КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ РАДІОФІЗИКИ, ЕЛЕКТРОНІКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ

Кафедра радіотехніки та радіоелектронних систем

«На правах рукопису»
Робота допущена до захисту в ЕК
рішенням кафедри радіотехніки та радіоелектронних
систем від _____ року, протокол № ____.
Завідувач кафедри доктор фіз.-мат. наук, професор
_____Ігор АНІСІМОВ

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА
на тему:
«СИСТЕМА ОПОВІЩЕННЯ ПОВІТРЯНОЇ ТРИВОГИ НА БАЗІ МІКРОКОНТРОЛЕРА ESP32»

Виконав:

студент 4-го курсу
денної форми навчання
спеціальності 172 - Телекомунікації та радіотехніка
ОПП «Інформаційна безпека телекомунікаційних систем і мереж»
Шестов Михайло Петрович _____

Науковий керівник:

канд. фіз.-мат. наук, асистент
Богданов Роман Вікторович - _____

Рецензент:

канд. фіз.-мат. наук, доцент
Баужа Олександр Стасісович -_____
Засвідчую, що у цій бакалаврській роботі
немає запозичень з праць інших авторів
без відповідних посилань
Студент _____

Київ – 2025

РЕФЕРАТ

Дипломна робота: 36 сторінки, 13 рисунків, 12 використаних джерел.

Ключові слова: повітряна тривога, система оповіщення, ESP32, мікроконтролер, Інтернет речей, IoT, API, DFPlayer Mini, автономний пристрій, апаратно-програмний комплекс.

Об'єкт розроблення: процес отримання, обробки та сповіщення про повітряну небезпеку за допомогою автономного апаратно-програмного комплексу.

Мета роботи — розробити та дослідити апаратно-програмний комплекс для оповіщення про повітряну тривогу на базі мікроконтролера ESP32, який забезпечує високу надійність, автономність, швидкість реакції та не залежить від наявності у користувача смартфона, вирішуючи проблему своєчасного інформування вразливих верств населення в умовах нестабільного мобільного зв'язку та доступу до Інтернету.

Стислий опис роботи.

У роботі проведено аналіз існуючих традиційних та цифрових систем оповіщення населення про повітряну небезпеку, виявлено їхні переваги та недоліки, зокрема залежність від наявності у користувача смартфона та стабільного інтернет-з'єднання. На основі аналізу сформульовано технічні вимоги до розроблюваного пристрою.

Обґрунтовано вибір мікроконтролера ESP32 як центрального елемента системи завдяки його високій продуктивності, інтегрованому модулю Wi-Fi та низькій вартості. Також підібрано інші апаратні компоненти, такі як звуковий модуль DFPlayer Mini, динамік та світлодіоди для забезпечення візуальної та звукової індикації.

Розроблено принципову електричну схему пристрою та детальний алгоритм роботи програмного забезпечення. Програмна логіка, реалізована мовою C++ в середовищі Arduino IDE, забезпечує підключення до мережі Wi-Fi, періодичне надсилання HTTP-запитів до публічного API Sirens.ia.ua , обробку отриманих

JSON-даних та керування індикацією залежно від статусу тривоги в обраному регіоні.

В результаті було створено та протестовано діючий прототип автономного оповіщувача, який успішно виконує поставлені завдання. Розробка має практичне значення, оскільки її низька собівартість та простота конструкції дозволяють організувати масове виробництво для забезпечення потреб населення.

ЗМІСТ

ВСТУП

Актуальність теми. В умовах повномасштабної російської агресії проти України, своєчасне та надійне оповіщення населення про повітряну небезпеку є питанням життя мільйонів громадян. Існуючі системи оповіщення, хоча й є ефективними, значною мірою покладаються на наявність стабільного мобільного зв'язку та доступу до мережі Інтернет на кінцевих пристроях користувачів, таких як смартфони. Проте, в умовах бойових дій або масованих атак на енергетичну інфраструктуру, ці канали зв'язку можуть бути нестабільними або взагалі відсутніми. Це створює ризик того, що частина населення не отримає вчасного попередження.

Крім того, для людей похилого віку, людей з вадами зору чи слуху, або тих, хто не користується смартфонами, мобільні додатки не є зручним чи доступним засобом отримання інформації. Отже, виникає гостра потреба у створенні автономного, надійного, недорогого та простого у використанні пристрою, здатного отримувати дані про повітряні тривоги та сповіщати про них незалежно від наявності у користувача смартфона. Використання сучасної елементної бази, зокрема мікроконтролерів серії ESP32, дозволяє вирішити цю задачу з високою ефективністю та мінімальними витратами.

Мета дослідження: Розробити та дослідити апаратно-програмний комплекс для оповіщення про повітряну тривогу на базі мікроконтролера ESP32, який забезпечує високу надійність, автономність та швидкість реакції.

Завдання дослідження:

1. Провести аналіз існуючих систем оповіщення, виявити їхні переваги та недоліки.
2. Сформулювати технічні вимоги до автономного оповіщувача.
3. Обґрунтувати вибір мікроконтролера ESP32 та інших апаратних і програмних компонентів.
4. Розробити принципову електричну схему та алгоритм роботи пристрою.

5. Реалізувати апаратну та програмну частини системи, розрахувати її вартість.

6. Створити та налагодити діючий прототип пристрою.

7. Запропонувати шляхи для подальшого вдосконалення розробки.

Об'єкт дослідження: Процес отримання та обробки даних про повітряну небезпеку в локальних системах оповіщення.

Предмет дослідження: Апаратні та програмні засоби реалізації автономного пристрою оповіщення про повітряну тривогу на базі мікроконтролера ESP32.

Практичне значення отриманих результатів. Розроблений пристрій може використовуватися в побутових умовах, у невеликих офісах, соціальних закладах та освітніх установах як дублююча або основна система оповіщення. Низька собівартість та простота конструкції дозволяють організувати масове виробництво таких оповіщувачів для забезпечення потреб населення, зокрема вразливих категорій. Програмний код та схемотехнічні рішення можуть бути використані як основа для подальших розробок у сфері систем безпеки та "Інтернету речей" (IoT).

РОЗДІЛ 1. СИСТЕМИ ОПОВІЩЕННЯ НАСЕЛЕННЯ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Огляд існуючих систем оповіщення населення

Система оповіщення – це комплекс організаційно-технічних заходів, пристроїв та каналів зв’язку, призначених для своєчасного донесення сигналів і повідомлень про загрози до органів влади та населення. Сигнал повітряної тривоги вмикається, коли військові виявляють загрозу з повітря. За це відповідають підрозділи радіолокації, які використовують РЛС-станції (радіолокаційні станції) та інші розвідувальні засоби, зокрема супутники. Важливо, що загрози фіксуються не тільки на території нашої держави, а й за її межами. Наприклад, якщо розвідка виявила зліт ворожих літаків або запуск ракет, населення буде негайно попереджено.

