Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Апаратно-програмний комплекс для дистанційного передавання аудіо. Частина 1. Кіберфізичний пристрій»

08-54.КБКР.036.00.000 ПЗ

Виконав: студент 4 курсу, групи 1КІ-21б
спеціальності 123 — «Комп'ютерна інженерія»
освітня програма — «Комп'ютерна інженерія»
Бондарчук О. Б. _____

Керівник д.т.н., проф. каф. ОТ
_____ Мартинюк Т. Б. _____

Рецензент к.т.н., доц. каф. ПЗ
_____ Катєльніков Д. І. _____

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.

«17» 06 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо — кваліфікаційний рівень бакалавр
Галузь знань — 12 Інформаційні технології
Спеціальність — 123 Комп'ютерна інженерія
Освітня програма — Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н. проф._____Азаров О. Д.
«21» березня 2025 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Бондарчуку Олександру Борисовичу

1 Тема роботи: Апаратно-програмний комплекс для дистанційного передавання аудіо. Частина 1. Кіберфізичний пристрій, керівник роботи Мартинюк Тетяна Борисівна, д.т.н., професор кафедри ОТ, затверджено наказом вищого навчального закладу від «20» березня 2025 року №97.

2 Строк подання студентом роботи 13.05.2025.

3 Вихідні дані до роботи: технічний опис кіберфізичного пристрою для дистанційного передавання аудіо, технології розробки: мова програмування С для ESP32, бібліотеки для роботи з обробкою аудіопотоків, керування записом файлів та мережевими функціями, технології для інтеграції з мобільними застосунками, цільова платформа системи — ESP32, середовище розробки — Arduino IDE.

4 Зміст розрахунково-пояснювальної записки: вступ; аналітичний огляд стану сучасних технологій дистанційного передавання аудіо; проєктування апаратно-програмного комплексу; конструкторська розробка кіберфізичного пристрою; оцінка результатів та перспективи розвитку; висновки.

5 Перелік графічного матеріалу: принципова схема підключення компонентів кіберфізичного пристрою; перелік ілюстративного матеріалу: макет розробленого

пристрою.

6 Консультанти розділів роботи приведені в таблиці 1.

Таблиця 1 — Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийня
1-4	д.т.н., проф. каф. ОТ Мартинюк Т. Б.	21.03.25	13.05.2
Нормоконтроль	ас. каф. ОТ Швець С. І.	14.05.25	30.05.2

7 Дата видачі завдання 21.03.2025 р.

8 Календарний план приведений в таблиці 2.

Таблиця 2 — Календарний план

№ з/п	Назва етапів дипломної роботи	Строк виконання	Підпис
1	Постановка задачі роботи	21.03.25	вик
2	Пошук матеріалів та інформаційних джерел	21.03.25— 30.03.25	вик
3	Аналітичний огляд стану сучасних технологій дистанційного передавання аудіо	31.03.25— 10.04.25	вик
4	Проєктування апаратно-програмного комплексу	11.04.25— 21.04.25	вик
5	Конструкторська розробка кіберфізичного пристрою	22.04.25— 30.04.25	вик
6	Оцінка результатів та перспективи розвитку	31.04.25— 02.05.25	вик
7	Оформлення пояснювальної записки та ілюстративного матеріалу	03.05.25— 11.05.25	вик
8	Перевірка якості виконання бакалаврської кваліфікаційної роботи та усунення недоліків	13.05.25	вик

Студент Олександр БОНДАРЧУК

Керівник роботи д.т.н., проф. Тетяна МАРТИНЮК

АНОТАЦІЯ

УДК 004.3

Бондарчук О. Б. Апаратно-програмний комплекс для дистанційного передавання аудіо. Частина 1. Кіберфізичний пристрій. Бакалаврська кваліфікаційна робота зі спеціальності 123 — Комп’ютерна інженерія, освітня програма — Комп’ютерна інженерія. Вінниця: ВНТУ, 2025. 77 ст.

На укр. мові. Бібліогр.: 22 назв; рис.: 31; табл.: 4.

У комплексній бакалаврській кваліфікаційній роботі розроблено апаратно-програмний комплекс для дистанційного передавання аудіо на базі мікроконтролера ESP32. Досліджено проблематику розробки кіберфізичних аудіосистем, зокрема забезпечення високої якості звуку, досягнення мінімальних затримок та роботи в стані обмежених ресурсів.

У ході виконання проєкту реалізовано двосторонній аудіострімінг у реальному часі через протокол UDP, передачу аудіофайлів через TCP та керування пристроєм за допомогою HTTP-запитів. Також реалізовано запис та відтворення отриманих аудіофайлів на MicroSD-картку.

Ключові слова: КФС, IoT, ESP32, апаратні компоненти, дистанційне передавання аудіо, мережеві протоколи.

ABSTRACT

Bondarchuk O. B. Hardware and software complex for remote audio transmission. Part 1. Cyber-physical device. Bachelor's qualification work in specialty 123 — Computer Engineering, educational program — Computer Engineering. Vinnytsia: VNTU, 2025. 77 p.

In Ukrainian. Bibliography: 22 titles; fig.: 31; tab.: 4.

In the comprehensive bachelor's qualification work, a hardware and software complex for remote audio transmission based on the ESP32 microcontroller was developed. The issues of developing cyber-physical audio systems were studied, in particular, ensuring high sound quality, achieving minimal delays and working in a state of limited resources.

During the project, two-way audio streaming in real time via the UDP protocol, audio file transmission via TCP and device control using HTTP requests were implemented. Recording and playback of received audio files on a MicroSD card were also implemented.

Keywords: CPS, IoT, ESP32, hardware components, remote audio transmission, network protocols.

