару").

Постачальники хмарних послуг – це компанії, які надають доступ до хмарних обчислень на основі підписки або оплати за використання, що дозволяє компаніям і розробникам користуватися обчислювальними потужностями без необхідності створення і підтримки власної фізичної інфраструктури. Основними постачальниками хмарних послуг є Amazon Web Services, Microsoft, Google Cloud, IBM Cloud, Oracle Cloud.

Застосування хмарних обчислень для вбудованих систем IoT має кілька важливих переваг:

- **Легко масштабувати** обчислювальні потужності та сховища даних у відповідь на збільшення кількості підключених пристроїв або зростання обсягів даних.
- **Зниження витрат**, не потрібно інвестувати у власну інфраструктуру (сервери, сховища, мережі). Обслуговування та підтримка інфраструктури, резервне копіювання та оновлення здійснюються на стороні хмарного провайдера.
- **Доступність та надійність**, дані та обчислювальні потужності доступні з будь-якої точки світу, а також забезпечується резервне копіювання та відновлення після збоїв.
- **Обробка великих даних** з використанням сучасних інструментів для аналітики та машинного навчання.

Основні типи послуг, які надають постачальники хмарних послуг:

- **Інфраструктура як послуга (IaaS)** – віртуальні сервери, сховища даних, мережеві ресурси і інші базові інфраструктурні елементи.
- **Платформа як послуга (PaaS)** – платформи для розробки, тестування, розгортання і управління додатками.
- **Програмне забезпечення як послуга (SaaS)** – доступ до програмних додатків через Інтернет без необхідності їх установки на пристрої користувача.

Платформа ThingsBoard.io

ThingsBoard [28] – це відкрита платформа для IoT, яка дозволяє керувати пристроями, збирати і обробляти дані, а також створювати дашборди для візуалізації інформації. Вона забезпечує інтеграцію з різними протоколами, такими як MQTT, CoAP, HTTP, і може бути розгорнута як в хмарі, так і на локальному сервері.

Офіційний посібник «Getting Started with ThingsBoard Cloud» [29] містить вичерпну інформацію про використання хмарної платформи, включаючи процес налаштування облікового запису, підключення і налаштування пристроїв, керування, візуалізацію та інше.

Серводвигун

Елемент Servo [30] (рис. 6.1) моделює стандартний серводвигун, який часто використовується в робототехніці та електроніці. Серводвигун зазвичай має три піни: живлення, земля та сигнальний пін, який отримує сигнал від мікроконтролера для встановлення положення серводвигуна.

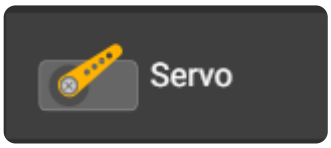


Рис. 6.1. Серводвигун в симуляторі Wokwi

Бібліотека ESP32Servo [31] дозволяє керувати сервоприводами за допомогою мікроконтролера ESP32.

Приклад програми:

```
#include <ESP32Servo.h> // Підключення бібліотеки для керування серводвигуном

Servo servo; // Створення об'єкту для керування серводвигуном

void setup()
{
  servo.attach(14); // Підключення серводвигуна до піна 14
}

void loop()
{
  servo.write(0);   // Оберт серводвигуна на 0 градусів
  delay(500);       // Затримка 500 мілісекунд
  servo.write(45);  // Оберт серводвигуна на 45 градусів
  delay(500);       // Затримка 500 мілісекунд
  servo.write(90);  // Оберт серводвигуна на 90 градусів
  delay(500);       // Затримка 500 мілісекунд
  servo.write(180); // Оберт серводвигуна на 180 градусів
  delay(500);       // Затримка 500 мілісекунд
}
```

Завдання

1. Щосекунди надсилати (по Wi-Fi) показники температури та вологості до хмарної платформи ThingsBoard за протоколом MQTT.
2. Налаштувати панель приладів IoT (Dashboard) для моніторингу показників температури та вологості.
3. На Dashboard додати елементи для керування кожним з трьох світлодіодів та серводвигуном.
4. Додати в програму можливість отримувати команди з хмари та реагувати на них, а саме: вмикати/вимикати світлодіоди, змінювати положення вихідного валу серводвигуна.

Додаткове завдання. Налаштувати «Alarm» (на платформі ThingsBoard), що буде видавати сповіщення при температурі вище 50 градусів Цельсія. Налаштувати відображення таких сповіщень на панелі приладів.

Порядок виконання роботи

Крок 1. Реєстрація та створення аккаунту:

- Відвідайте демо-вебсайт ThingsBoard (https://demo.thingsboard.io/) і зареєструйте обліковий запис, якщо його ще немає.
- Увійдіть у свій обліковий запис, щоб отримати доступ до платформи.

Крок 2. Після входу в систему створіть новий пристрій, натиснувши наступні кнопки «Панель керування ⟶ Entities ⟶ Devices ⟶ Add device ⟶ Add new device ⟶ *вказати назву пристрою, наприклад*: MyEsp32 ⟶ Add».

Крок 3. Написання програми:

- Створіть новий проект на базі ESP32 у Wokwi.
- Підключення бібліотек для роботи з Wi-Fi та MQTT (модель pub-sub):

```
#include <WiFi.h>
#include <PubSubClient.h>
```

- Встановлення назви мережі Wi-Fi та паролю:

```
const char *ssid = "Wokwi-GUEST";
const char *password = "";
```

- Встановлення облікових даних для підключення до MQTT-брокера платформи ThingsBoard:

```
const char* clientId = "ClientId_пристрою_на_платформі_ThingsBoard";
const char* mqttUsername = "UserName_пристрою_на_платформі_ThingsBoard";
const char* mqttPassword = "Password_пристрою_на_платформі_ThingsBoard ";
```

На сайті ThingsBoard можливо згенерувати ці дані для кожного пристрою обравши потрібний та натиснувши «Device details ⟶ Manage Credentials ⟶ MQTT Basic».

- Налаштування параметрів MQTT-брокера. У вікні «Device details», натиснувши кнопку «Check connectivity», можливо переглянути параметри різних протоколів для обміну даними. Оберіть протокол MQTT і скопіюйте надані параметри для використання в коді:

```
const char* mqttServer = "demo.thingsboard.io";
int mqttPort = 1883;
```

- Створення клієнту **PubSubClient**, який використовує Wi-Fi для з'єднання:

```
WiFiClient wifiClient;
PubSubClient client(wifiClient);
```

- Створення змінної для імітації показника температури отриманого з давача:

```
int temperature = 1;
```

- Ініціалізація послідовного порту (для виведення даних у консоль), підключення до Wi-Fi та встановлення адреси і порта MQTT-сервера (брокера):

```
void setup()
{
  // Ініціалізація порту для виведення даних у консоль
  Serial.begin(115200);
  // Підключення до Wi-Fi
  WiFi.begin(ssid, password);
  Serial.print("Підключення до Wi-Fi...");
  while(WiFi.status() != WL_CONNECTED)
  {
    delay(300);
    Serial.print(".");
  }
  Serial.println("\nПідключення успішне");

  client.setServer(mqttServer, mqttPort);
}
```

- Реалізація основного циклу:

```
void loop()
{
  if (!client.connected())
  {
    reconnect();
  }
  Serial.println("Поточна температура: " + String(temperature));
  String payload = "{\"temperature\":" + String(temperature) + "}";
  client.publish("v1/devices/me/telemetry", payload.c_str());
  temperature++;
  delay(1000);
  client.loop();
}
```

Кожна ітерація **loop** розпочинається з перевірки з'єднання, якщо воно відсутнє – виконується підпрограма **reconnect**, в іншому випадку в серійний монітор виводиться

поточне значення температури (за допомогою конкатенації, оператор «+») та формується JSON-рядок (згідно правил платформи ThingsBoard) для передачі на сервер. Відправляється цей рядок за допомогою методу **client.publish(*topic, payload*)**, де:

- *topic* – це "**v1/devices/me/telemetry**", де замість «**me**» буде автоматично підставлено ідентифікатор пристрою.
- *payload* – на першій ітерації буде JSON-рядком **{"temperature" : "1"}**, який, за допомогою функція c_str(), конвертується у масив символів.

В кінці імітується збільшення температури на один градус та встановлюється затримка, що обмежує кількість відправлених на сервер повідомлень до одного на секунду.

- Підпрограма **reconnect** перевіряє стан Wi-Fi і відновлює з'єднання за потреби. Також вона відповідає за підключення до MQTT-сервера з повторенням спроб кожні 3 секунди у разі невдачі. Для підключення пристрою до сервера використовуються його облікові дані на платформі ThingsBoard: **clientId**, **mqttUsername** та **mqttPassword**.

```
void reconnect()
{
  //Перевірка і перепідключення до Wi-Fi
  while (!client.connected())
  {
    if (WiFi.status() != WL_CONNECTED)
    {
      Serial.print("Підключення до Wi-Fi...");
      WiFi.begin(ssid, password);
      while(WiFi.status() != WL_CONNECTED)
      {
        delay(300);
        Serial.print(".");
      }
      Serial.println("Підключено");
    }

    //Перепідключення до хмари
    Serial.println("Підключення до ThingsBoard...");
    if (client.connect(clientId, mqttUsername, mqttPassword))
    {
      Serial.println("Клієнт: " + String(clientId) + " підключений.");
    }
    else
    {
       Serial.println("помилка: " + String(client.state()) + " повторна спроба
через 3 секунди.");
      delay(3000);
    }
  }
}
```

Крок 4. Запуск та перевірка.

Завантаживши та запустивши створену програму, за допомогою повідомлень у серійному моніторі, можливо впевнитися, що:

- Підключення до мережі Wi-Fi встановлено:

```
Підключення до Wi-Fi......
Підключення успішне
```

- З'єднання з сервером ThingsBoard налаштовано вірно та клієнт підключений:

```
Підключення до ThingsBoard...
Клієнт: {device_id} підключений.
```

- Показники температури продовжують надсилатися кожну секунду:

```
Поточна температура: 1
Поточна температура: 2
Поточна температура: 3
```

Доставку даних на сервер можливо перевірити виконавши наступне:

- Увійдіти до ThingsBoard
- В меню зліва обрати «Devices» та натиснути на пристрій, для якого потрібно перевірити дані.
- Перейдти на вкладку «Latest Telemetry», де відображаються останні дані, які були отримані сервером від обраного пристрою. В рядку «temperature» значення повинні оновлюватись кожну секунду.