Традиційні системи включають сирени, радіо- та телебачення а також текстові повідомлення (SMS).

Сирени: Найпоширеніший метод. Перевага – миттєве охоплення великої аудиторії. Недоліки – обмежене покриття, неможливість передати детальну інформацію.

Радіо та телебачення: Дозволяють передати детальну інформацію, але залежать від наявності електроенергії та того, чи увімкнені пристрої у громадян.

SMS-повідомлення: Цей метод має високу ймовірність досягнення конкретного адресата та можливість передати конкретні інструкції. Проте затримки в доставці через навантаження мережі та залежність від мобільного покриття і функціонування операторів є значними недоліками.

З початком повномасштабного вторгнення система була значно доповнена цифровими каналами. Сучасні технології дозволяють створювати більш ефективні та надійні системи оповіщення.

Мобільний додаток "Повітряна тривога": Розроблений за підтримки Мінцифри. Надсилає push-сповіщення. Є дуже ефективним, але залежить від наявності смартфона, доступу до Інтернету та справності сервісів Google/Apple.

Інтернет-ресурси та соціальні мережі: Офіційні Telegram-канали, онлайн-карти тривог агрегують дані і часто надають API для розробників.

Комерційні пристрої: На ринку присутні готові рішення у вигляді інтерактивних карт або сповіщувачів, проте їхня вартість є значним недоліком для масового споживача.

Сучасні технології, такі як мобільні застосунки та інтерактивні мапи, демонструють високу ефективність у швидкості та детальності передачі інформації. Вони дозволяють оперативно доносити дані, що є критично важливим у багатьох сферах. Однак, попри значні переваги, ці технології стикаються з певними технічними та інфраструктурними викликами. Швидкість передачі інформації через мобільні застосунки та соціальні мережі, хоч і висока, може бути обмежена наявністю стабільного інтернет-з'єднання та доступністю пристроїв. У зонах зі слабким або відсутнім покриттям зв'язку ефективність цих засобів значно знижується, а не всі верстви населення мають сучасні смартфони або доступ до інтернету, що може створювати інформаційний розрив. Таким чином, для повноцінного використання потенціалу мобільних застосунків та інтерактивних мап необхідно вирішувати питання, пов'язані з доступом до інтернету та пристроїв.

1.2. Аналіз вимог до пристрою оповіщення

На основі виявлених проблем, сформулюємо вимоги до розроблюваного оповіщувача.

Функціональні вимоги:

- Підключення до мережі Wi-Fi для отримання даних: Щоб отримувати інформацію про повітряні тривоги з віддалених серверів, система має підключатися до інтернету через Wi-Fi. Для цього буде використано мікроконтролер ESP32, який має вбудований Wi-Fi модуль. Він забезпечить стабільне та швидке з'єднання, дозволяючи системі оперативно отримувати нові дані та оновлювати інформацію про тривоги.
- Періодичне опитування надійного джерела даних (API) про статус тривог: Інформація про повітряні тривоги має оновлюватися автоматично, щойно надходять нові дані. Для цього необхідно розробити систему, яка буде постійно

моніторити інформацію з віддалених серверів. Такий підхід гарантує актуальність даних та своєчасне інформування користувачів про будь-які зміни в ситуації.

- Можливість налаштування цільового регіону для моніторингу. Можливість самостійно обирати область тривоги та подавати світовий сигнал в залежності від стану в області.

Нефункціональні вимоги:

- Надійність: система повинна мати стабільну роботу 24/7 та автоматичне перепідключення до Wi-Fi та відновлення роботи після збоїв. Тобто якщо стався збій, то система повинна запам`ятати останній стан в якому перебувала.
- Швидкодія: Мінімальна затримка між появою інформації в API та реакцією пристрою .
- Енергоефективність: Сам мікроконтроллер ESP32 має низьку енергоспоживання, що дозволяє зменшити затрати енергій, проте також необхідна можливість живлення від резервного джерела.

1.3. Огляд апаратних компонентів

За основу було взято мікроконтролер ESP32(рис.1.1) через його високу продуктивність. Його двоядерний процесор дозволяє ефективно обробляти великі обсяги даних та виконувати одночасно багато завдань[1]. Крім того, ESP32 має більше GPIO пінів, що спрощує підключення численних світлодіодів та інших компонентів.

Хоча розглядалися Arduino Uno та Nano, їх відсутність вбудованого Wi-Fi модуля стала критичним недоліком. Для реалізації Wi-Fi функціональності з Arduino довелося б використовувати додатковий модуль, такий як ESP8266, що ускладнило б конструкцію та збільшило загальну вартість проєкту.

Рисунок 1.1 Зовнішній вигляд ESP32

За візуальний сигнал будуть відповідати світлодіоди 5мм(рис. 1.2).

Рисунок 1.2 5мм світлодіоди

Сигнал звуку керується через плату DFPlayerMini (рис.1.3), яка керує.

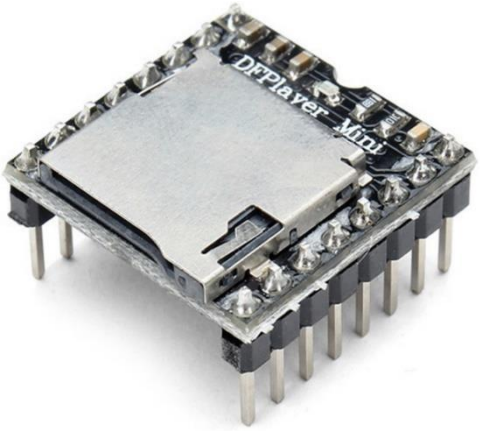

Рисунок 1.3 плата DFPlayer mini

Звук виводиться через динамік(рис.1.4) потужністю 0,5 Вт з опором 8 Ом.

Рисунок 1.4 Динаміки для виводу звуку

Для обмеження струму до світлодіодів у системі буде використано резистор стандартний вивідний MFR-25FRF52-330R Yageo (рис. 1.5).