ЗМІСТ

ВСТУП

У сучасному світі системи дистанційної передачі аудіо відіграють важливу роль у багатьох сферах життєдіяльності, зокрема в Інтернеті речей, телекомунікаціях, розумних будинках, освіті та мультимедійних технологіях. Зі зростанням популярності кіберфізичних пристроїв, таких як мікроконтролери, що забезпечують обробку та передачу аудіоданих у реальному часі, виникає потреба у створенні ефективних, зручних і надійних систем для дистанційного аудіообміну. Такі пристрої мають відповідати вимогам до якості звуку, стабільності зв’язку та обмежених ресурсів вбудованих систем, що робить їх розробку актуальною та затребуваною.

Актуальність теми полягає в інтеграції аудіосистеми у робототехнічну платформу з метою розширення функціональних можливостей пристрою. Йдеться про створення системи, яка дозволяє здійснювати аудіовзаємодію між роботом і користувачем, включаючи запис аудіофайлів на MicroSD-карту, відтворення аудіофайлів за командами з мобільного застосунку, а також передачу аудіо у реальному часі.

Метою роботи є створення системи для дистанційної передачі аудіо на базі мікроконтролера ESP32, яка забезпечуватиме двосторонній аудіострімінг у реальному часі через UDP, передачу аудіофайлів через TCP та керування пристроєм через HTTP-протокол.

Об'єктом дослідження є процеси обробки, передачі та керування аудіоданими в локальній мережі між клієнтським застосунком та мікроконтролером ESP32.

Предмет дослідження — апаратно-програмні засоби реалізації двостороннього аудіострімінгу, обміну файлами та керування IoT-пристроєм на базі протоколів UDP, TCP і HTTP.

До задач роботи належать:

— аналіз проблематики реалізації систем дистанційного передавання аудіо, зокрема якості звуку, затримок і обмежень ресурсів;

— дослідження архітектури обміну даними між клієнтським застосунком та ESP32 через протоколи UDP, TCP і HTTP;

— розробка функціональності для двостороннього аудіострімінгу в реальному часі з використанням UDP-протоколу;

— реалізація функціоналу запису та зчитування аудіофайлів з SD-карти через мікроконтролер ESP32;

— організація керування режимами роботи пристрою з мобільного застосунку;

— тестування розробленої системи у реальних умовах та аналіз її стабільності, швидкодії та якості звуку.

Апробація результатів комплексної бакалаврської кваліфікаційної роботи була проведена на конференції «LIV Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2025)» [1]:

Бондарчук О. Б., Дацюк В. В., Богомолов С. В., Мартинюк Т. Б. Дистанційна передача аудіо в IoT — Режим доступу : https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/24413

1 АНАЛІТИЧНИЙ ОГЛЯД СТАНУ СУЧАСНИХ ТЕХНОЛОГІЙ ДИСТАНЦІЙНОГО ПЕРЕДАВАННЯ АУДІО

1.1 Кіберфізичні системи в обробці та передачі аудіо

У сучасному світі, де фізичні об'єкти дедалі частіше взаємодіють з цифровим середовищем, важливу роль відіграють кіберфізичні системи (КФС). Кіберфізичні системи є широко розповсюдженим класом технологій, які поєднують обчислювальні, комунікаційні та фізичні процеси в цілісний працюючий механізм. У КФС програмне забезпечення безпосередньо керує фізичними процесами через обчислювальні та мережеві технології, чим забезпечує гнучкість та широку варіативність таких систем.

Типова кіберфізична система складається з: датчиків, які фіксують зміни навколишнього середовища; мікроконтролерів або серверів, які обробляють та аналізують дані з датчиків; комунікаційних мереж, тобто дротової або бездротової інфраструктури, яка передає дані між фізичними пристроями; виконавчих пристроїв, які керують фізичним процесом на основі команд від обчислювальних пристроїв; та програмного забезпечення, яке координує роботу усієї системи [2].

Роль КФС у сучасних технологіях полягає в їх здатності до створення спеціалізованих комплексів, які можуть отримувати інформацію із навколишнього середовища, обробляти великі обсяги даних і забезпечувати високу надійність, ефективність та масштабованість у різних сферах застосування. У галузі аудіообробки КФС дозволяють реалізувати системи із високою якістю відтворення, низькою затримкою передачі звуку та можливістю інтеграції з іншими пристроями, за допомогою екосистем Інтернету речей. Залучення таких механізмів дозволило сильно спростити реалізацію проектів, основою яких є обробка аудіоданих. Якщо раніше для запису, обробки чи передачі звуку була необхідна участь людини або складна апаратура зі значною кількістю ручних налаштувань, то сучасні системи дозволяють автоматизувати ці процеси майже повністю.

Реалізація такого підходу забезпечує постійне та автономне отримання, аналіз і відправку аудіоданих, оскільки кіберфізичні системи працюють в режимі

реального часу та не потребують втручання з боку людини. Наприклад, мікрофон типу MEMS може постійно «слухати» середовище й одразу передавати аудіосигнал на мікроконтролер, який уже в режимі реального часу обробляє отримані дані та, за потреби, надсилає їх далі.

Це дозволяє реалізовувати нові системи, які можуть: реагувати на зміни навколишнього середовища, наприклад вмикати запис при виявлені звуку, що успішно використовується в охоронних комплексах; записувати звуки на сервери або носії інформації, що також широко застосовується в охоронних або системах спостереження та багатьох інших галузях; передавати аудіо на інші пристрої або записувати його в «хмару»; або вмикати певні звуки згідно за розкладом або подією, наприклад для оповіщення в навчальних закладах або загального інформування населення при певних надзвичайних ситуаціях тощо.