Крок 5. Під'єднайте до ESP32 давач DHT22, три світлодіоди та серводвигун (рис. 6.2) або замініть вміст файлу **`diagram.json`** наступним текстом:

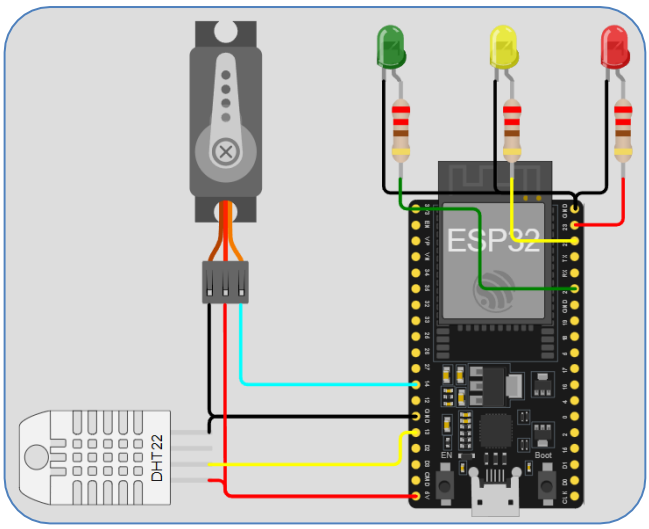


Рис. 6.2. Схема підключення компонентів

```
{ "version": 1, "author": "", "editor": "wokwi", "parts": [ { "type": "board-esp32-devkit-c-v4", "id":
"esp", "top": 28.8, "left": -4.76, "attrs": {} }, { "type": "wokwi-servo", "id": "servo1", "top":
-29.8, "left": -199.8, "rotate": 270, "attrs": {} }, { "type": "wokwi-dht22", "id": "dht1", "top":
148.8, "left": -209.7, "rotate": 270, "attrs": { "temperature": "51.7" } }, { "type": "wokwi-led",
"id": "led1", "top": -61.2, "left": -34.6, "attrs": { "color": "green" } }, { "type": "wokwi-led",
```

```
"id": "led2", "top": -61.2, "left": 32.6, "attrs": { "color": "yellow" } }, { "type": "wokwi-led",
"id": "led3", "top": -61.2, "left": 99.8, "attrs": { "color": "red" } }, { "type": "wokwi-resistor",
"id": "r1", "top": 4.8, "left": 95.45, "rotate": 90, "attrs": { "value": "220" } }, { "type":
"wokwi-resistor", "id": "r2", "top": 4.8, "left": 28.25, "rotate": 90, "attrs": { "value": "220" } },
{ "type": "wokwi-resistor", "id": "r3", "top": 4.8, "left": -38.95, "rotate": 90, "attrs": { "value":
"220" } } ], "connections": [ [ "esp:TX", "$serialMonitor:RX", "", [] ], [ "esp:RX",
"$serialMonitor:TX", "", [] ], [ "esp:GND.1", "servo1:GND", "black", [ "h-124.65", "v-67.2" ] ], [
"esp:5V", "dht1:VCC", "red", [ "h-115.05", "v-9.6" ] ], [ "esp:GND.1", "dht1:GND", "black", [ "h0" ]
], [ "led3:C", "esp:GND.2", "black", [ "v67.2", "h-18.8" ] ], [ "led2:C", "esp:GND.2", "black", [
"v67.2", "h48.4" ] ], [ "led1:C", "esp:GND.2", "black", [ "v67.2", "h115.6" ] ], [ "servo1:V+",
"esp:5V", "red", [ "h0.1", "v115.2" ] ], [ "servo1:PWM", "esp:14", "cyan", [ "v0" ] ], [ "dht1:SDA",
"esp:13", "yellow", [ "h115.2", "v-19.3" ] ], [ "r1:1", "led3:A", "red", [ "h0" ] ], [ "r1:2",
"esp:23", "red", [ "v27.6", "h-28.8" ] ], [ "esp:22", "r2:2", "yellow", [ "h0" ] ], [ "r2:1", "led2:A",
"yellow", [ "h0" ] ], [ "esp:21", "r3:2", "green", [ "h-57.6", "v-48", "h-48" ] ], [ "r3:1", "led1:A",
"green", [ "h0" ] ] ], "dependencies": {}}
```


Крок 6. Модифікуйте наявний код для щосекундного опитування давача DHT22 та надсилання показників температури і вологості в хмару.

Крок 7. На платформі ThingsBoard створіть панель приладів IoT (Dashboard). Вона повинна відображати показники температури та вологості у вигляді графіків, а також містити віджети для керування серводвигуном та кожним з світлодіодів. На панель приладів можливо додади будь-який тип віджету, головне правильно його налаштувати. Для керування світлодіодом необхідно:

- Обрати перемикач з набору віджетів (напр. Buttons ⟶ Power button).
- В спливаючому вікні (рис. 6.3), в полі «Target device» обрати потрібний девайс (MyESP32).

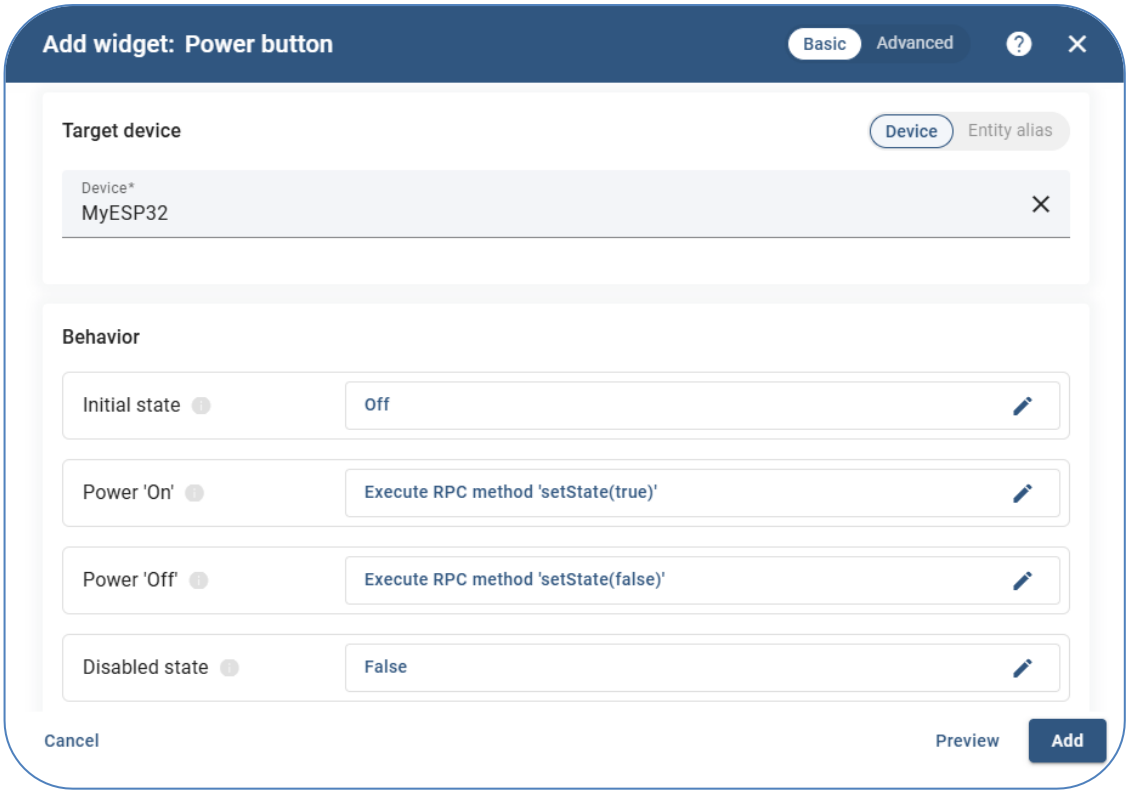


Рис. 6.3. Налаштування зв’язку між віджетом та пристроєм на платформі ThingsBoard

- Ознайомитись з поточними налаштуваннями поведінки (Behavior) перемикача (рис. 6.3), де:
  - **Initial state** – визначає дію для отримання початкового стану віджета або її відсутність.
  - **Power 'On'** – при «увімкненні» перемикача надсилається віддалений виклик функції (RPC) **`setState`** (із значенням **`true`**) від серверної програми до пристрою MyESP32.
  - **Power 'Off'** – при «вимкненні» перемикача надсилається віддалений виклик функції (RPC) **`setState`** (із значенням **`true`**) від серверної програми до пристрою MyESP32.
- Налаштувати кожну дію натиснувши на символ олівця. Наприклад, для увімкнення першого світлодіода, можливо налаштувати виконання віддаленого виклику функції LED1 з параметром типу String «Увімкнути» (рис. 6.4).

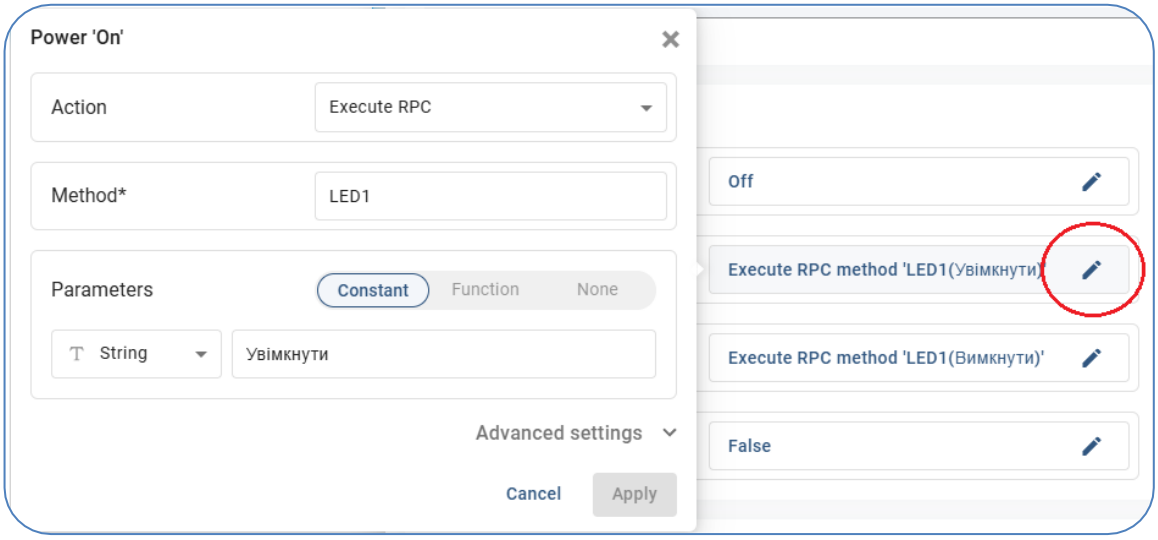


Рис. 6.4. Налаштування дії Power 'On'

- Налаштувати решту віджетів для керування усіма світлодіодами. Приклад отриманих повідомлень від сервера пристроєм ESP32 зображено на рисунку 6.5.

Рис. 6.5. Зв’язок між налаштуваннями та отриманим повідомленням на пристрої

Для керування серводвигуном можливо використовувати віджет Control widgets → Knob Control. Якому, в розділі налаштувань «Appearance», можливо відредагувати початкове, мінімальне та максимальне значення (рис. 6.6).