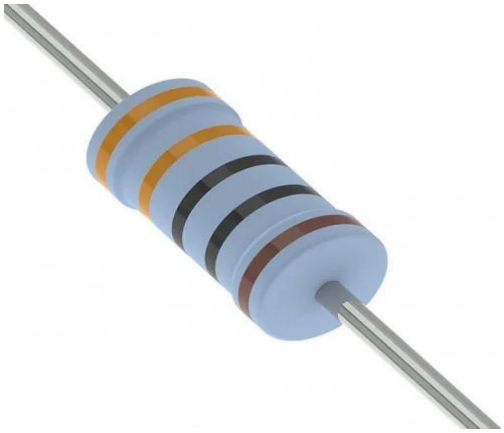

Рисунок 1.5 Резистор MFR-25FRF52-330R Yageo

Для стабілізації живлення ESP32 підібрано конденсатор(рис.1.6) EEU-FC1C101.

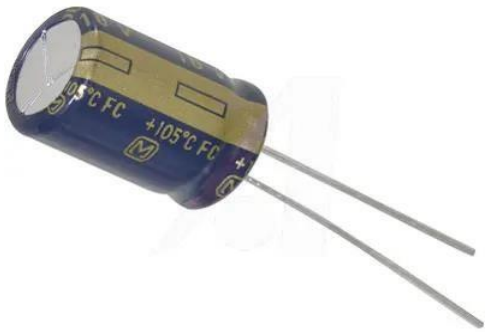

Рисунок 1.6 Конденсатор EEU-FC1C101

1.4. Огляд програмних компонентів

Середовище розробки: Для написання та завантаження коду в мікроконтролер.

- Arduino IDE: Найпопулярніший вибір для початківців та швидкого прототипування. Має величезну кількість бібліотек та прикладів. Використовує спрощену версію C++.
- PlatformIO: Більш професійне середовище, інтегрується з VS Code. Надає кращі інструменти для керування проектами та залежностями.
- MicroPython: Інтерпретатор мови Python для мікроконтролерів. Дозволяє швидше розробляти, але може поступатися у продуктивності C++.

Висновок до розділу 1

Проведений аналіз показав гостру потребу в створенні автономного, доступного та надійного пристрою для оповіщення про повітряні тривоги. Сформульовані функціональні та нефункціональні вимоги визначають ключові характеристики майбутнього пристрою. Вибір мікроконтролера ESP32 як основної платформи, разом із простими компонентами індикації та середовищем розробки Arduino IDE, створює міцну основу для успішної реалізації проекту.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА МОДЕЛЮВАННЯ ПРИСТРОЮ

2.1. Обґрунтування обраних технологій та компонентів

Мікроконтролер ESP32: Обраний як центральний елемент системи завдяки оптимальному поєднанню характеристик:

- Продуктивність: Двоядерний процесор Tensilica Xtensa LX6 (до 240 МГц) та 520 КБ SRAM є більш ніж достатніми для обробки HTTP-запитів, парсингу JSON та керування периферією.[1]
- Інтегрований Wi-Fi: Усуває потребу в зовнішніх модулях, що спрощує схему та знижує вартість.[1]
- Екосистема: Величезна підтримка спільнотою, наявність готових бібліотек для Arduino IDE.
- Ціна: Дуже конкурентна вартість робить його ідеальним для бюджетних проектів.

Середовище розробки Arduino IDE: Вибрано через простоту, доступність та широку базу знань. Це дозволяє швидко розгорнути середовище та почати програмування без складних налаштувань, що є ідеальним для швидкого прототипування.

API від sirens.in.ua: Це публічне API є надійним агрегатором даних про тривоги з офіційних джерел. Воно надає структуровану інформацію у форматі JSON, що легко обробляється мікроконтролером.

2.2. Розробка принципової електричної схеми пристрою

Схема пристрою є максимально простою для забезпечення надійності та низької вартості.

Перелік компонентів:

1. Плата ESP32.
2. Звукова плата DFPlayer mini.
3. Червоний світлодіод (5 мм).
4. Зелений світлодіод (5 мм).
5. Резистор R1: 330 Ом (для червоного LED).
6. Резистор R2: 330 Ом (для зеленого LED).

7. Конденсатор 100мкФ.

Принципова схема(рис.2.1) (створена у Fritzing):

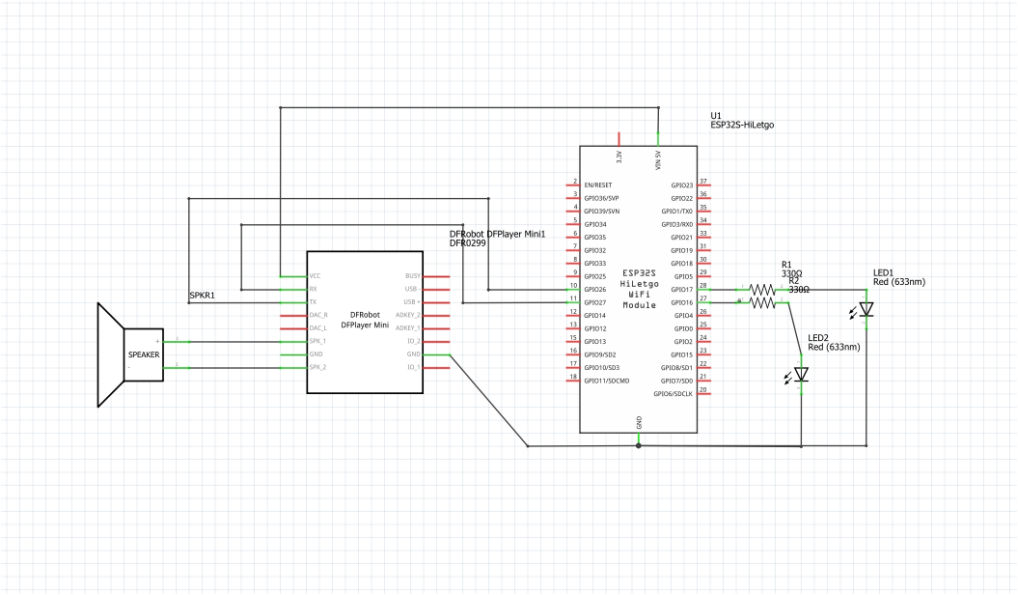

Рисунок 2.1 Принципова схема

До плати ESP32 підключено модуль DFPlayer Mini, динамік та два світлодіоди з резисторами. DFPlayer Mini живиться від пінів 5V та GND ESP32, забезпечуючи стабільну роботу. Для обміну даними DFPlayer Mini підключено до ESP32 через UART2: пін TX DFPlayer (виводить дані) з'єднано з GPIO26 ESP32, а RX DFPlayer (приймає дані) — з GPIO27 ESP32. Аудіосигнал подається безпосередньо з виводів SPK1 і SPK2 DFPlayer Mini на динамік, який відтворює звук. Два світлодіоди (LED1 і LED2) підключено до цифрових виходів ESP32. Перший світлодіод з'єднано через резистор 330 Ом до GPIO17, другий — через аналогічний резистор до GPIO16. Катоди обох світлодіодів підключено до загальної землі (GND). Це дозволяє керувати кожним світлодіодом окремо через програму в ESP32.