Однією з передових сфер застосування КФС є їхня інтеграція в екосистеми Інтернету речей (IoT). Згідно з технічним звітом NIST, кіберфізичні системи прийнято вважати основою для IoT-систем, оскільки вони включають в себе розумні об’єкти, які можуть обробляти дані та керувати фізичними процесами. Прикладами таких застосувань у екосистемах Інтернету речей є фітнес-трекери, які вимірюють пульс і видають звукові сповіщення. Такі пристрої є КФС, що інтегруються з IoT для передачі мультимедійних даних [3].

Також аудіопристрої на базі кіберфізичних систем можуть виконувати функції голосового керування, охоронної системи, потокової передачі музики або двосторонньої комунікації у IoT-комплексах. Яскравим прикладом таких КФС є смарт-колонки Amazon Echo від Amazon або Google Home від Google (рис. 1.1). Вони поєднують обробку голосових команд та їх обчислення за допомогою хмарних сервісів, забезпечуючи користувачам зручність у використанні, та дозволяють інтегрувати інші IoT-пристрої, такі як датчики або камери, що дозволяє на їх базі розробити цілісну систему розумного будинку. Завдяки вбудованим мікрофонам і високочутливим аудіомодулям ці пристрої можуть розпізнавати голос користувача навіть на відстані. Це робить їх зручними не лише для побутового використання, а й у медичних, освітніх або виробничих середовищах

та вказує на актуальність розробки таких систем.

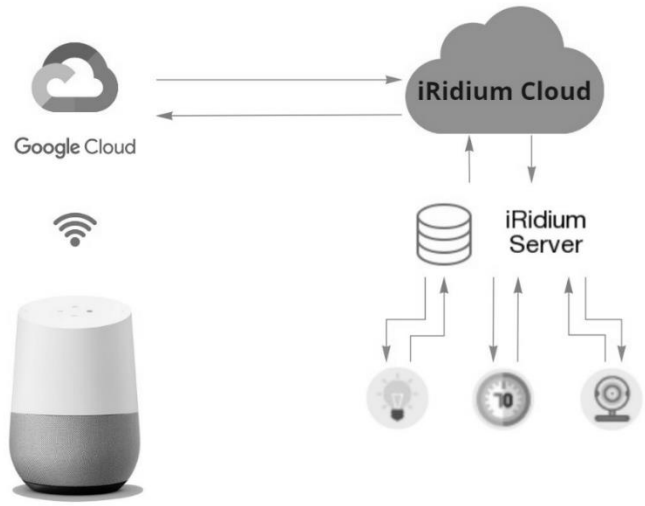

Рис. 1.1 — Приклад роботи кіберфізичної системи Google Home

1.2 Аналіз сучасних методів і технологій передачі аудіо

1.2.1 Протоколи передачі аудіо

Основою сучасних систем дистанційної передачі аудіо є мережеві протоколи, які забезпечують ефективну передачу даних із мінімальними затримками та високою якістю звуку. Базовими протоколами, які використовуються для потокової передачі аудіо, є UDP (User Datagram Protocol), TCP (Transmission Control Protocol) та WebRTC (Web Real-Time Communication). Кожен із них має ряд своїх переваг та обмежень, які й визначають доцільність їх використання у різних системах.

UDP (User Datagram Protocol) є одним з найпростіших протоколів транспортного рівня моделі OSI, який характеризується низькою затримкою та мінімальними накладними витратами. Він не передбачає процедури встановлення логічного з’єднання перед початком передачі, тому UDP відносять до протоколів, що здійснюють передавання даних без встановлення з’єднання [4]. У UDP дані передаються у вигляді дейтаграм без гарантії доставки чи порядку пакетів, що може призводити до втрати даних у нестабільних мережах, проте для аудіопотоків, таких як голосовий зв’язок або радіотрансляція, втрата окремих пакетів не дуже

критична, оскільки людське вухо зазвичай не помічає незначних пропусків. Структуру UDP-дейтаграми наведено на рисунку 1.2.

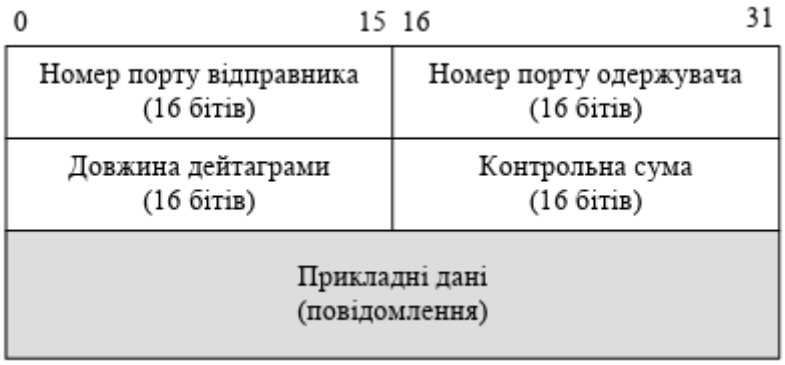

Рис. 1.2 — Структура UDP-дейтаграми

Цей протокол широко використовують у системах VoIP (Voice over IP), таких як Skype або Discord, оскільки він має невеликий розмір заголовка, покращений механізм керування передаванням даних, що дозволяє набагато швидше передавати дані, у порівнянні з іншими мережевими протоколами. Саме з цієї причини програми, що працюють в реальному часі, надають перевагу протоколу UDP [4]. Однак у мережах із високим рівнем перешкод або втратою пакетів якість звуку може погіршуватися, що вимагає додаткових механізмів буферизації або корекції помилок.