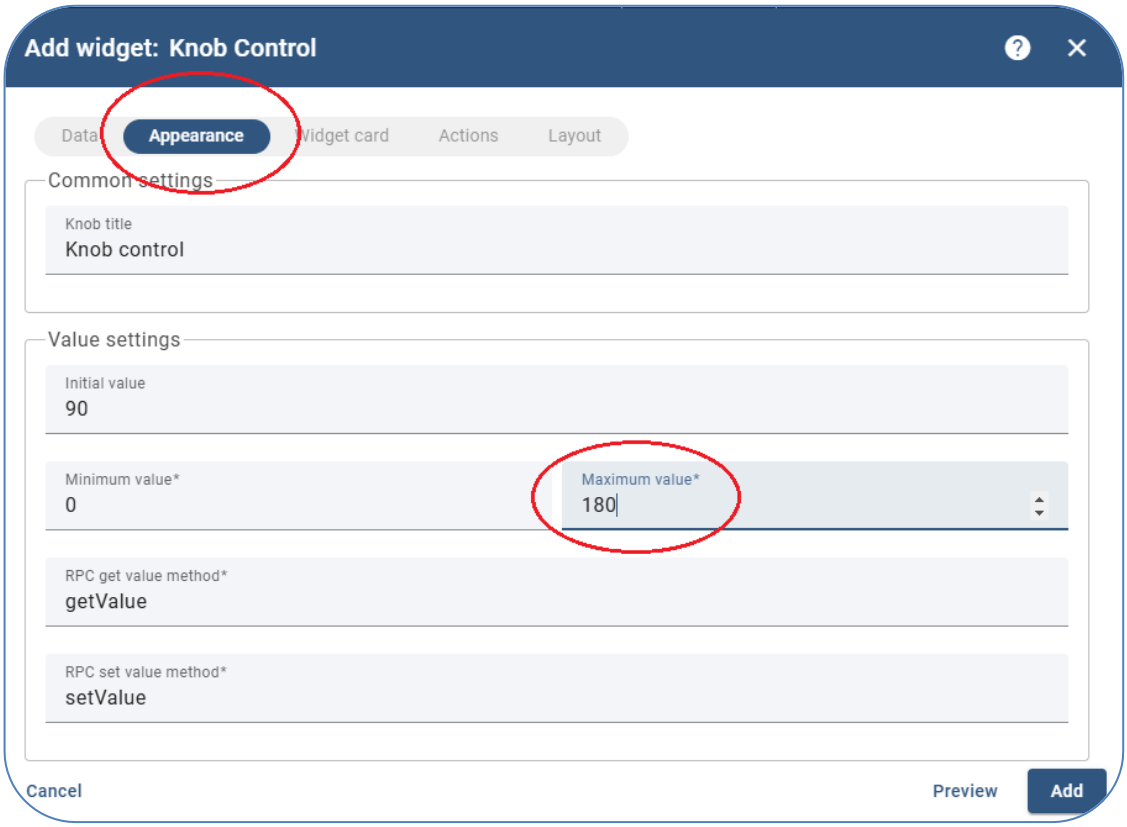


Рис. 6.6. Налаштування віджета Knob Control

Крок 8. Написання коду для обробки вхідних повідомлень:

- Після підключення до MQTT-брокера ThingsBoard, для отримання повідомлень від брокера необхідно підписатись на відповідну тему **"v1/devices/me/rpc/request/+"**:

```
if (client.connect(clientId, mqttUsername, mqttPassword))
{
    Serial.println("Клієнт: " + String(clientId) + " підключений.");
    client.subscribe("v1/devices/me/rpc/request/+");
}
```

- В **setup()**, для можливості обробки вхідних повідомлень, необхідно призначити **callback** функцію, яка буде викликатись при їх наявності:

```
client.setServer(mqttServer, mqttPort);
client.setCallback(callback);
```

- В функції **callback** необхідно реалізувати десеріалізацію отриманих повідомлень та їх обробку, тобто – виконання дій відповідно до завдання. Шаблон:

```
void callback(char* topic, byte* message, unsigned int length)
{
  String stMessage;
```

```
  for (int i = 0; i < length; i++)
  {
    stMessage += (char)message[i];
  }

  //Вивід всіх повідомлень в консоль
  Serial.print("Отримано повідомлення з топіка ");
  Serial.print(topic);
  Serial.print(": ");
  Serial.println(stMessage);
}
```

Крок 9. Надання публічного доступу до створеної панелі приладів:

- Перейти на сторінку «Devices» та натиснути «Make device public» (рис. 6.7), щоб зробити відповідний пристрій публічними.

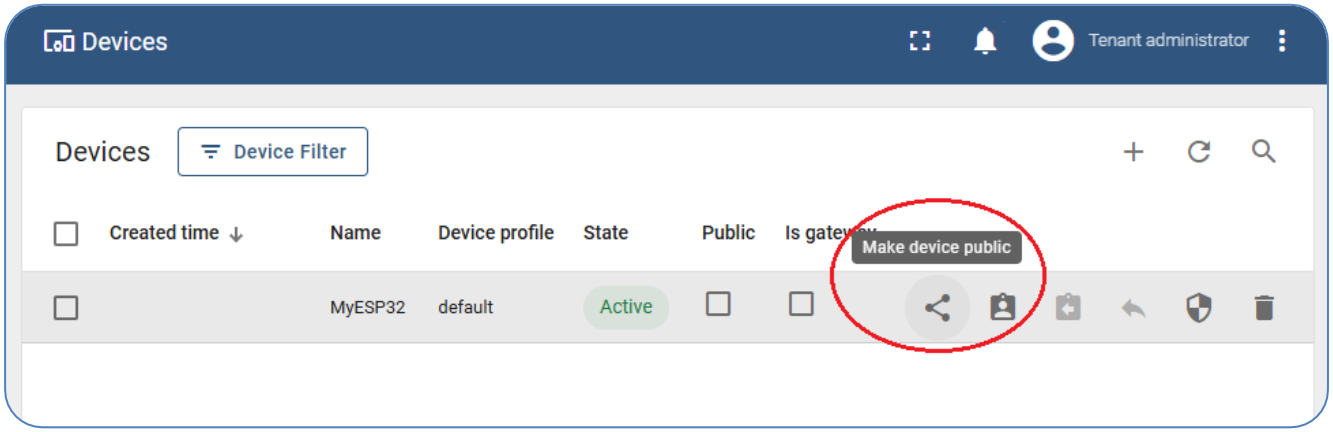


Рис. 6.7. Встановлення публічного доступу до пристрою

- Перейти на сторінку «Dashboards» натиснути «Make dashboard public» (рис. 6.8), щоб зробити відповідну панель приладів публічною.

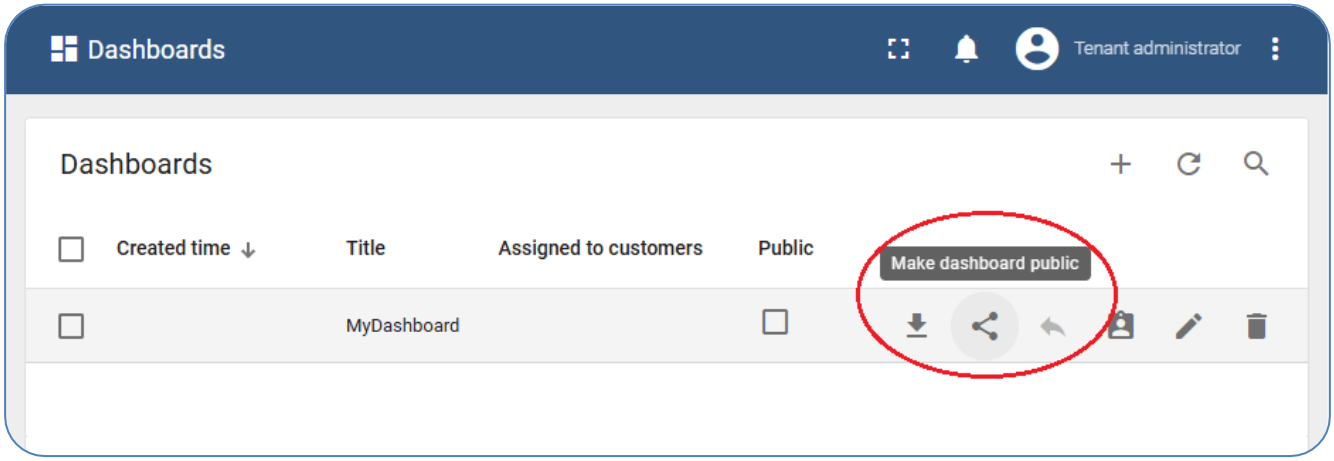


Рис. 6.8. Встановлення публічного доступу до панелі приладів

- Натиснути на кнопку (рис. 6.9) та скопіювання посилання на створений Dashboard.

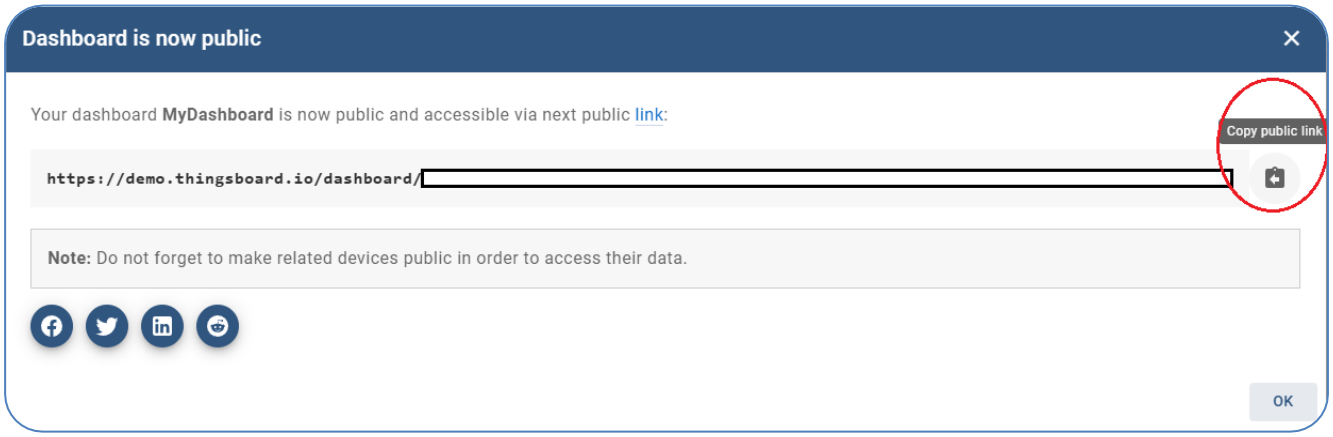


Рис. 6.9. Створення публічного посилання на Dashboard

- Відкрити нове вікно браузера в «анонімному» режимі та перейти за отриманим посиланням.
- Запустити програму у симуляторі Wokwi та перевірити працездатність панелі приладів.

Оформлення звіту та порядок його подання

1. Увійдіть в дистанційний курс Moodle з власного аккаунту.
2. Відкрийте активність присвячену даній лабораторній роботі.
3. Натисніть кнопку «Здати роботу».
4. У текстове поле завантажте:
    - посилання на створений проект в симуляторі Wokwi з виконаними завданнями;
    - публічне посилання на створену панель приладів (Dashboard);
    - висновки до лабораторної роботи.
5. Натисніть кнопку «Зберегти».

Контрольні запитання

1. Які кроки потрібно виконати для налаштування підключення до хмарної платформи IoT?
2. Які механізми безпеки застосовуються при взаємодії з платформою?
3. Які протоколи прикладного рівня підтримуються розглянутою платформою?
4. Які можливості забезпечує панель приладів для керування вбудованими системами?