Усі модулі та компоненти мають спільну землю (GND), що забезпечує стабільну та узгоджену роботу системи**.**

2.3. Алгоритм роботи програмного забезпечення

Програмне забезпечення працює за циклічним алгоритмом, який можна представити у вигляді блок-схеми(рис.2.2).

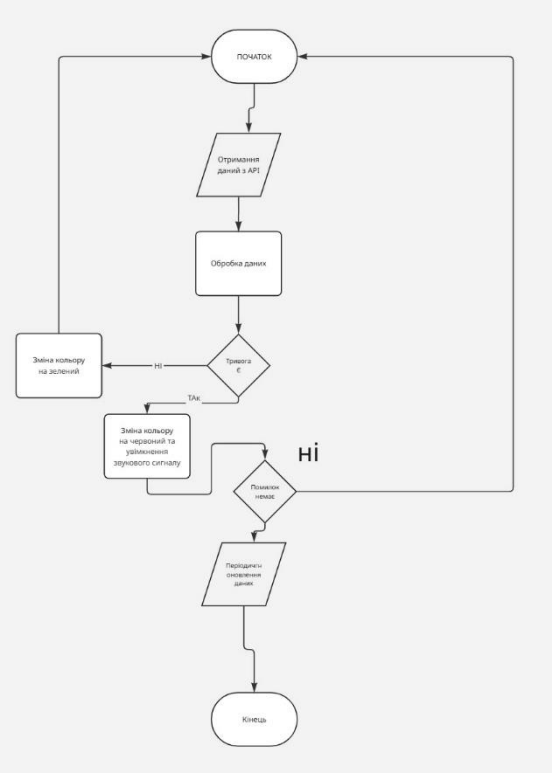

Рисунок 2.2 Блок-схема

1. Ініціалізація системи

На початковому етапі програмне забезпечення здійснює ініціалізацію роботи мікроконтролера ESP32. Це включає налаштування всіх необхідних периферійних пристроїв. Піни ESP32 конфігуруються для керування світлодіодами та забезпечення взаємодії із платою dfplayer mini. Далі мікроконтролер під’єднується до Wi-Fi мережі для доступу до Інтернету. Цей етап є ключовим для отримання актуальних даних про стан тривог через API. Додатково встановлюються бібліотеки звуковим модулем та Wi-Fi.

2. Отримання даних про тривоги

Основне завдання програми полягає у зборі актуальної інформації про стан тривог в Україні. Для цього періодично відправляються HTTP-запити до API за адресою sirens.in.ua. API надсилає відповідь у форматі JSON, яка містить деталі про поточний стан тривог у різних регіонах країни.

3. Обробка отриманих даних

Після отримання відповіді JSON-дані аналізуються для виявлення інформації щодо кожного регіону. Програмне забезпечення перевіряє, чи активна тривога на певній території, розбираючи необхідні поля у структурі даних JSON.

4. Періодичне оновлення даних

Програма налаштована на періодичну перевірку стану тривог через фіксовані інтервали (кожні 30 секунд). Кожен новий запит до API дозволяє отримати актуалізовану інформацію. На основі цих даних оновлюється стан як світлодіодів, так і системи звукового оповіщення для забезпечення точного відображення поточної ситуації.

РОЗДІЛ 3. АПАРАТНО-ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

У цьому розділі розглянемо апаратно-програмну реалізацію, що відповідає вимогам компонентів, базуючись на попередніх розділах.

3.1 Апаратна частина

Головним елементом нашої схеми є мікроконтролер ESP32(рис.3.1). До якого підключатимемо всі інші елементи схеми.

Рисунок 3.1 ESP32

З’єднання буде відбуватися за допомогою кабелів типу Dupont(рис.3.2). Ці кабелі предназначені для прототипування електронних схем та забеспечують швидке з’єднання без використання паяльника.

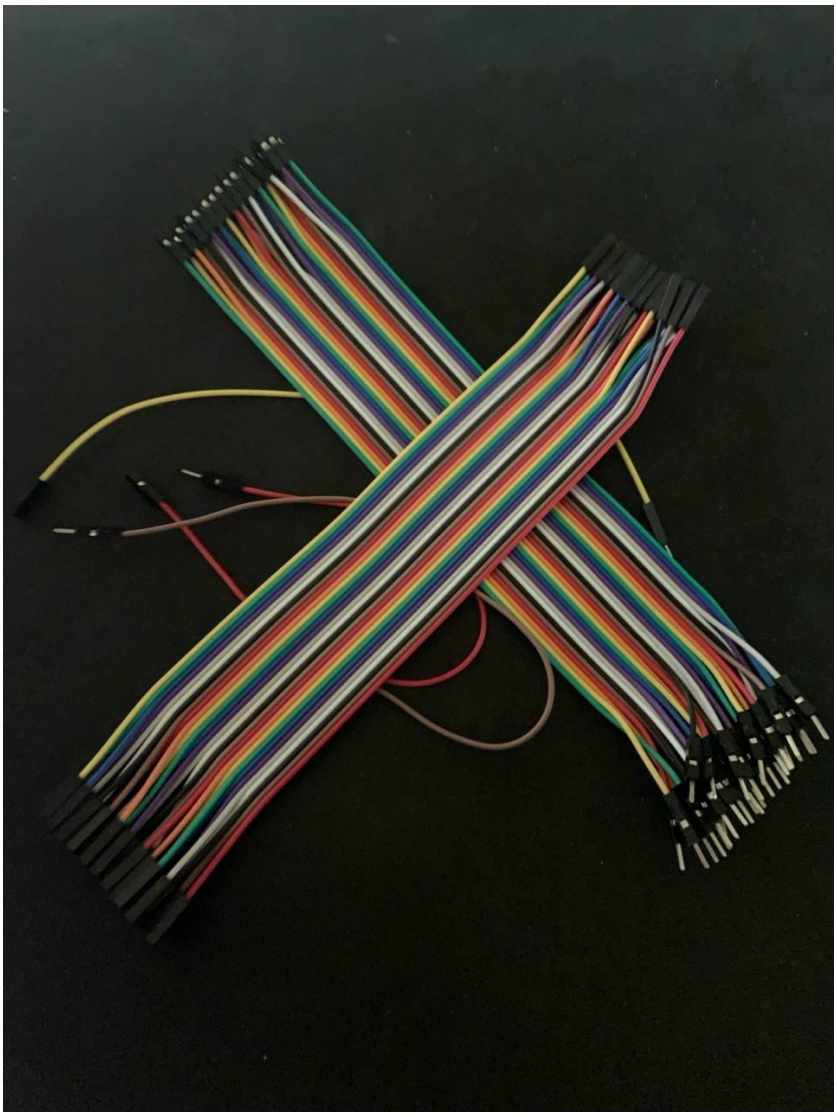

Рисунок 3.2 Кабелі Dupont

Живлення світлодіодів відбувається від пінів які подають логічну напругу 3.3В. Саме тому обрано резистори 330 ОМ оскільки ESP32 GPIO витримують до 12 мА на пін. Світлодіод зазвичай споживає ~10 мА, тому резистор 330 Ом цілком підходить.