TCP (Transmission Control Protocol) — це основний транспортний протокол зі стека протоколів TCP/IP. Він забезпечує надійне передавання потоку даних, використовуючи при цьому ненадійний сервіс транспортування пакетів, що надається протоколом IP. В мережах IP протокол TCP використовується для обробки запитів на вхід в мережу, доступу до файлових ресурсів в локальних і глобальних мережах, передавання списків ресурсів тощо [4]. На відміну від UDP, TCP є протоколом із встановленням з’єднання, який гарантує надійну доставку даних у правильному порядку. TCP застосовує механізми підтвердження отримання пакетів і повторної передачі у випадку їх втрати, що робить його корисним для систем, у яких потрібна висока надійність, наприклад, для передачі аудіофайлів або завантаження їх на пристрій.

У TCP дані передаються у вигляді сегментів. Кожен сегмент містить частину даних, які потрібно передати, а також метадані, необхідні для забезпечення надійної доставки. За гарантію доставки у протоколі відповідає механізм номеру в послідовності, який визначає номер першого байта у черзі байтів в поточному сегменті. А також механізм номеру підтвердження, що містить наступний номер байта, який очікує відправник у відповідь на надісланий сегмент [4]. Формат TCP-сегмента зображено на рисунку 1.3.

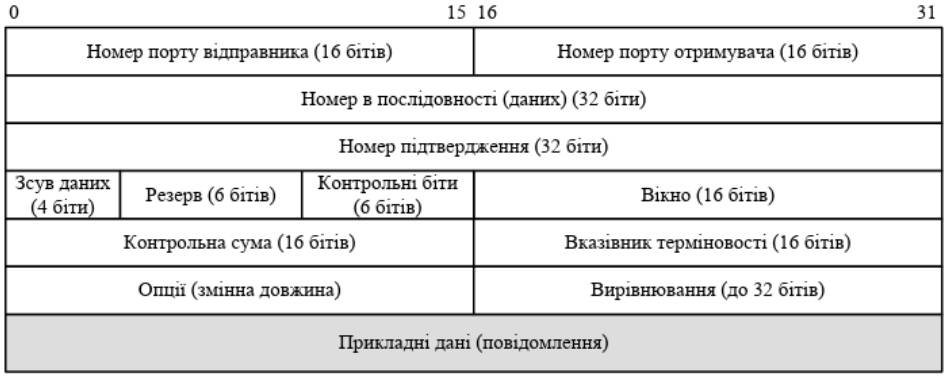

Рис. 1.3 — Формат TCP-сегмента

У сфері аудіообробки цей протокол часто використовують для передачі великих файлів на сервер або пристрій зберігання. Хоча він має більші затримки через використання механізму, який потрібен для встановлення з’єднання та перевірку цілісності даних, тому це робить його не таким ефективним для потокового аудіо в реальному часі, як протокол UPD.

WebRTC (Web Real-Time Communication) — це інтернет-протокол із відкритим кодом, призначений для організації голосового та відеозв'язку через інтернет у режимі реального часу [5]. Дана технологія широко використовується у системах відеоконференцій та голосового зв’язку, оскільки WebRTC забезпечує надійність сесії та дозволяє обійтися без передачі даних медіасерверу, зменшити затримку і забезпечити кращу якість контенту, що передається. Також, це зменшує навантаження на сервер [6].

Хоча WebRTC спочатку був розроблений для веб-браузерів, він має й додатки для пристроїв без браузера, включаючи мобільні платформи та пристрої

Інтернету речей. Для передачі аудіоданих протокол використовує аудіокодек Opus, який генерує аудіо данні з високою якістю [6]. Однак WebRTC вимагає набагато більше обчислювальних ресурсів для обробки кодеків та мережевих протоколів, у порівнянні з UDP, тому використання цього потоколу у проектах, які базуються на мікропроцесорах із невеликим обсягом пам’яті, не є доцільним.

При порівнянні цих протоколів, можна позначити, що вибір певного протоколу залежить від конкретного завдання. Наприклад, UDP доцільніше використовувати при потоковій передачі аудіоданих, щоб забезпечити стабільну систему реального часу. TCP варто використовувати для передачі файлів або керування пристроєм через HTTP-запити, а WEBRTC найбільше підходить складних систем, які мають великі обчислювальні потужності.

1.2.2 Апаратні платформи для передачі аудіо

При розробці систем для дистанційної передачі аудіо важливим етапом є коректний вибір апаратної платформи. Серед найбільш поширених платформ варто відзначити ESP32, Raspberry Pi та STM32, оскільки вони пропонують підтримку різних мережевих інтерфейсів, надають гнучкі інструменти для обробки аудіоданих та є доволі доступними у ціновій категорії. Кожна із цих платформ має ряд своїх переваг та обмежень, які й визначають доцільність їх використання у різних системах.

ESP32 — це двоядерний мікроконтролер від Espressif Systems із вбудованими модулями Wi-Fi та Bluetooth. Він оснащений 32-бітним процесором Tensilica Xtensa LX6 із частотою до двісті сорока МГц, має 520 КБ оперативної пам’яті SRAM та підтримує популярні інтерфейси I2S, SPI, I2C, DAC, ADC та UART [7]. Вбудований Wi-Fi модуль підтримує стандарти 802.11 b/g/n (2.4 ГГц), що дозволяє підключатися до бездротових мереж або створювати точки доступу. Мікроконтролер ESP32 зображено на рисунку 1.4.

У контексті аудіообробки ESP32 підтримує інтерфейс I2S для підключення цифрових мікрофонів і підсилювачів, що забезпечує якісний запис аудіоданих і відтворення звуку з частотою дискретизації до сорока восьми кГц [8]. Також він

підтримує інтерфейси ADC та DAC, які використовуються для перетворення аналогових аудіосигналів у цифровий формат та перетворення цифрових аудіоданих у аналогові сигнали відповідно. У випадку із дистанційною передачою аудіо, доцільніше використовувати інтерфейс I2S, оскільки він обробляє цифрові аудіодані напряму, минаючи обмеження вбудованих перетворювачів.