ЛАБОРАТОРНА РОБОТА № 7

Тема: Паралельне виконання завдань з використанням RTOS

Мета: Ознайомитись з принципами та практичним застосуванням операційних систем реального часу (RTOS) та навчитись паралельного виконувати декілька різних завданнь.

План проведення лабораторної роботи

- Ознайомлення з короткими теоретичними відомостями.
- Виконання завдання згідно порядку виконання роботи.
- Подання звіту в Moodle (кнопка «Здати роботу»).

Короткі теоретичні відомості

Операційна система (ОС) – це програмне забезпечення, яке керує апаратними ресурсами комп'ютера або вбудованої системи та забезпечує виконання інших програм. Основні функції ОС включають управління файлами, процесами, пам'яттю, введенням/виведенням даних та забезпечення користувацького інтерфейсу.

Переваги використання ОС у вбудованих системах:

- **Зручність інтеграції та стандартизація**. ОС абстрагує специфіку апаратного забезпечення, пропонуючи уніфіковані методи взаємодії з різними пристроями. Це означає, що код, написаний для роботи з певним периферійним пристроєм в одній системі, може бути легко перенесений на іншу систему.
- **Управління ресурсами**. ОС забезпечує ефективне управління ресурсами, такими як процесорний час, пам'ять, вхідні/вихідні операції.
- **Спрощення розробки**. Багато стандартних задач вже вирішені і реалізовані в ОС, включаючи набори драйверів для різних периферійних пристроїв, завдяки чому, програмісти можуть зосередитися на розробці програмного забезпечення високого рівня та реалізації специфічних функцій, не займаючись низькорівневим програмуванням для кожного різновиду апаратури.
- **Керування багатозадачністю**. ОС дозволяють виконувати кілька завдань одночасно, що важливо для систем, де, наприклад, потрібно паралельно обробляти дані сенсорів, керувати пристроями та комунікувати через мережу.
- **Безпека та надійність**. Багато ОС пропонують засоби для забезпечення безпеки, такі як контроль доступу, шифрування даних, вірусні сканери, мережеві екрани та інше.

У вбудованих системах використовуються різні операційні системи, які можливо розділити на два основних типи:

- **Операційні системи загального призначення (GPOS, General Purpose Operating System)** – використовуються у різних сценаріях управління великою кількістю процесів та виконання багатьох завдань за одиницю часу, де немає жорстких вимог до виконання в точно визначений час. Для забезпечення доступу до CPU різним процесам і задачам GPOS використовують політику розподілу ресурсів. Забезпечення великої

продуктивності відбувається шляхом виконання якумога більшої кількості завдань, не дивлячись на їх важливість, що може спричинити затримку виконання для дійсно важливих завдань.

- **Операційні системи реального часу (RTOS, Real-time Operating System)** гарантують, що виконання операцій та завдань відбувається за строго визначеними часовими рамками. Це дозволяє передбачати та керувати часовою поведінкою системи. Система готова негайно реагувати на події та виконувати відповідні завдання з найменшими можливими затримками. В RTOS завдання може мати визначений пріоритет, і система віддає перевагу виконанню важливіших завдань перед менш важливими.

Вибір між звичайною операційною системою (GPOS) та операційною системою реального часу (RTOS) залежить від конкретних вимог та характеристик системи, що розробляється.

Рекомендації для застосування RTOS:

- Потребується невелика та легка ОС для полегшення взаємодії з апаратною частиною.
- Вбудована система має жорсткі вимоги до часу виконання завдань і вимагає гарантованої відповіді на події у визначений термін.
- В системі присутні багато завдань і важливо забезпечити правильну взаємодію та синхронізацію між ними.
- Вбудована система сумісна з RTOS.

RTOS дає можливість запустити кілька завдань/потоків на мікроконтролері, а також прогнозувати точний час виконання кожного з них. Замість того, щоб вся програма виконувалась циклічно в єдиному циклі Super Loop, кожне завдання виконується в окремому, незалежному циклі (рис. 7.1). Важливим завданням можливо встановити вищий пріорітет виконання, щоб вони виконувались швидше або мали більше часу CPU.

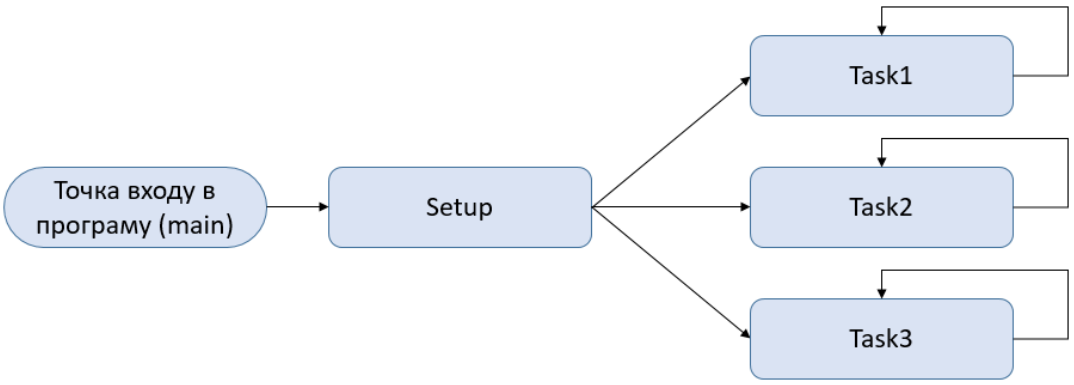


Рис. 7.1. Паралельне виконання багатьох завдань

Кожне завдання потребує певного часу CPU на виконання. При наявності чотирьох завдань в 4-ядерній системі, за кожне з них може відповідати окреме ядро (рис. 7.2). В такому випадку буде відбуватись одночасне виконання всіх завдань.

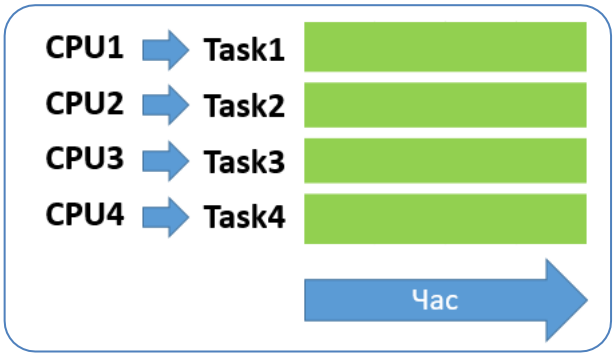


Рис. 7.2. Паралельне виконання завдань в чотириядерній системі

В системі з одним ядром, процесор може виконувати лише одне завдання одночасно, але він може швидко перемикатися між завданнями, що створює ілюзію паралельного виконання (рис. 7.3).

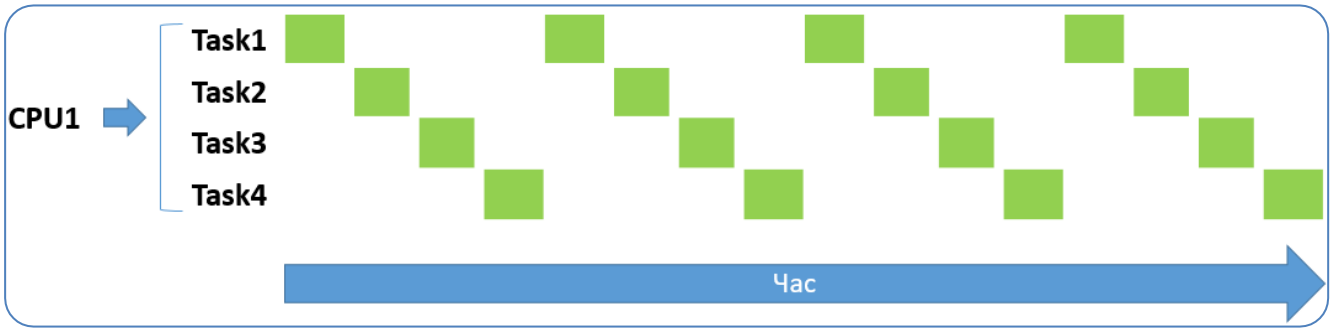


Рис. 7.3. Паралельне виконання завдань в одноядерній системі

Використання FreeRTOS

Щоб інсталювати FreeRTOS і почати її використовувати на мікроконтролері, потрібно виконати наступні кроки:

1) Завантажити з офіційного сайту FreeRTOS (https://freertos.org/) архів з вихідним кодом системи. В архіві знаходяться дві основні папки: FreeRTOS та FreeRTOS-Plus. Кожна з них має своє призначення.
   - Папка FreeRTOS включає файли ядра FreeRTOS для різних процесорів, а також приклади використання системи на різних платформах і в різних середовищах розробки.
   - Папка FreeRTOS-Plus містить додаткові компоненти, які розширюють можливості системи. Сюди входять бібліотеки для роботи з мережею, безпекою, файловими системами та інше.
2) У папці FreeRTOS/Demo оберіть підпапку, яка відповідає вашій апаратній платформі та середовищу розробки.
3) Імпортуйте наведені файли до власного проекту. Додаткові налаштування можливо виконати у файлі `FreeRTOSConfig.h`.
4) Скомпілюйте приклад, завантажити зібраний код у мікроконтролер та перевірити роботу демонстраційної програми.

Примітка. Мікроконтролери ESP32 вже мають вбудовану FreeRTOS, тому виконувати наведені вище пункти не потрібно.

Основні функції FreeRTOS

Основні функції FreeRTOS можна розділити на кілька груп відповідно до їх призначення:

- Управління задачами – функції для створення, керування, призупинення, відновлення та видалення задач, наприклад: **`xTaskCreate`**, **`vTaskSuspend`**, **`vTaskResume`**, **`vTaskDelete`**.
- Механізми синхронізації – використовуються для синхронізації задач та управління доступом до спільних ресурсів: **`xSemaphoreCreateBinary`**, **`xSemaphoreTake`**, **`xSemaphoreGive`**.
- Передача даних між задачами: **`xQueueCreate`**, **`xQueueSend`**, **`xQueueReceive`**.
- Таймери для виконання періодичних завдань або відкладених операцій, наприклад: **`xTimerCreate`**, **`xTimerStart`**, **`xTimerStop`**.
- Отримання інформації про стан системи та управління пам'яттю, наприклад: **`uxTaskGetStackHighWaterMark`**, **`pvPortMalloc`**, **`vPortFree`**.

Для виконання даної роботи достатньо використання функцій із групи «управління задачами». З усіма функціями FreeRTOS детально можливо ознайомитись на офіційному на сайті [32] в розділі API references. З додатковими функціями, які присутні у версії FreeRTOS для ESP32 можливо ознайомитись у документації на сайті Espressif [33].

`xTaskCreate` [34] – використовується для створення нової задачі. Якщо задача створена успішно повертається значення **`pdPASS`**, а в іншому випадку – **`errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY`**.