Це безпечний струм для ESP32 та досить яскраве світіння для червоних та зелених світлодіодів які ми використовуємо (2.0В типова напруга включення) .

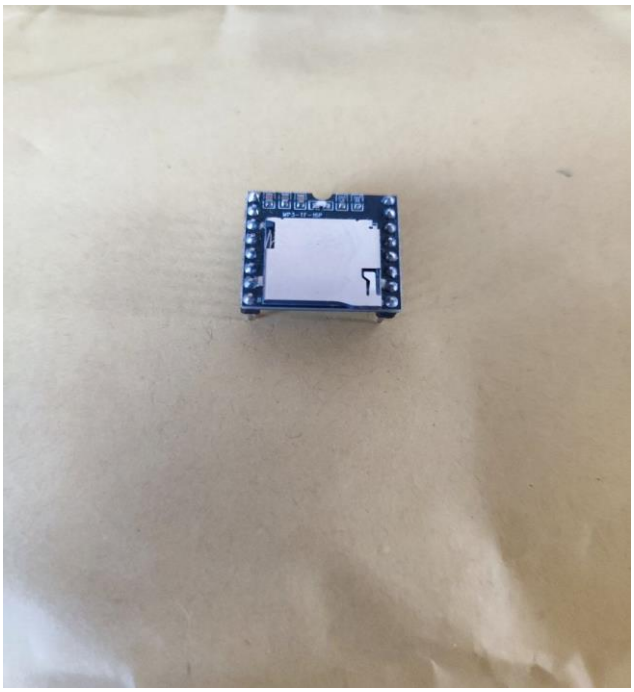

Рисунок 3.3 Звуковий модуль DFPlayer mini

Підключення звукового модуля DFPlayer mini(рис.3.3) відбувається за допомогою напруги 5В, щоб він міг стабільно працювати та підсилювати звук безпосередньо на динамік. Дані до модуля передаються через виходи GPIO26 та GPIO27 на ESP32 та TX і RX на звуковому модулі відповідно. Динамік підключений напряму до виходів на DFPlayer mini.

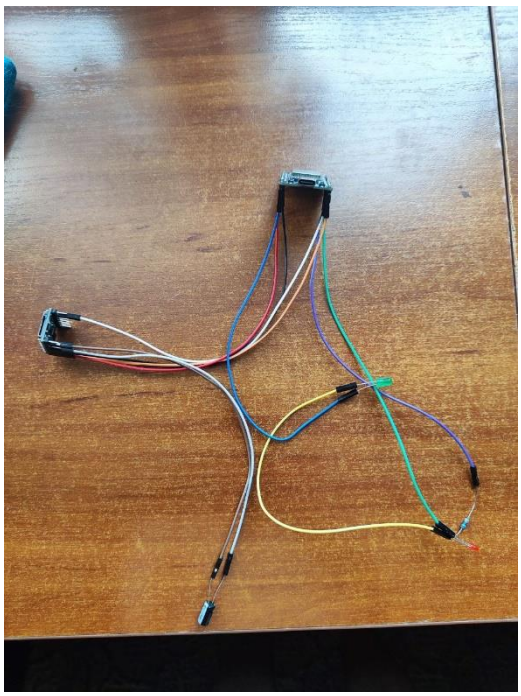

Рисунок 3.4 Фінальний результат зібраної схеми

3.2 Отримання даних з API

Перед тим як розпочати роботу з програмною частиною пристрою потрібно спочатку отримати токен для того щоб працювати з API.

API (Application Programming Interface) – це набір правил і механізмів, який дозволяє різним програмам взаємодіяти одна з одною. API виступає посередником, який дозволяє одній програмі запитувати дані або функції в іншої програми або сервісу та отримувати необхідні результати. Основними поняттями API є ендпоінти, запити, відповіді та токени доступу.

Ендпоінт – це конкретна URL-адреса для запиту до API.

Запит – це HTTP-запит (GET, POST, PUT, DELETE), який відправляється до API для отримання або надсилання даних.

Відповідь – дані, які повертаються від API у відповідь на запит, зазвичай у форматі JSON або XML

Токен доступу – унікальний ключ для автентифікації і авторизації запитів до API

У цій роботі використано API з відкритого доступу sirens.in.ua/api/v1/


```
{
  "Mykolayiv": null,
  "Chernihiv": null,
  "Rivne": null,
  "Chernivtsi": null,
  "Ivano-Frankivs'k": null,
  "Khmel'nyts'kyy": null,
  "L'viv": null,
  "Ternopil'": null,
  "Transcarpathia": null,
  "Volyn": null,
  "Cherkasy": null,
  "Kirovohrad": null,
  "Kyiv": null,
  "Odessa": null,
  "Vinnytsya": null,
  "Zhytomyr": null,
  "Sumy": "full",
  "Dnipropetrovs'k": null,
  "Donets'k": null,
  "Kharkiv": "full",
  "Poltava": null,
  "Zaporizhzhya": null,
  "Kyiv City": null,
  "Kherson": null,
  "Luhans'k": "full",
  "Sevastopol": "no_data",
  "Crimea": "no_data"
}
```


Рисунок 3.5 масив даних для роботи коду

3.3 Програмна частина

Код для програмування оповіщувача повітряної тривоги написаний мовою програмування C++ з використанням Arduino IDE.

Arduino Ide використовує спрощену версію C++ . Використовуючи яку можна написати програми для отриманні та обробки HTTP запити до API.

Опис програмної частини розробки(повноцінний код доданий у додатку А):

```
#include "Arduino.h"
#include "DFRobotDFPlayerMini.h"
#include <WiFi.h>
#include <HTTPClient.h>
#include <WiFiClientSecure.h>
#include <ArduinoJson.h>
```

На початку коду підключаємо необхідні для роботи бібліотеки з роботою wi-fi, Json-даними, HTTP-запитами та плеєром.