Перевагами ESP32, у порівнянні з іншими платформами, є низька вартість, енергоефективність і компактні розміри, що робить його найкращим вибором для портативних аудіопристроїв. Однак обмежена обчислювальна потужність і пам’ять, порівняно з більш потужними платформами, такими як Raspberry Pi, можуть ускладнювати обробку складних аудіокодеків, таких як MP3 або AAC.

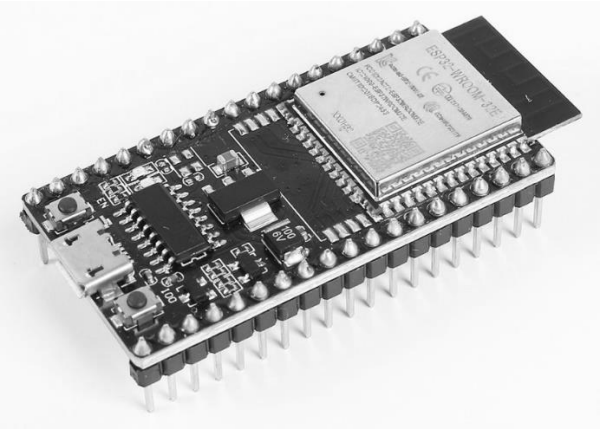

Рис. 1.4 — Мікроконтролер ESP32

Raspberry Pi — це назва серії одноплатних комп'ютерів, створених Raspberry Pi Foundation, британською благодійною організацією, метою якої є навчання людей комп'ютерним технологіям і полегшення доступу до комп'ютерної освіти. По суті це дуже дешевий комп'ютер, що працює під управлінням Linux, але й також має набір виводів GPIO [9]. Популярним слоганом компанії є «Це комп’ютер, розміром з кредиту карту», що не є перебільшенням, оскільки він пропонує значно більшу обчислювальну потужність порівняно з мікроконтролерами. Наприклад, Raspberry Pi 4 оснащений чотирьохядерним процесором Cortex-A72 із частотою до 1.5 ГГц, 2-8 ГБ оперативної пам’яті та підтримкою операційних систем. Raspberry Pi підтримує обробку аудіо через USB-аудіоінтерфейси, HDMI або зовнішні I2S-

модулі, що дозволяє використовувати складні кодеки і потокові протоколи для реалізації аудіосистем [10]. Крім того, Raspberry Pi має вбудовані бездротові можливості і може бути підключений до таких пристроїв, як клавіатура, миша і монітор. Однак Raspberry Pi має вищу ціну, більше енергоспоживання і більші розміри, що робить його менш придатним для компактних портативних пристроїв. Мікрокомп'ютер Raspberry Pi зображено на рисунку 1.5.

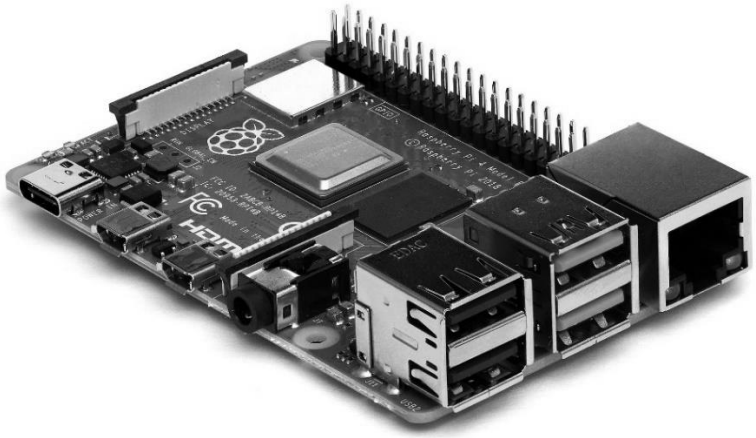

Рис. 1.5 — Мікрокомп'ютер Raspberry Pi

Ще однією популярною платформою для обробки аудіо є STM32. Це сімейство мікроконтролерів компанії STMicroelectronics, побудованих на архітектурі ARM-сімейства мікропроцесорів Cortex-M. Це актуальна та ефективна архітектура від компанії ARM, призначена для побудови пристроїв з невеликим електроспоживаням і високою продуктивністю [11]. Завдяки високій продуктивності та підтримці цифрової обробки сигналів, вони часто використовуються в аудіообробці. Наприклад, STM32H7 оснащений ядром Cortex-M7 із частотою до чотириста вісімдесяти МГц і вбудованим модулем DSP для обробки аудіосигналів та підтримує інтерфейс I2S, який може працювати з цифровими мікрофонами та підсилювачами [11]. На відміну від ESP32, STM32 зазвичай не має вбудованих модулів Wi-Fi або Bluetooth. Тому при виборі цієї платформи для реалізації систем дистанційної передачі аудіо, необхідно додатково докупляти зовнішній бездротовий модуль. Переваги STM32 включають високу обчислювальну потужність для обробки складних алгоритмів і гнучкість у

конфігурації апаратних інтерфейсів. Однак складність програмування та вища ціна роблять STM32 менш доступним для простих проектів порівняно з ESP32. Мікроконтролер STM32 зображено на рисунку 1.6.