Синтаксис:

```
xTaskCreate(pvTaskCode, pcName, usStackDepth, pvParameters, uxPriority, pxCreatedTask);
```

- `pvTaskCode` – вказівник на функцію, яка буде виконуватись як задача. Така функція повинна бути визначена як «**`void`** *`НазваФункції`***`(void *pvParameters)`**».
- `pcName` – ім'я задачі, яке можна використовувати для дебагінгу чи відстеження задач. Воно не має ніякого функціонального впливу на виконання задачі.
- `usStackDepth` – розмір стека вказує скільки пам'яті виділяється задачі для виконання.
- `pvParameters` – вказівник на параметри, які будуть передані задачі. Якщо параметри не потрібні, цей аргумент можна встановити в **`NULL`**.
- `uxPriority` – пріоритет задачі. Задачі з вищим пріоритетом отримують більше процесорного часу. FreeRTOS дозволяє налаштувати пріоритети задач в діапазоні від 0 до максимального значення, визначеного конфігурацією (**`configMAX_PRIORITIES`**).
- `pxCreatedTask` – вказівник на змінну, в яку буде записано дескриптор створеної задачі. Цей дескриптор може бути використаний для подальших маніпуляцій з задачею (наприклад, для призупинення чи відновлення). Якщо це не потрібно, можна встановити **`NULL`**.

`xTaskCreatePinnedToCore` – функція, що є розширеною версією **`xTaskCreate`** і використовується в системах на базі ESP32, які мають два ядра. Вона дозволяє не лише створити задачу, а і закріпити її виконання задачі за певним ядром процесора.

Синтаксис:

```
xTaskCreatePinnedToCore(pvTaskCode, pcName, usStackDepth, pvParameters, uxPriority,
                        pxCreatedTask, xCoreID);
```

Приклад:

```
xTaskCreatePinnedToCore(MusicPlayer,"MusicPlayer",10000,NULL,1,&MusicPlayerHandle,0);
```

- `MusicPlayer` – функція, яка буде виконуватися.
- `"MusicPlayer"` – ім'я, що надається завданню.
- `10000` – розмір стеку, області пам'яті, де зберігаються локальні змінні завдання та інші деталі.
- `NULL` – вказує, що жодних додаткових параметрів функції MusicPlayer не передається.
- `1` – пріоритет завдання, вказує, що воно буде виконуватись перед завданнями з меншим пріорітетом (0).
- `&MusicPlayerHandle` – адреса змінної для збереження ідентифікатора створеного завдання.
- `0` – номер ядра, на якому буде виконуватися завдання. У двоядерних ESP32 «0» – першоме ядро, «1» – друге.

`vTaskSuspend` [35] – використовується для призупинення виконання задачі. Призупинена задача не буде виконуватись, поки її не відновлять за допомогою функції **`vTaskResume`**.

Синтаксис:

```
vTaskSuspend(xTaskToSuspend);
```

- `xTaskToSuspend` – дескриптор задачі, яку потрібно призупинити. Якщо цей параметр дорівнює **`NULL`**, то призупиняється викликаюча задача.

`vTaskResume` [36] – використовується для відновлення задачі, яка була призупинена за допомогою **`vTaskSuspend`**. Після виклику цієї функції задача знову почне виконуватись відповідно до її пріоритету.

Синтаксис:

```
vTaskResume(xTaskToResume);
```

- `xTaskToResume` – дескриптор задачі, яку потрібно відновити.

`vTaskDelay` – використовується для введення затримки виконання задачі на певний проміжок часу. Дозволяє іншим задачам виконуватися під час цієї затримки, на відміну від **`delay`**, що просто блокує виконання всього коду і може призводити до неефективного використання процесорного часу.

Синтаксис:

```
vTaskDelay(xTicksToDelay);
```

- `xTicksToDelay` – кількість тактів системного таймера, на які треба призупинити задачу. Щоб задати час затримки у мілісекундах, потрібно перевести їх в такти (ticks). Це можна зробити розділивши кількість мілісекунд на довжину такту, наприклад: «**`500/portTICK_PERIOD_MS`**». Або за допомогою макросу **`pdMS_TO_TICKS`**, який конвертує мілісекунди в кількість тактів на основі налаштувань частоти таймера: «**`vTaskDelay(pdMS_TO_TICKS(500));`**».

Вся логіка програми реалізується у контексті функцій завдань, які виконуються безкінечно, наприклад:

```
// Реалізація функції завдання
void TaskBlinkLED(void *pvParameters)
{
  for(;;) // Нескінченний цикл
  {
```

```
    digitalWrite(2, HIGH);  // Увімкнути світлодіод
    vTaskDelay(1000 / portTICK_PERIOD_MS); // Затримка на 1 секунду
    digitalWrite(2, LOW);   // Вимкнути світлодіод
    vTaskDelay(1000 / portTICK_PERIOD_MS); // Затримка на 1 секунду
  }
}
```

При цьому, головний цикл **`loop`** залишається порожнім, хоча включити код в нього можливо, але це може призвезти до виникнення проблем та конфліктів у роботі програми.

Завдання

1. У вигляді окремого завдання (Task) реалізувати обробку натискання кнопок:
   - «Rewind» – для відтворення попередньої мелодії.
   - «Forward» – для відтворення наступної мелодії.
   - «Pause» – для призупинення відтворення мелодії (кнопка «Play», відповідно, повинна поновлювати відтворення мелодії з місця зупинки).
2. У вигляді окремого завдання реалізувати вивід назви відтворюваної мелодії на дисплей (можна назвати кожну мелодію довільно або навіть додати інші, замість існуючих).
3. Підключиим ESP32 до хмарної платформи ThingsBoard та додати на панель приладів (Dashboard) елементи для відображення назви поточної мелодії і кнопки для керування відтворенням (Rewind, Play, Pause, Forward).
4. Реалізувати відправку (по Wi-Fi) назви відтворюваної мелодії до хмарної платформи ThingsBoard за протоколом MQTT.
5. Реалізувати обробку повідомлень, отриманих з хмарної платформи після натискання відповідних кнопок на панелі приладів: Rewind, Play, Pause, Forward.

Додаткове завдання 1. Після «збору», за допомогою джойстика, всіх літер «О» – виводити привітальне повідомлення з кількістю вдалих спроб та наповнювати екран новими символами.

Додаткове завдання 2. Виконати всі завдання на платформі Arduino Cloud замість ThingsBoard.

Порядок виконання роботи

Крок 1. Створення нового проекту у симуляторі Wokwi та підключення елементів до ESP32 відповідно до схеми (рис. 7.4) або заміна вмісту **`diagram.json`** відповідним текстом:

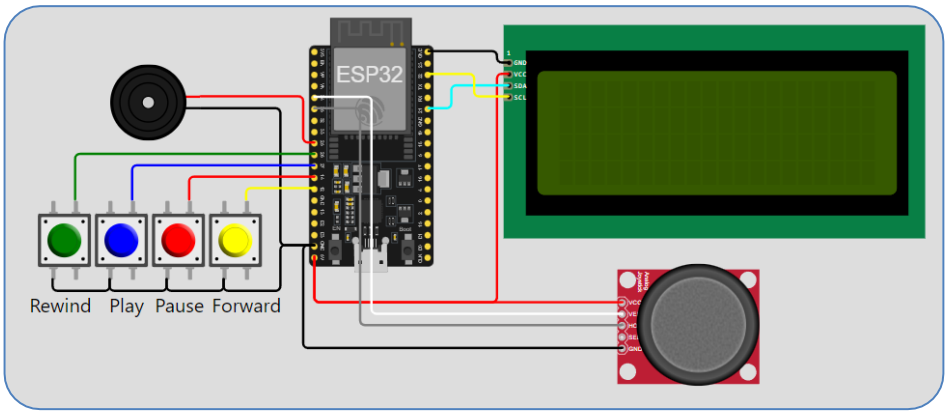


Рис. 7.4. Схема

```
{"version": 1, "author": "", "editor": "wokwi", "parts": [{"type": "board-esp32-devkit-c-v4", "id":
"esp", "top": 9.6, "left": -187.16, "attrs": {}}, {"type": "wokwi-lcd2004", "id": "lcd1", "top": 16,
"left": -23.2, "attrs": {"pins": "i2c"}}, {"type": "wokwi-pushbutton", "id": "btn1", "top": 172.7,
"left": -284.5, "rotate": 90, "attrs": {"color": "yellow"}}, {"type": "wokwi-pushbutton", "id":
"btn2", "top": 172.7, "left": -428.5, "rotate": 90, "attrs": {"color": "green"}}, {"type":
"wokwi-buzzer", "id": "bz1", "top": 33.9, "left": -357.9, "rotate": 270, "attrs": {"volume": "0.1"}},
{"type": "wokwi-pushbutton", "id": "btn3", "top": 172.7, "left": -380.5, "rotate": 90, "attrs":
{"color": "blue"}}, {"type": "wokwi-text", "id": "led1", "top": 240, "left": -422.4, "attrs": {"text":
"Rewind"}}, {"type": "wokwi-text", "id": "led2", "top": 240, "left": -355.2, "attrs": {"text":
"Play"}}, {"type": "wokwi-text", "id": "led3", "top": 240, "left": -268.8, "attrs": {"text":
"Forward"}}, {"type": "wokwi-pushbutton", "id": "btn4", "top": 172.7, "left": -332.5, "rotate": 90,
"attrs": {"color": "red"}}, {"type": "wokwi-text", "id": "led4", "top": 240, "left": -316.8, "attrs":
{"text": "Pause"}}, {"type": "wokwi-analog-joystick", "id": "joystick1", "top": 208.1, "left": 81.1,
"rotate": 90, "attrs": {}}], "connections": [["esp:TX", "$serialMonitor:RX", "", []], ["esp:RX",
"$serialMonitor:TX", "", []], ["lcd1:VCC", "esp:5V", "red", ["h-9.6", "v192.1", "h-153.75"]],
["esp:21", "lcd1:SDA", "cyan", ["h19.2", "v-19.2", "h48"]], ["lcd1:SCL", "esp:22", "yellow",
["h-28.8", "v-18.9"]], ["lcd1:GND", "esp:GND.2", "black", ["h-9.6", "v-9.6"]], ["bz1:1", "esp:CMD",
"black", ["h67.2", "v115.2"]], ["esp:CMD", "btn1:2.r", "black", ["h-28.65", "v38.4", "h-47.8"]],
["btn3:2.r", "btn2:2.r", "black", ["v9.8", "h-48.2"]], ["btn1:1.l", "esp:12", "yellow", ["v0"]],
["btn1:2.r", "btn4:2.r", "black", ["v9.8", "h-29"]], ["btn4:2.r", "btn3:2.r", "black", ["v9.8",
"h-38.6"]], ["esp:25", "bz1:2", "red", ["h-9.45", "v-29.2"]], ["esp:14", "btn4:1.l", "red", ["h0"]],
["esp:27", "btn3:1.l", "blue", ["h0"]], ["esp:26", "btn2:1.l", "green", ["h0"]], ["joystick1:VCC",
"esp:5V", "red", ["h0"]], ["joystick1:GND", "esp:CMD", "black", ["h-268.8", "v-86.4"]],
["joystick1:VERT", "esp:34", "white", ["h-211.2", "v-182.4"]], ["joystick1:HORZ", "esp:35", "gray",
["h-220.8", "v-182.4"]]], "dependencies": {}}
```


Крок 2. Додавання коду з файлу Example7.cpp [4] у редактор коду та ознайомлення з прикладом використання FreeRTOS.