Для підключення до інтернету та отримання даних задаємо параметри даних мережі Wi-fi та адресу API .

```
const char* ssid = "11";
const char* password = "11111111";
const char* api_url = "https://sirens.in.ua/api/v1/";
const char* MY_REGION = "Chernihiv";
```

Налаштовуємо піни та інтервал оновлення даних 30 секунд в даному випадку

```
const int RED_LED_PIN = 16;
const int GREEN_LED_PIN = 17;
unsigned long previousMillis = 0;
const long interval = 30000;
```

У функції setup() виконується ініціалізація світлодіодів та ініціалізація звукового модуля.

```
Serial.begin(115200);
  pinMode(RED_LED_PIN, OUTPUT);
  pinMode(GREEN_LED_PIN, OUTPUT);
  digitalWrite(RED_LED_PIN, LOW);
  digitalWrite(GREEN_LED_PIN, LOW);
  setup_wifi();
  mySoftwareSerial.begin(9600, SERIAL_8N1, 27, 26);
  Serial.println(F("\nІніціалізація DFPlayer..."));
  if (!myDFPlayer.begin(mySoftwareSerial)) {
    Serial.println(F("Не вдалося підключитися до DFPlayer Mini."));
    while (true) { }
  }
  Serial.println(F("DFPlayer Mini успішно ініціалізовано."));
  myDFPlayer.volume(10);
}
```

У функції setup() виконується підключення до Wi-Fi

```
void setup_wifi() {
  delay(10);
  Serial.println();
  Serial.print("Підключення до Wi-Fi: ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
```

```
  Serial.println("Wi-Fi підключено успішно!");
  Serial.print("IP адреса: ");
  Serial.println(WiFi.localIP());
 }
```

Функція loop перевіряє статус тривоги кожні 30 секунд та постійно отримує повідомлення від плеєру.

```
 void loop() {
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval) {
   previousMillis = currentMillis;

   if(WiFi.status() == WL_CONNECTED){
    checkSirenStatus();
   } else {
    Serial.println("Wi-Fi з'єднання втрачено. Спробую перепідключитися...");
    setup_wifi();
   }
  }

  if (myDFPlayer.available()) {
   printDetail(myDFPlayer.readType(), myDFPlayer.read());
  }
 }
```

Функція checkSirenStatus є ключовою частиною проєкту. Оскільки вона:

1. Надсилання запиту: Вона надсилає запит на URL-адресу, вказану у змінній api_url, щоб отримати актуальний список усіх областей та статусів тривоги в них.

2. Отримання відповіді: Функція чекає на відповідь від сервера. Якщо відповідь успішна вона зчитує отримані дані. Ці дані надходять у форматі JSON.
3. Розбір даних : За допомогою бібліотеки ArduinoJson вона перетворює текстову відповідь сервера на структуровані дані, з якими може працювати. Вона шукає масив (список) під назвою states.
4. Пошук свого регіону: Функція перебирає кожну область у списку, отриманому з сайту. Вона порівнює назву кожної області з тією, що вказана у змінній MY_REGION.
5. Визначення статусу тривоги: Як тільки вона знаходить потрібну область, вона перевіряє значення поля alert. Якщо воно full — значить, у вашому регіоні є тривога. Якщо null — тривоги немає.
6. Прийняття рішення :
   - Вона порівнює поточний статус (тривога є/немає) з тим, який був раніше (збережений у змінній currentState).
   - Якщо статус змінився (наприклад, раніше тривоги не було, а тепер є), вона виконує відповідні дії: вмикає потрібний світлодіод (червоний для тривоги, зелений для відбою) та запускає відповідний музичний трек (1-й або 2-й).
   - Якщо статус не змінився, вона нічого не робить, а просто виводить повідомлення про це, щоб не вмикати музику та світлодіоди заново кожні 30 секунд.

```
void checkSirenStatus() {
  HTTPClient http;
  WiFiClientSecure clientSecure;

  clientSecure.setInsecure();

  Serial.print("Перевірка статусу тривоги для регіону: ");
  Serial.println(MY_REGION);
```

```
if (http.begin(clientSecure, api_url)) {
  int httpCode = http.GET();

  if (httpCode > 0) {
    if (httpCode == HTTP_CODE_OK) {
      String payload = http.getString();

      JsonDocument doc;
      DeserializationError error = deserializeJson(doc, payload);

      if (error) {
        Serial.print(F("Помилка розбору JSON: "));
        Serial.println(error.c_str());
        http.end();
        return;
      }

      bool isAlertInRegion = null;

      JsonArray states = doc["states"];
      for (JsonObject state : states) {
        const char* regionName = state["name"];

        if (strcmp(regionName, MY_REGION) == 0) {
          isAlertInRegion = state["alert"];
          break;
        }
      }
```

```
            SystemState newStatus = isAlertInRegion ? SystemState::ALERT_ACTIVE :
SystemState::ALERT_INACTIVE;
            if (newStatus == SystemState::ALERT_ACTIVE && currentState !=
SystemState::ALERT_ACTIVE) {
              Serial.println("РЕЗУЛЬТАТ: Тривога активна! Вмикаю червоний LED та
трек 1.");
              digitalWrite(RED_LED_PIN, HIGH);
              digitalWrite(GREEN_LED_PIN, LOW);
              myDFPlayer.play(1);
              currentState = SystemState::ALERT_ACTIVE;
            } else if (newStatus == SystemState::ALERT_INACTIVE && currentState
!= SystemState::ALERT_INACTIVE) {
              Serial.println("РЕЗУЛЬТАТ: Тривоги немає. Вмикаю зелений LED та
трек 2.");
              digitalWrite(RED_LED_PIN, LOW);
              digitalWrite(GREEN_LED_PIN, HIGH);
              myDFPlayer.play(2);
              currentState = SystemState::ALERT_INACTIVE;
            } else {
               Serial.println("РЕЗУЛЬТАТ: Статус не змінився.");}
          }
        } else {
          Serial.printf("[HTTP] Помилка запиту")
      http.errorToString(httpCode).c_str());
        }
        http.end();
      } else {
        Serial.printf("[HTTP] Не вдалося підключитися до URL");
      }
    }
```

Висновок до розділу 3

Цей розділ присвячено опису розробки апаратної та програмної частини інформеру на мікроконтролері ESP32. Було розглянуто основні використані елементи, зокрема сам мікроконтролер ESP32, плеєр DFPlayer mini та динамік.

Для спрощення проєктування та документування схеми було застосовано програму Fritzing. Окремо описано процедуру отримання API-токену від сервісу https://sirens.in.ua/, що є ключовим для роботи програмного забезпечення.