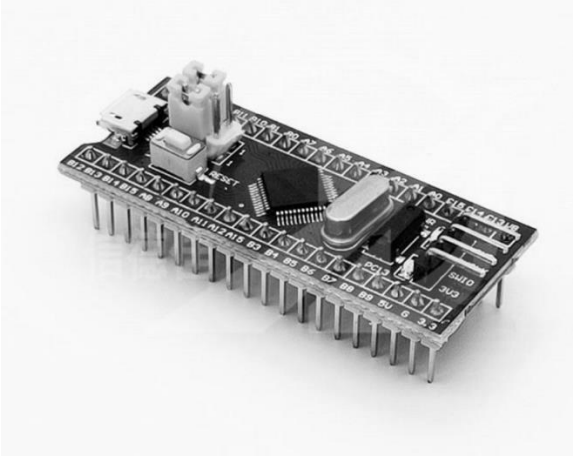

Рис. 1.6 — Мікроконтролер STM32

У таблиці 1.1 показана порівняльна характеристика доцільності використання кожної із платформ при реалізації системи для дистанційної передачі аудіо.

Табл. 1.1 — Порівняльна характеристика платформ

Критерій	Raspberry Pi	ESP32	STM32
Обчислювальна потужність	Висока: 4 ядра, до 1.5–2.0 ГГц, до 8 ГБ RAM	Середня: 2 ядра, 240 МГц, ~520 КБ SRAM	Середня: до 2 ядра, 72–216 МГц, до 512 КБ SRAM
Мережеві можливості	Wi-Fi + Bluetooth, Ethernet через USB	Вбудовані Wi-Fi та Bluetooth	Потребує зовнішніх модулів
Енергоефективність	Високе споживання (~400–700 мА, 5В)	Дуже енергоефективний (активний режим ~150 мА, глибокий сон ~5 мкА)	Помірне споживання (залежить від моделі, ~30–150 мА)
Вартість	Вища (~30–80 $ залежно від моделі та наявності)	Низька (~4–10 $), висока доступність	Середня (~10–20 $), широка модельна лінійка

Продовження табл 1.1

Критерій	Raspberry Pi	ESP32	STM32
Інтерфейси для аудіо	I2S, USB, потребує зовнішніх аудіо HAT або USB-карт	Вбудований I2S, просте підключення мікрофонів/підсилювачів	Вбудований I2S, добре підходить для роботи з аудіо ADC/DAC
Простота інтеграції	Потребує ОС та SD-карти, запуск Linux	Просте програмування через Arduino IDE або ESP-IDF	Вимагає складніших засобів розробки (STM32CubeIDE, бібліотеку)
Оптимальне застосування	Потужні мультимедійні/мережеві пристрої з високим навантаженням	Компактні, бездротові, енергоефективні аудіопристрої	Системи з низьким енергоспоживанням і складною цифровою обробкою сигналу

Згідно з цією таблицею, ESP32 є найкращим варіантом для розробки компактної, бездротової та енергоефективної системи передачі аудіо, що й реалізовано у межах цього проєкту.

1.3 Проблематика реалізації систем дистанційного передавання аудіо

Під час розробки систем дистанційного передавання аудіо виникає ряд труднощів, які стосуються технічних, апаратних, програмних та мережевих взаємодій. Успішна реалізація таких систем потребує глибокого розуміння цифрової обробки сигналів, ефективного використання обчислювальних ресурсів, а також забезпечення стабільності бездротового з’єднання.

Однією із головних проблем є забезпечення високої якості звучання. Цифрові мікрофони та підсилювачі часто вносять шуми, такі як фоновий гул або кліки, що погіршує сприйняття музики чи мовлення, тому їх використання потребує фільтрації та калібрування. Частота дискретизації 8-16 кГц, яку зазвичай використовують при розробці IoT або КФС-пристроїв, обмежує частотний діапазон до чотирьох-восьми кГц, що недостатньо для високоякісного відтворення [12].

Використання кодеків MP3 чи AAC, зменшує навантаження на мережу за

рахунок стиснення аудіофайлу, але втрата даних погіршує якість, що критично для розбірливості мовлення чи точності музики, тому в деяких варіантах доцільніше використовувати PCM (імпульсно-кодову модуляцію), Вона забезпечує високу точність, хоча й генерує великі потоки даних, ускладнюючи передачу в обмежених мережах. На рисунку 1.7 зображено графік-порівняння використання PCM та MP3 для дистанційної передачі аудіо.