Для запуску програми необхідно встановити бібліотеки зображені на рисунку 7.5.

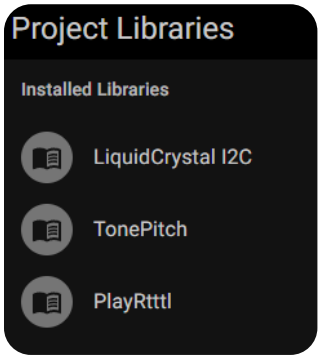


Рис. 7.5. Перелік бібліотек проекту

Наведений приклад демонструє використання мікроконтролера ESP32 (або подібних засобів) для одночасного управління дисплеєм, кнопками, джойстиком та відтворенням музики. Програму можна розділити на основні частини:

- У перших рядках визначаються константи для пінів, які використовуються для підключення різних пристроїв: дисплею, джойстика, кнопок, тощо.
- За допомогою бібліотеки LiquidCrystal_I2C створюється об'єкт для керування дисплеєм з вказаною адресою та розміром.

- Змінна **`MusicPlayerHandle`** використовується для зберігання ідентифікатора завдання, яке створене для відтворення музики.
- У функції **`setup`** встановлюються початкові налаштування мікроконтролера: ініціалізація зв'язку через серійний порт, налаштування пінів для кнопок, ініціалізація дисплею, створення та призупинення виконання завдання **`MusicPlayer`**, створення завдань для кнопок та джойстика.
- Функція **`MusicPlayer`** у нескінченному циклі відтворює послідовно всі мелодії.
- Функція **`ButtonPlay`** обробляє натискання кнопки "Play" і відновлює виконання функції **`MusicPlayer`**.
- Функція **`Joystick`** відповідає за керування курсором на дисплеї. Залежно від значень, зчитаних з аналогових входів джойстика, курсор переміщується вгору, вниз, ліворуч та праворуч.
- Функція **`loop`** порожня, оскільки вся логіка виконується у вищезазначених функціях.
- В кінці наведено набір функцій, які відповідають за відтворення кожної з мелодій.

Крок 3. Обробка натискання кнопок Rewind, Forward, Pause:

- Налаштуйте піни до яких підключені кнопки.
- Створення функції обробки натискання кнопок, які будуть перевіряти стан кожної кнопки, а при виявленні натискання – виконувати відповідні дії.
- Створіть завдання FreeRTOS з вказівниками на відповідні функції.

Крок 4. Вивід назви або номеру відтворюваної мелодії на дисплей.

- Присвойте кожній мелодії назву або номер.
- Створіть функцію яка буде буде виводити на дисплей назву мелодії.
- Щоразу, коли відбувається зміна мелодії, викликайте функцію для оновлення дисплею.

Крок 5. Інтеграція з ThingsBoard:

- Створіть новий пристрій на платформі ThingsBoard: «Панель керування → Devices → Add device → MusicPlayer → Add» та згенеруйте відповідні облікові дані для його підключення.
- Підключіть до програми бібліотек для роботи з Wi-Fi та MQTT.
- Налаштуйте облікові дані мережі Wi-Fi та MQTT-брокера.
- Створіть PubSubClient та реалізуйте відправку назви або номеру пісні, що відтворюється, на платформу ThingsBoard.
- На платформі ThingsBoard додайте на панель приладів кнопки (Rewind, Play, Pause, Forward), а також елемент для відображення поточної мелодії. Налаштуйте відправку та отримання повідомлень даними віджетами.

Крок 6. Реалізуйте відправку назви або номеру відтворюваної мелодії до ThingsBoard через MQTT.

Крок 7. Реалізуйте функцію для отримання вхідних повідомлень від пратформи ThingsBoard, їх десеріалізації та виконання відповідних команд (Rewind, Play, Pause, Forward).

Оформлення звіту та порядок його подання

1. Увійдіть в дистанційний курс Moodle з власного аккаунту.
2. Відкрийте активність присвячену даній лабораторній роботі.
3. Натисніть кнопку «Здати роботу».
4. У текстове поле завантажте:
    - посилання на створений проект в симуляторі Wokwi з виконаними завданнями;
    - публічне посилання на створену панель приладів (Dashboard);
    - висновки до лабораторної роботи.
5. Натисніть кнопку «Зберегти».

Контрольні запитання

1. Що таке операційна система реального часу (RTOS)?
2. Які основні відмінності між GPOS та RTOS?
3. Чи можливо на одноядерному процесорі паралельно виконувати декілька завдань?
4. Як створити та запутити нове завдання?
5. В чому ключова відмінність функцій **`delay`** та **`vTaskDelay`**?

ЛАБОРАТОРНА РОБОТА № 8

Тема: Синхронізація доступу до спільних ресурсів в RTOS

Мета: Навчитися використовувати можливості RTOS для ефективного управління задачами та безпечного доступу до спільних даних та ресурсів.

План проведення лабораторної роботи

- Ознайомлення з короткими теоретичними відомостями.
- Виконання завдання згідно порядку виконання роботи.
- Подання звіту в Moodle (кнопка «Здати роботу»).

Короткі теоретичні відомості

Механізми синхронізації – це інструменти та техніки, які забезпечують управління і координацію доступу до спільних ресурсів і даних у багатозадачних системах. Вони дозволяють уникати проблем, таких як перегони за ресурсами, конфлікти доступу, а також непередбачуваної поведінки програми. FreeRTOS надає кілька механізмів синхронізації, основні з яких це: черги, м'ютекси, бінарні семафори та семафори з лічильником.

Черга (Queue)

Черга – це специфічна структура даних, яка реалізує концепцію «перший увійшов, перший вийшов» (FIFO, First In – First Out). Це означає, що елементи видаляються з черги в порядку їх надходження. Використання черг дозволяє організовувати асинхронний обмін даними між задачами.

На рисунку 8.1 зображено сценарій, в якому два завдання (Task 1 та Task 2) генерують дані та підправляють їх у чергу, а третє завдання (Task 3) отримує дані з черги та обробляє їх.

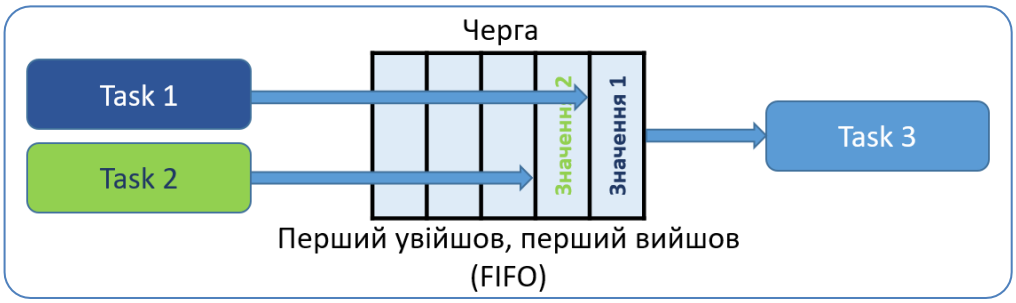


Рис. 8.1. Передача даних між завданнями за допомогою черги

Якщо в описаному сценарії, для передачі даних між завданнями, замість черги використовувати глобальну змінну – можуть виникнути кілька проблем:

- Task 1 і Task 2 можуть одночасно намагатися змінити значення глобальної змінної, що може призвести до спотворення інформації.
- Дані можуть бути втрачені або переписані, якщо Task 1 і Task 2 генерують дані швидше, ніж Task 3 може їх обробити.

Тому, використання черг забезпечує більш безпечний, організований і масштабований підхід для обміну даними між завданнями.

Робота з чергами в FreeRTOS [37] включає:

- Створення черги за допомогою функції **xQueueCreate** [38], яка повертає дескриптор черги **myQueue** типу **QueueHandle_t**, який використовується для подальших операцій з чергою. Якщо створення черги не вдалося (наприклад, через недостатню пам'ять), функція поверне **NULL**.

```
QueueHandle_t myQueue = xQueueCreate(10, sizeof(int));
```

  - **10** – максимальна кількість елементів, які можуть зберігатися в черзі одночасно.
  - **sizeof(int)** – розмір кожного елемента в черзі. У цьому випадку це розмір цілих чисел (int).
    - Відправка значення до черги за допомогою функції **xQueueSend** [39]:

```
int valueToSend = 14;
xQueueSend(myQueue, &valueToSend, portMAX_DELAY);
```

  - **myQueue** – дескриптор черги, куди відправляється дане значення.
  - **&valueToSend** – вказівник на змінну, значення якої потрібно відправити в чергу.
  - **portMAX_DELAY** – час очікування до успішної відправки даних. Значення **portMAX_DELAY** означає, що функція буде чекати необмежений час, поки черга не стане достатньо вільною.

  Якщо відправка була успішною, функція поверне **pdTRUE**, або **errQUEUE_FULL**, якщо черга була переповнена. Оскільки, в попередньому пункті, черга створена з розміром 10 і пустою, відправка повинна бути успішною.

- Отримання значення з черги за допомогою функції **xQueueReceive** [40]:

```
int receivedValue;
xQueueReceive(myQueue, &receivedValue, portMAX_DELAY);
```

  - **myQueue** – дескриптор черги, з якої потрібно отримати значення.
  - **&receivedValue** – вказівник на змінну, куди буде збережене отримане значення.
  - **portMAX_DELAY** – функція буде чекати необмежений час до того, як в черзі з'являться дані.

  Якщо отримання було успішним, функція поверне **pdTRUE**, або **pdFALSE**, якщо черга була порожньою.

- Видалення черги та звільнення пам’яті, яку вона займала, за допомогою функції **vQueueDelete** [41]:

```
vQueueDelete(myQueue);
```

  - **myQueue** – дескриптор черги, яку потрібно видалити.

  Після видалення черги, вона більше не буде доступна, і будь-які спроби доступу до неї можуть призвести до непередбачуваних результатів.

Семафор (Semaphore)

Семафори – це механізми синхронізації, основне завдання яких – обмежити кількість завдань, які одночасно можуть отримати доступ до певного ресурсу або секції коду.

У FreeRTOS є 4 типи семафорів:

- М’ютекс (Mutex).
- Семафор з лічильником (Counting Semaphore).
- Двійковий/Бінарний семафор (Binary Semaphore).
- Рекурсивний (Recursive).

М'ютекс (Mutex)

М'ютекс (mutex, від "mutual exclusion") забезпечує взаємне виключення при доступі до спільних ресурсів. За цього гарантується, що лише одне завдання може одночасно мати доступ до ресурсу або виконувати певну критичну секцію коду.

М'ютекс діє як маркер, який використовується для захисту ресурсу. Механізм захисту базується на концепції «захоплення» і «відпускання» маркера. Доступ до спільного ресурсу надається лише одному завданню, яке захопило цей маркер.