Програмна логіка була реалізована на C++ в середовищі Arduino IDE. Було налаштовано необхідні бібліотеки та написано код, який забезпечує взаємодію з апаратними компонентами, надсилає запити до API для отримання даних про тривоги, обробляє їх та керує світлодіодною і звуковою індикацією.

В результаті, всі етапи розробки були успішно завершені, що гарантує злагоджену роботу компонентів та відповідність проєкту поставленим функціональним вимогам.

Висновок

В умовах повномасштабної військової агресії Російської Федерації проти України, забезпечення населення надійними та своєчасними засобами оповіщення про повітряну небезпеку набуло безпрецедентної актуальності та стало питанням збереження життів мільйонів громадян. Дана робота була присвячена розробці та дослідженню апаратно-програмного комплексу для оповіщення про повітряну тривогу, який би вирішував ключові проблеми існуючих систем, а саме їхню залежність від стабільного мобільного зв'язку, інтернет-доступу на кінцевих пристроях та доступності для всіх верств населення. Метою дослідження було створення автономного, надійного, недорогого та простого у використанні пристрою на базі мікроконтролера ESP32, здатного функціонувати незалежно від наявності смартфона у користувача.

На першому етапі роботи було проведено глибокий аналіз існуючих систем оповіщення, як традиційних, так і сучасних. Було встановлено, що класичні методи, такі як вуличні сирени, радіо та телебачення, мають свої недоліки: сирени не забезпечують деталізованої інформації та мають обмежене покриття , тоді як радіо і телебачення залежать від наявності електроенергії та активності з боку громадян. Цифрові інструменти, зокрема мобільний додаток "Повітряна тривога" та Telegram-канали, хоч і є надзвичайно ефективними у швидкості донесення інформації , створюють значний інформаційний розрив, оскільки їх використання неможливе без смартфона, стабільного інтернет-з'єднання та належної цифрової грамотності, що виключає значну частину населення, зокрема людей похилого віку та осіб з обмеженими можливостями. Комерційні пристрої, що вже існують на ринку, часто є занадто дорогими для масового споживача. Цей аналіз дозволив чітко сформулювати набір технічних вимог до розроблюваного пристрою, які лягли в основу подальшого проєктування. Функціональні вимоги включали підключення до Wi-Fi, періодичне опитування надійного джерела даних (API) та можливість налаштування регіону моніторингу. Нефункціональні вимоги акцентували увагу на

надійності роботи в режимі 24/7, високій швидкодії, енергоефективності та можливості резервного живлення.

На основі сформульованих вимог було обґрунтовано вибір апаратної та програмної бази. В якості центрального елемента системи було обрано мікроконтролер ESP32, який вигідно вирізняється на тлі альтернатив (як-от Arduino Uno/Nano) завдяки вбудованому Wi-Fi модулю, високій продуктивності двоядерного процесора, великій кількості пінів вводу-виводу та низькій вартості. Це дозволило значно спростити електричну схему та знизити собівартість кінцевого виробу. Для візуальної та звукової індикації були обрані прості та надійні компоненти: 5-мм світлодіоди, звуковий модуль DFPlayer Mini та динамік потужністю 0,5 Вт. Як середовище розробки було обрано Arduino IDE через її простоту, доступність, величезну базу готових бібліотек та прикладів, що ідеально підходить для швидкого прототипування. В якості джерела даних було обрано публічне API від sirens.in.ua, яке надає структуровану інформацію про тривоги у

JSON.

Процес проєктування включав розробку принципової електричної схеми та детального алгоритму роботи програмного забезпечення. Електрична схема була реалізована з акцентом на мінімалізм та надійність. Усі компоненти були підключені до плати ESP32: світлодіоди через струмообмежувальні резистори до цифрових пінів , а модуль DFPlayer Mini — через UART-інтерфейс для передачі команд та з окремим живленням 5В для стабільної роботи. Для уникнення збоїв у роботі звукового модуля при пікових навантаженнях було передбачено згладжувальний конденсатор. Алгоритм роботи програми був спроєктований як циклічний процес, що включає ініціалізацію системи та підключення до Wi-Fi, періодичне (кожні 30 секунд) надсилання HTTP-запиту до API, обробку отриманої JSON-відповіді, пошук цільового регіону та перевірку статусу тривоги. Ключовою особливістю алгоритму є перевірка зміни стану: система реагує (вмикає світлодіод та звуковий сигнал) лише в тому випадку, якщо статус тривоги змінився порівняно з попередньою перевіркою, що запобігає зайвим сповіщенням.

Апаратно-програмна реалізація проєкту підтвердила правильність обраних рішень. Було зібрано діючий прототип пристрою на макетній платі з використанням з'єднувальних дротів Dupont, що дозволило обійтися без пайки на етапі прототипування. Програмний код, написаний мовою C++ в середовищі Arduino IDE, успішно реалізував увесь запланований функціонал. Були використані та налаштовані стандартні бібліотеки для роботи з Wi-Fi (WiFi.h), HTTP-запитами (HTTPClient.h) та обробки JSON (ArduinoJson.h), що значно прискорило розробку. Логіка програми забезпечує автоматичне перепідключення до мережі Wi-Fi у разі втрати з'єднання , надійний розбір даних від API та адекватне керування світловою та звуковою індикацією залежно від статусу тривоги в обраному користувачем регіоні.

Таким чином, усі поставлені на початку дослідження завдання були повністю виконані. Розроблений апаратно-програмний комплекс є функціональним, надійним та доступним рішенням для оповіщення про повітряну тривогу. Практичне значення отриманих результатів полягає у створенні пристрою, який може бути використаний як основна або дублююча система оповіщення в побутових умовах, освітніх закладах, невеликих офісах та соціальних установах. Його низька собівартість та простота конструкції відкривають можливість для масового виробництва, що є критично важливим для забезпечення потреб вразливих категорій населення.

В якості шляхів для подальшого вдосконалення розробки можна виокремити декілька напрямків. В апаратній частині доцільно додати модуль резервного живлення (наприклад, на базі акумулятора та), що забезпечить повну автономність пристрою під час відключень електроенергії. Також можлива інтеграція невеликого OLED-дисплея для відображення додаткової інформації (назва регіону, час, тривалість тривоги). У програмній частині можна реалізувати веб-інтерфейс, що дозволить користувачам зручно налаштовувати Wi-Fi та обирати регіон через браузер без необхідності перепрошивки пристрою. Окрім того, схемотехнічні рішення та програмний код можуть слугувати основою для розробки інших пристроїв у сфері систем безпеки та "Інтернету речей".