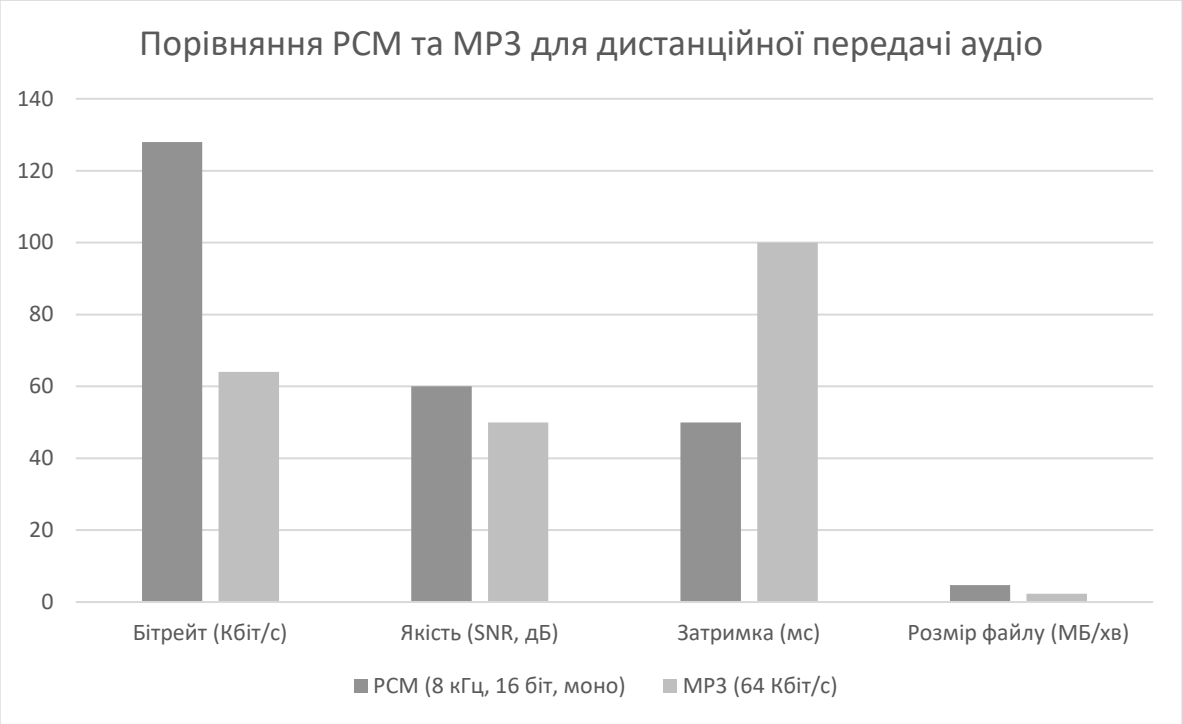

Рис. 1.7 — Графік-порівняння PCM та MP3

Ще одним ключовим параметром є наявність затримок передачі. Для систем реального часу затримки у 250 мс та більше є дуже критичними, оскільки вони напряму впливають на якість та бажання користування системою. Затримки виникають під час буферизації аудіосигналу, яка потрібна для запобігання розривів, та передачі сигналу в мережу. Також вони можуть бути зумовлені логікою апаратних компонентів, які використовуються при розробці.

Більшість мікроконтролерів, таких як ESP32 або STM32, мають обмежений обсяг оперативної пам’яті і обчислювальної потужності. Це ускладнює одночасну роботу з периферійними аудіокомпонентами та мережею. Програмування таких систем вимагає глибокого розуміння роботи інтерфейсів I2S, SPI, UART, які є

основою взаємодії між компонентами.

Також на затримки передачі сильно впливають мережеві протоколи, які використовуються системою. Наприклад протокол UDP частково вирішує цю проблему, оскільки цей протокол не передбачає процедури встановлення логічного з’єднання перед початком передачі. Це дозволяє зменшити наявність затримок, але не гарантує доставки сигналу, що іноді викликає «заїкання» аудіо, особливо в умовах із низькою стабільністю мережі [12]. TCP, хоч і надійніший, але він уповільнює передачу через механізми підтвердження, що не підходить для систем реального часу.

На рисунку 1.8 зображено графік втрати UDP-пакетів в залежності від відстані клієнта до системи.

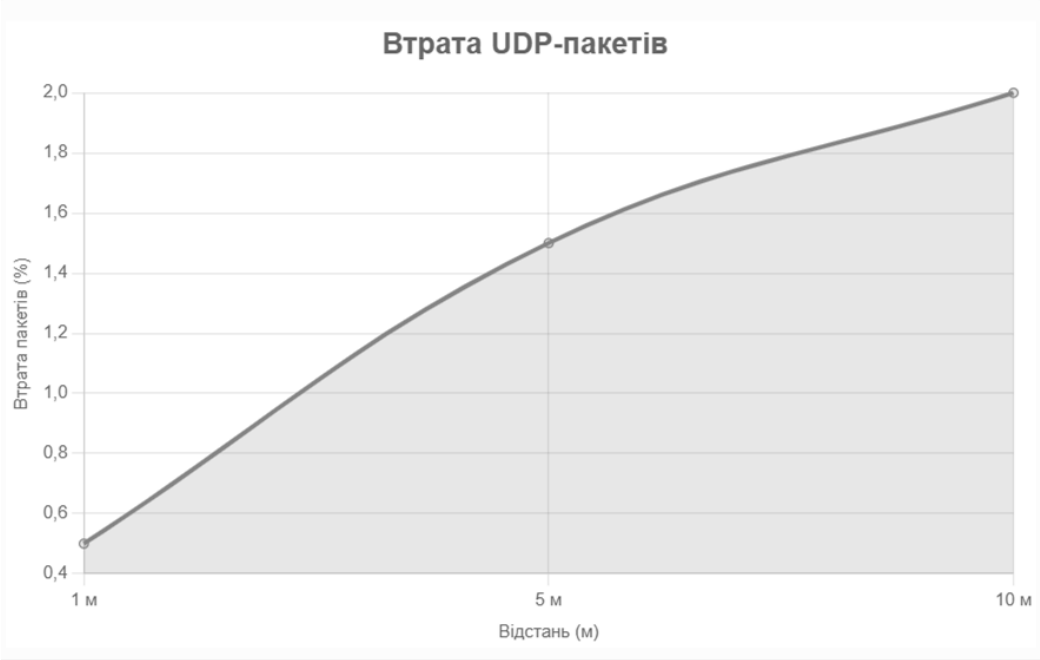

Рис. 1.8 — Втрата UPD-пакетів

Отже, реалізація систем дистанційного передавання аудіо пов’язана з низкою технічних проблем — від забезпечення стабільного з’єднання до програмного керування буферизацією та ресурсами пристрою. Вирішення цих задач вимагає ретельного проектування як апаратної, так і програмної частини системи, вибору відповідної платформи та застосування оптимізованих алгоритмів обробки сигналів і передачі даних.

2 ПРОЄКТУВАННЯ АПАРАТНО-ПРОГРАМНОГО КОМПЛЕКСУ

2.1 Визначення технічних вимог до комплексу

Для вдалої реалізації будь-якої системи перш за все необхідно чітко визначити технічні вимоги, які забезпечуватимуть реалізацію її функціональних можливостей та відповідність стандартам якості й ефективності. Визначення цих вимог є основою для вибору апаратних компонентів і програмних методів, а також для оцінки їхньої взаємодії в межах кіберфізичної системи, яка має працювати в умовах обмежених ресурсів і змінних мережевих умов.