Створюється м'ютекс за допомогою функції FreeRTOS **`xSemaphoreCreateMutex`**:

```
SemaphoreHandle_t myMutex = xSemaphoreCreateMutex();
```

Коли завдання хоче отримати доступ до ресурсу, воно намагається захопити м'ютекс за допомогою функції **`xSemaphoreTake`** [42]:

```
xSemaphoreTake(myMutex, portMAX_DELAY);
```

Якщо м'ютекс вільний (розблокований), завдання отримує доступ до ресурсу, а м'ютекс переходить у заблокований стан.

Якщо інше завдання намагається захопити м'ютекс у той час, коли він вже заблокований, це завдання блокується (переключається в стан очікування), доки м'ютекс не стане вільним.

Завдання, яке успішно захопило м'ютекс, виконує критичну секцію коду, наприклад, доступ до спільної змінної, периферійного пристрою або іншого ресурсу. Після завершення роботи з ресурсом завдання «відпускає» м'ютекс за допомогою функції **`xSemaphoreGive`** [43]:

```
xSemaphoreGive(xMutex);
```

М'ютекс повертається у розблокований стан, і будь-яке інше завдання, яке очікує на цей м'ютекс, може тепер захопити його і отримати доступ до ресурсу.

Приклад:

Програма створює два завдання, які намагаються одночасно отримати доступ до спільного ресурсу (в даному випадку – це критична секція коду, яка виводить текст у консоль). Коли одне з завдань захоплює м'ютекс, інше блокується, очікуючи поки перше не завершить виконання своєї критичної секції і не відпустить м'ютекс.

```
SemaphoreHandle_t myMutex = xSemaphoreCreateMutex(); // Створення м'ютекса

void setup()
{
  Serial.begin(115200); // Ініціалізація серійного зв'язку
  // Створення завдань
  xTaskCreate(Task1, "Task 1", 1000, NULL, 1, NULL);
  xTaskCreate(Task2, "Task 2", 1000, NULL, 1, NULL);
}
```

```
void loop()
{
  // Основний цикл залишається порожнім, оскільки все виконується в завданнях
}

void Task1(void *pvParameters)
{
  while (1)
  {
    if (xSemaphoreTake(myMutex, portMAX_DELAY) == pdTRUE)  // Захоплення м'ютекса
    {
      // Виконання критичної секції коду
      Serial.println("Завдання 1 виконується");
      vTaskDelay(500 / portTICK_PERIOD_MS); // Затримка для симуляції роботи завдання
      xSemaphoreGive(myMutex); // Відпускання м'ютекса
      // Решта "некритичного" коду може бути тут
    }
    vTaskDelay(1000 / portTICK_PERIOD_MS); // Затримка перед наступною спробою
  }
}

void Task2(void *pvParameters)
{
  while (1)
  {
    if (xSemaphoreTake(myMutex, portMAX_DELAY) == pdTRUE)  // Захоплення м'ютекса
    {
      // Виконання критичної секції коду
      Serial.println("Завдання 2 виконується");
      vTaskDelay(500 / portTICK_PERIOD_MS); // Затримка для симуляції роботи завдання
      xSemaphoreGive(myMutex); // Відпускання м'ютекса
      // Решта "некритичного" коду може бути тут
    }
    vTaskDelay(1000 / portTICK_PERIOD_MS); // Затримка перед наступною спробою
  }
}
```

Ключова відмінність м’ютекса від **бінарного семафора** заключається в тому, що, при використанні м'ютекса, тільки те завдання, яке «захопило» м'ютекс, може його «відпустити». Це забезпечує більш строгий контроль за доступом до ресурсу.

Додатоково слід зазначити, що м'ютекси підтримують поняття пріоритетної спадковості (priority inheritance), яке запобігає інверсії пріоритетів у випадках, коли завдання з низьким пріоритетом захоплює м'ютекс, потрібний завданню з вищим пріоритетом.

Семафор з лічильником (Counting Semaphore)

Семафор з лічильником (іноді просто «семафор») дозволяє контролювати декілька одночасних доступів до спільних ресурсів або критичних секцій коду. Лічильник семафора визначає максимальну кількість одночасних дозволених доступів до ресурсу. Початкове

значення встановлюється при створенні семафора. Наприклад, можливо встановити початкове значення «3», щоб одночасно тільки три завдання могли надсилати дані в мережу Інтернет, а інші – очікували доки лічильник не прийме значення менше трьох. Це значно відрізняється від м'ютекса та бінарного семафора, де доступ до спільного ресурсу можливий лише для одного завдання в будь-який момент часу.

Функції для роботи з семафором:

- **`xSemaphoreCreateCounting(`***`uxMaxCount, uxInitialCount`***`)`** [44] – використовується для створення семафора. Вона приймає два параметри: *`uxMaxCount`* – максимальне значення лічильника і *`uxInitialCount`* – початкове значення.
- **`xSemaphoreTake`** [42] – якщо лічильник більше нуля, функція успішно захоплює семафор і зменшує лічильник.
- **`xSemaphoreGive`** [43] – викликається коли завдання закінчує використання ресурсу. Віддача семафору збільшує лічильник, дозволяючи іншим завданням отримати доступ до ресурсу.

Завдання

1. Організувати передачу згенерованих завданням **`DataCreate`** чисел до завдання **`DataDisplay`**, та їх подальше послідовне відображення у серійному моніторі, з затримкою пів секунди, після кожного числа.
2. Створити 4 завдання, кожне з них має відповідати за рух символу «**-**», зліва направо, у окремому рядку дисплея. Реалізувати можливість зміни швидкості руху кожного з мінусів за допомогою відповідних потенціометрів.
3. За допомогою м'ютекса обмежити доступ до критичної секції коду, що відповідає за оновлення дисплея. Уникнути виникнення зайвих символів (артефактів) під час роботи програми.
4. За допомогою семафора з лічильником обмежити трьома кількість завдань які можуть взаємодіяти з дисплеєм. Після завершення завдання (досягнення мінусом останнього стовбця) воно повинно звільняти семафор.
5. Визначити та встановити мінімальний розмір стеку для кожного завдання (при якому програма буде нормально працювати).

Порядок виконання роботи

Крок 1. Ознайомлення з прикладами за посиланнями:

- Використання черги: Queue.cpp [4].
- Використання м`ютексу:
  - Проблема перегонів за ресурсами: Race.cpp [4].
  - Вирішення проблеми: Mutex.cpp [4].
- Використання семафору з лічильником: Semaphore.cpp [4].

Крок 2. Підключення компонентів до ESP32 відповідно до рисунку 8.2 або заміна вмісту **`diagram.json`** відповідним текстом:

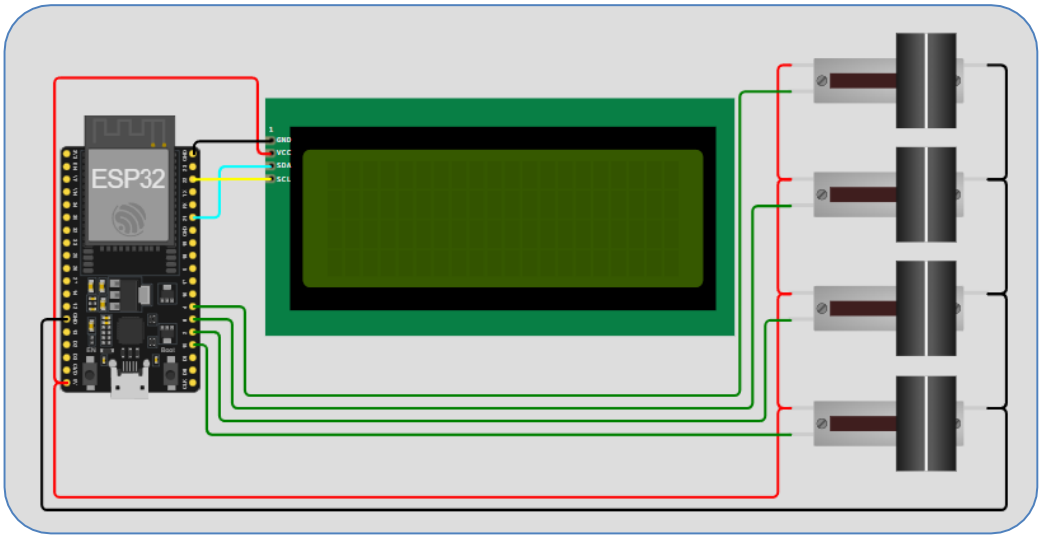


Рис. 8.2. Схема

```
{"version": 1, "author": "", "editor": "wokwi", "parts": [{"type": "board-esp32-devkit-c-v4", "id":
"esp", "top": 0, "left": -4.76, "attrs": {}}, {"type": "wokwi-lcd2004", "id": "lcd1", "top": -12.8,
"left": 149.6, "attrs": {"pins": "i2c"}}, {"type": "wokwi-slide-potentiometer", "id": "pot1", "top":
-81.4, "left": 546.2, "attrs": {"travelLength": "15"}}, {"type": "wokwi-slide-potentiometer", "id":
"pot2", "top": 5, "left": 546.2, "attrs": {"travelLength": "15"}}, {"type":
"wokwi-slide-potentiometer", "id": "pot3", "top": 91.4, "left": 546.2, "attrs": {"travelLength":
"15"}}, {"type": "wokwi-slide-potentiometer", "id": "pot4", "top": 177.8, "left": 546.2, "attrs":
{"travelLength": "15"}}], "connections": [["esp:TX", "$serialMonitor:RX", "", []], ["esp:RX",
"$serialMonitor:TX", "", []], ["lcd1:VCC", "esp:5V", "red", ["h-9.6", "v-57.5", "h-153.6", "v220.8"]],
["lcd1:GND", "esp:GND.2", "black", ["h0"]], ["lcd1:SDA", "esp:21", "cyan", ["h-38.4", "v29"]],
["lcd1:SCL", "esp:22", "yellow", ["h-28.8", "v-9.3"]], ["pot1:GND", "pot2:GND", "black", ["h13.9",
"v86.4"]], ["pot2:GND", "pot3:GND", "black", ["v0", "h13.9", "v76.8"]], ["pot3:GND", "pot4:GND",
"black", ["v0", "h13.9", "v86.4", "h-9.6"]], ["pot4:GND", "esp:GND.1", "black", ["h13.9", "v76.8",
"h-729.6", "v-144"]], ["pot1:VCC", "pot2:VCC", "red", ["h-9.6", "v86.4"]], ["pot2:VCC", "pot3:VCC",
"red", ["h-9.6", "v9.6"]], ["pot3:VCC", "pot4:VCC", "red", ["h-9.6", "v9.6"]], ["pot4:VCC", "esp:5V",
"red", ["h-9.6", "v67.2", "h-547.2", "v-86.4"]], ["pot4:SIG", "esp:15", "green", ["h-441.6", "v-68"]],
["pot3:SIG", "esp:2", "green", ["h-19.2", "v76", "h-412.8", "v-57.6"]], ["pot2:SIG", "esp:0", "green",
["h-28.8", "v152.8", "h-393.6", "v-57.6"]], ["pot1:SIG", "esp:4", "green", ["h-38.4", "v229.6",
"h-374.4", "v-67.2"]]], "dependencies": {}}
```

Крок 3. Додавання коду з файлу Example8.cpp [4] редактор коду та ознайомлення з прикладом. Перед запуском програми встановіть бібліотеку LiquidCrystal_I2C.