Підсумовуючи, дана робота демонструє успішний приклад застосування сучасних доступних технологій для вирішення гострої соціальної проблеми. Створений пристрій є не просто технічним прототипом, а життєздатним і вкрай необхідним інструментом, що підвищує рівень безпеки громадян України в умовах війни.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Cameron N. ESP32 Microcontroller. ESP32 Formats and Communication. Berkeley, CA, 2023. P. 1–54. URL: https://doi.org/10.1007/978-14842-9376-8_1 .

2. Arduino Documentation URL: https://docs.arduino.cc/

3. Installing the ESP32 Board in Arduino IDE (Windows, Mac OS X, Linux) URL: https://randomnerdtutorials.com/installing-the-esp32-board-inarduino-ide-windows-instructions/

4. DF Player Mini Interface with ESP32 URL: https://www.hackster.io/munir03125344286/df-player-mini-interface-with-esp32f1efca

5. The Evolution of Air Threats in Future Conflicts URL: https://www.researchgate.net/publication/375975202_The_Evolution_of_Air_Threats_in_Future_Conflicts .

6. Lussier F. Future Air and Missile Threats. RAND Corporation, 2002. URL: https://doi.org/10.7249/rb3023

7. ESP32 HTTPS Requests (Arduino IDE) URL: https://randomnerdtutorials.com/esp32-https-requests

8. ESP32 Tutorials URL: https://esp32io.com/esp32-tutorials

9. Рэндэлл, Н. Інтернет речей: проєктування та створення пристроїв на базі Arduino та Raspberry Pi. – Київ: «Діалектика», 2019. – 480 с.

10. Проектування мікропроцесорних систем : навчальний посібник / С. М. Цирульник, Г. Л. Лисенко. – Вінниця : ВНТУ, 2012. – 191 с

Додаток А

Код системи оповіщення тривоги

```
#include "Arduino.h"
#include "DFRobotDFPlayerMini.h"
#include <WiFi.h>
#include <HTTPClient.h>
#include <WiFiClientSecure.h>
#include <ArduinoJson.h>


const char* ssid = "11";
const char* password = "11111111";

const char* api_url = "https://sirens.in.ua/api/v1/";
const char* MY_REGION = "Chernihiv";

const int RED_LED_PIN = 27;
const int GREEN_LED_PIN = 14;
unsigned long previousMillis = 0;
const long interval = 30000;


HardwareSerial mySoftwareSerial(2);
DFRobotDFPlayerMini myDFPlayer;


enum SystemState {
  NONE,          // Початковий стан, невизначений
  ALERT_ACTIVE,   // Тривога активна ("Full")
  ALERT_INACTIVE  // Тривога неактивна ("null")
};
SystemState currentState = SystemState::NONE;


void setup_wifi();
void checkSirenStatus();
void printDetail(uint8_t type, int value);

void setup() {
  Serial.begin(115200);

  pinMode(RED_LED_PIN, OUTPUT);
  pinMode(GREEN_LED_PIN, OUTPUT);
  digitalWrite(RED_LED_PIN, LOW);
  digitalWrite(GREEN_LED_PIN, LOW);

  setup_wifi();

  mySoftwareSerial.begin(9600, SERIAL_8N1, 27, 26);
  Serial.println(F("\nІніціалізація DFPlayer..."));
```

```
  if (!myDFPlayer.begin(mySoftwareSerial)) {
    Serial.println(F("Не вдалося підключитися до DFPlayer Mini."));
    while (true) {  }
  }
  Serial.println(F("DFPlayer Mini успішно ініціалізовано."));
  myDFPlayer.volume(10);
}

void setup_wifi() {
  delay(10);
  Serial.println();
  Serial.print("Підключення до Wi-Fi: ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("Wi-Fi підключено успішно!");
  Serial.print("IP адреса: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;

    if(WiFi.status() == WL_CONNECTED){
      checkSirenStatus();
    } else {
      Serial.println("Wi-Fi з'єднання втрачено. Спробую перепідключитися...");
      setup_wifi();
    }
  }


  if (myDFPlayer.available()) {
    printDetail(myDFPlayer.readType(), myDFPlayer.read());
  }
}

void checkSirenStatus() {
  HTTPClient http;
  WiFiClientSecure clientSecure;

  clientSecure.setInsecure();
```

```
Serial.print("Перевірка статусу тривоги для регіону: ");
Serial.println(MY_REGION);

if (http.begin(clientSecure, api_url)) {
  int httpCode = http.GET();

  if (httpCode > 0) {
    if (httpCode == HTTP_CODE_OK) {
      String payload = http.getString();

      JsonDocument doc;
      DeserializationError error = deserializeJson(doc, payload);

      if (error) {
        Serial.print(F("Помилка розбору JSON: "));
        Serial.println(error.c_str());
        http.end();
        return;
      }

      bool isAlertInRegion = false;

      JsonArray states = doc["states"];
      for (JsonObject state : states) {
        const char* regionName = state["name"];

        if (strcmp(regionName, MY_REGION) == 0) {
          isAlertInRegion = state["alert"];
          break;
        }
      }


      SystemState newStatus = isAlertInRegion ? SystemState::ALERT_ACTIVE : SystemState::ALERT_INACTIVE;


      if (newStatus == SystemState::ALERT_ACTIVE && currentState != SystemState::ALERT_ACTIVE) {
        Serial.println("РЕЗУЛЬТАТ: Тривога активна! Вмикаю червоний LED та трек 1.");
        digitalWrite(RED_LED_PIN, HIGH);
        digitalWrite(GREEN_LED_PIN, LOW);
        myDFPlayer.play(1);
        currentState = SystemState::ALERT_ACTIVE;
      } else if (newStatus == SystemState::ALERT_INACTIVE && currentState != SystemState::ALERT_INACTIVE) {
        Serial.println("РЕЗУЛЬТАТ: Тривоги немає. Вмикаю зелений LED та трек 2.");
        digitalWrite(RED_LED_PIN, LOW);
        digitalWrite(GREEN_LED_PIN, HIGH);
        myDFPlayer.play(2);
        currentState = SystemState::ALERT_INACTIVE;
      } else {
         Serial.println("РЕЗУЛЬТАТ: Статус не змінився.");
      }
    }
  } else {
```

```
      Serial.printf("[HTTP] Помилка запиту: %s\n", http.errorToString(httpCode).c_str());
    }
    http.end(); // Завершуємо сесію
  } else {
    Serial.printf("[HTTP] Не вдалося підключитися до URL\n");
  }
}

void printDetail(uint8_t type, int value) {

    case DFPlayerPlayFinished:
      Serial.print(F("Трек завершено, номер файлу: "));
      Serial.println(value);
      break;
    case Error:
      Serial.print(F("Помилка DFPlayer: "));
      Serial.println(value);
      break;
    default:
      break;
  }
}
```