Крок 4. Реалізація передачі даних між завданнями DataCreate та DataDisplay:

- Створіть чергу для передачі даних за допомогою **`xQueueCreate`**.
- Налаштуйте відправку даних в чергу за допомогою **`xQueueSend`** у завданні **`DataCreate`**.
- Налаштуйте обробку даних з черги за допомогою **`xQueueReceive`** у завданні **`DataDisplay`**.

Крок 5. Реалізація пересування символів у кожному рядку дисплея.

- Створіть чотири завдання, кожне з яких, в своєму рядку дисплея, повинно рухати символ «-» зліва направо з використанням функцій бібліотеки LiquidCrystal_I2C.

- Реалізуйте вплив положення кожного з потенціомертів на швидкість руху «-» в відповідних рядках.
- Запустіть програму та, при різних положеннях потенціометрів, перевірте чи з’являються на дисплеї артефакти під час одночасного доступу декількох завдань до нього.

Крок 6. Обмеження доступу до ресурсу (дисплея):

- За допомогою `xSemaphoreCreateMutex` створіть м'ютекс для обмеження доступу до дисплея.
- В кожному завданні, перед оновленням дисплея, захоплюйте м'ютекс за допомогою `xSemaphoreTake`.
- Одразу після критичної секції коду звільняйте м'ютекс за допомогою `xSemaphoreGive`.

Крок 7. Управління переміщенням символів:

- Створіть семафор з лічильником за допомогою `xSemaphoreCreateCounting`.
- Встановіть максимальне значення лічильника «3».
- Змініть функцію завдання таким чином, щоб воно виконувалось лише після взятя семафору.
- Після виконання кожного завдання, тобто, після досягнення кожним мінусом кінця дисплея, звільняйте семафор для зменшення значення лічильника.
- Додайте невелику затримку (наприклад 50 мс), щоб дати можливість іншому завданню взяти семафор.

Крок 8. Налаштування розміру стеку:

- Визначте та встановіть мінімальний розміру стеку.
- Перевірте, чи виконуються всі завдання правильно.

Оформлення звіту та порядок його подання

1. Увійдіть в дистанційний курс Moodle з власного аккаунту.
2. Відкрийте активність присвячену даній лабораторній роботі.
3. Натисніть кнопку «Здати роботу».
4. У текстове поле завантажте:
    - посилання на створений проект в симуляторі Wokwi з виконаними завданнями;
    - висновки до лабораторної роботи.
5. Натисніть кнопку «Зберегти».

Контрольні запитання

1. Що таке спільні ресурси в контексті RTOS? Наведіть приклади спільних ресурсів.
2. Чому важливо забезпечувати безпечний доступ до спільних ресурсів?
3. Що таке черги (queues) і як вони допомагають при взаємодії між завданнями?
4. Які основні особливості таких механізмів синхронізації, як семафори?
5. Яка ключові відмінності між м’ютексом, бінарим та лічильним семафорами?

ПЕРЕЛІК ПОСИЛАНЬ

1. World's most advanced ESP32 simulator. *Wokwi*. URL: https://wokwi.com/ (дата звернення: 27.02.2024).
2. Welcome to Wokwi! *Wokwi Docs*. URL: https://docs.wokwi.com/ (дата звернення: 27.02.2024).
3. Language Reference. *Arduino*. URL: https://www.arduino.cc/reference/en/ (дата звернення: 27.02.2024).
4. Посилання. *Microsoft Word Online*. URL: https://1drv.ms/w/c/39f68fc85737ad2f/EWHJrQgQbPBBtS4oLl3VQi4Bwyua31co0rbx9pmNJaPoAA (дата звернення: 27.02.2024).
5. Digitalwrite. *Arduino*. URL: https://www.arduino.cc/reference/en/language/functions/digital-io/digitalwrite/ (дата звернення: 27.02.2024).
6. Wokwi-7segment Reference. *Wokwi Docs*. URL: https://docs.wokwi.com/parts/wokwi-7segment (дата звернення: 27.02.2024).
7. Serial. *Arduino*. URL: https://www.arduino.cc/reference/en/language/functions/communication/serial/ (дата звернення: 27.02.2024).
8. Universal Asynchronous Receiver-Transmitter (UART). *Arduino Documentation*. URL: https://docs.arduino.cc/learn/communication/uart/#examples (дата звернення: 27.02.2024).
9. Arduino & Serial Peripheral Interface (SPI). *Arduino Documentation*. URL: https://docs.arduino.cc/learn/communication/spi/ (дата звернення: 27.02.2024).
10. Serial Peripheral Interface. *Wikipedia*. URL: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface (дата звернення: 27.02.2024).
11. Barometric Pressure Sensor (SPI). *Arduino Documentation*. URL: https://docs.arduino.cc/tutorials/communication/BarometricPressureSensor/ (дата звернення: 27.02.2024).
12. Digital Potentiometer Control. *Arduino Documentation*. URL: https://docs.arduino.cc/tutorials/communication/DigitalPotControl/ (дата звернення: 27.02.2024).
13. Wire. *Arduino*. URL: https://www.arduino.cc/reference/en/language/functions/communication/wire/ (дата звернення: 27.02.2024).
14. MPU-6000 and MPU-6050 Register Map and Descriptions Revision 4.2. *InvenSense*. URL: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf (дата звернення: 27.02.2024).
15. LiquidCrystal I2C. *Arduino*. URL: https://www.arduino.cc/reference/en/libraries/liquidcrystal-i2c/ (дата звернення: 27.02.2024).
16. WiFi. *Arduino*. URL: https://www.arduino.cc/reference/en/libraries/wifi (дата звернення: 27.02.2024).
17. WiFi - WiFi.status(). *Arduino*. URL: https://www.arduino.cc/reference/en/libraries/wifi/wifi.status/ (дата звернення: 27.02.2024).
18. NTPClient. *Arduino*. URL: https://www.arduino.cc/reference/en/libraries/ntpclient/ (дата звернення: 27.02.2024).
19. JSON. *Wikipedia*. URL: https://uk.wikipedia.org/wiki/JSON (дата звернення: 27.02.2024).
20. ArduinoJson: Efficient JSON serialization for embedded C++. *ArduinoJson*. URL: https://arduinojson.org/ (дата звернення: 27.02.2024).

21. Substring. *Arduino Documentation*. URL: https://www.arduino.cc/reference/en/language/variables/data-types/string/functions/substring (дата звернення: 27.02.2024).
22. Weather Forecast API. *Open-Meteo*. URL: https://open-meteo.com/en/docs/ (дата звернення: 27.02.2024).
23. Free Public MQTT Broker. *HiveMQ*. URL: https://www.hivemq.com/mqtt/public-mqtt-broker/ (дата звернення: 27.02.2024).
24. Interfacing DHT11 and DHT22 Sensors with Arduino. *Last Minute ENGINEERS*. URL: https://lastminuteengineers.com/dht11-dht22-arduino-tutorial/ (дата звернення: 27.02.2024).
25. PubSubClient. *Arduino*. URL: https://www.arduino.cc/reference/en/libraries/pubsubclient/ (дата звернення: 27.02.2024).
26. API Documentation. *Arduino Client for MQTT*. URL: https://pubsubclient.knolleary.net/api (дата звернення: 27.02.2024).
27. MQTT Websocket client. *HiveMQ*. URL: https://www.hivemq.com/demos/websocket-client/ (дата звернення: 27.02.2024).
28. ThingsBoard Open-source IoT Platform. *ThingsBoard*. URL: https://thingsboard.io/ (дата звернення: 27.02.2024).
29. Getting Started with ThingsBoard Cloud. *ThingsBoard Cloud*. URL: https://thingsboard.io/docs/paas/getting-started-guides/helloworld/ (дата звернення: 27.02.2024).
30. Wokwi-servo Reference. *Wokwi Docs*. URL: https://docs.wokwi.com/parts/wokwi-servo (дата звернення: 27.02.2024).
31. ESP32Servo: Servo Class Reference. *ESP32Servo Documentation*. URL: https://madhephaestus.github.io/ESP32Servo/classServo.html (дата звернення: 27.02.2024).
32. FreeRTOS Documentation. *FreeRTOS*. URL: https://www.freertos.org/Documentation/00-Overview (дата звернення: 27.02.2024).
33. FreeRTOS - ESP32. *ESP-IDF Programming Guide v4.3 documentation*. URL: https://docs.espressif.com/projects/esp-idf/en/v4.3/esp32/api-reference/system/freertos.html (дата звернення: 27.02.2024).
34. Task Creation. *FreeRTOS*. URL: https://www.freertos.org/Documentation/02-Kernel/04-API-references/01-Task-creation/00-TaskHandle (дата звернення: 27.02.2024).
35. Task control – vTaskSuspend. *FreeRTOS*. URL: https://www.freertos.org/Documentation/02-Kernel/04-API-references/02-Task-control/06-vTaskSuspend (дата звернення: 27.02.2024).
36. Task control – vTaskResume. *FreeRTOS*. URL: https://www.freertos.org/Documentation/02-Kernel/04-API-references/02-Task-control/07-vTaskResume (дата звернення: 27.02.2024).
37. Queue Management. *FreeRTOS*. URL: https://www.freertos.org/Documentation/02-Kernel/04-API-references/06-Queues/00-QueueManagement (дата звернення: 27.02.2024).
38. Queue Management – xQueueCreate. *FreeRTOS*. URL: https://www.freertos.org/Documentation/02-Kernel/04-API-references/06-Queues/01-xQueueCreate (дата звернення: 27.02.2024).
39. Queue Management – xQueueSend. *FreeRTOS*. URL: https://www.freertos.org/Documentation/02-Kernel/04-API-references/06-Queues/03-xQueueSend (дата звернення: 27.02.2024).
40. Queue Management – xQueueReceive. *FreeRTOS*. URL: https://www.freertos.org/Documentation/02-Kernel/04-API-references/06-Queues/09-xQueueReceive (дата звернення: 27.02.2024).

41. Queue Management – vQueueDelete. *FreeRTOS*. URL: https://freertos.org/Documentation/02-Kernel/04-API-references/06-Queues/00-QueueManagement#vqueuedelete (дата звернення: 27.02.2024).
42. Semaphore and Mutexes – xSemaphoreTake. *FreeRTOS*. URL: https://www.freertos.org/Documentation/02-Kernel/04-API-references/10-Semaphore-and-Mutexes/12-xSemaphoreTake (дата звернення: 27.02.2024).
43. Semaphore and Mutexes – xSemaphoreGive. *FreeRTOS*. URL: https://www.freertos.org/Documentation/02-Kernel/04-API-references/10-Semaphore-and-Mutexes/15-xSemaphoreGive (дата звернення: 27.02.2024).
44. Semaphore and Mutexes – xSemaphoreCreateCounting. *FreeRTOS*. URL: https://www.freertos.org/Documentation/02-Kernel/04-API-references/10-Semaphore-and-Mutexes/04-xSemaphoreCreateCounting (дата звернення: 27.02.2024).