Реалізований у цій роботі апаратно-програмний комплекс має забезпечувати чітку обробку, запис, передачу та відтворення аудіоінформації в режимі реального часу. Цей пристрій позиціонується як додатковий модуль до вже готового кіберфізичного пристрою — розумного робота «ELEGOO Smart Robot Cat Kit V4.0». ELEGOO Smart Robot зображено на рисунку 2.1.

Рис. 2.1 — ELEGOO Smart Robot Cat Kit V4.0

Така інтеграція передбачає розширення можливостей базового пристрою за рахунок реалізації аудіофункціоналу, що робить робота не просто мобільним пристроєм, а більш інтерактивною кіберфізичною системою, здатною реагувати на звукове середовище.

Основна мета розробки полягає у створенні компактної, енергоефективної і стабільної системи, яка працюватиме автономно або з мінімальним втручанням користувача. Для реалізації цієї мети, було сформовано перелік технічних вимог, що охоплюють як апаратну, так і функціональну частину системи.

У якості центрального елемента комплексу обрано мікроконтролер ESP32, який поєднує в собі підтримку бездротового зв’язку, інтерфейс I2S для обміну аудіоданими та достатню обчислювальну потужність. Це дає змогу обробляти аудіосигнал безпосередньо на модулі, не навантажуючи обчислювальні ресурси самого робота.

Для захоплення звуку використовується цифровий MEMS мікрофон INMP441 з інтерфейсом I2S, який забезпечує частоту дискретизації 8-16 кГц і розрядність у 16 біт. Отримані дані повинні передаватися через Wi-Fi у форматі UDP-потоку. Для голосового зв’язку, як у системах типу рації, частота дискретизації 8 кГц є достатньою, оскільки вона охоплює діапазон частот людської мови, забезпечуючи чіткість при мінімальному обсязі даних.

Передача звуку з мережі на пристрій реалізується за допомогою цифрового аудіопідсилювача MAX98357 і динаміка, що дозволяє відтворювати голосові повідомлення або аудіофайли, попередньо записані на micro SD картку. Для якіснішого відтворення музики чи складніших аудіосигналів частота дискретизації може бути збільшена до шістнадцяти-сорока чотирьох кГц.

Затримка передачі є критичним параметром для систем реального часу. У бездротових мережах, таких як Wi-Fi, затримка залежить від мережевої швидкості, обробки сигналу та буферизації. Для голосового зв’язку затримка не повинна перевищувати 150 мс, оскільки більші значення сприймаються користувачем як неприродні паузи. У Wi-Fi мережах стандарту 802.11n затримка передачі пакетів становить 10-50 мс у стабільних умовах, але може зростати через перешкоди або завантаженість мережі [13]. Обробка аудіосигналу, зокрема аналогово-цифрове перетворення та буферизація, додає затримки в діапазоні 1-5 мс для обробки та до шестидесяти чотирьох мс для буфера при частоті 8 кГц. Для забезпечення низької затримки комплекс має використовувати протоколи з мінімальними накладними

витратами, такі як UDP, і оптимізувати розмір буфера, щоб уникнути надмірного накопичення даних. Крім того, система має бути стійкою до мережевих перебоїв, що досягається шляхом буферизації.

Додаткові вимоги включають енергоефективність, оскільки комплекс призначений для портативного використання на базі розумного робота ELEGOO Smart Robot Car Kit V4.0. У зв'язку з тим що робот працює автономно й живиться від акумулятора, то модуль повинен мати низьке енергоспоживання: не перевищувати 150-200 мА у робочому режимі та досягати менше ніж 10 мкА у режимі очікування. Це дозволяє забезпечити тривалий час автономної роботи без значного навантаження на загальну систему живлення робота.

Через обмежений вільний простір усередині платформи робота, компактність конструкції є ключовою вимогою. Всі компоненти мають бути мініатюрними та легко встановлюватися. Такий підхід забезпечує просту фізичну інтеграцію модуля до вже наявної електронної системи робота без необхідності значних апаратних модифікацій.

Вимоги до керування включають можливість віддаленої взаємодії через веб-інтерфейс, що забезпечує інтуїтивне налаштування функцій, таких як вибір файлу для відтворення, , перегляд логів або ініціація потокової передачі. Це дозволяє локально керувати модулем через точку доступу ESP32.

З огляду на портативну специфіку та обмежені ресурси мікроконтролера ESP32, аудіообробка базується на простому форматі PCM з фіксованою частотою дискретизації, без використання складних кодеків. Це дозволяє зменшити затримки й гарантує сумісність із наявними бібліотеками та апаратними можливостями.

Таким чином, визначені вимоги до кіберфізичного пристрою враховують усі ключові особливості інтеграції з ELEGOO Smart Robot, забезпечуючи його портативність, енергоефективність, простоту підключення, стабільність роботи та інтуїтивне керування.

2.2 Вибір апаратних компонентів та їх взаємодії

Однією із основних частин розробки кіберфізичної системи є коректний та

ретельний вибір апаратних складових, які будуть не лише виконувати необхідну функціональність, а й врахуватимуть обмеження платформи, до якої вони інтегруються. Враховуючи, що комплекс має на меті доповнити функціональність вже існуючого робота ELEGOO Smart Robot Car Kit V4.0, ключовими критеріями при виборі компонентів стали: компактність, енергоефективність, сумісність з Arduino-подібними платформами, підтримка аудіоінтерфейсів та бездротової передачі даних та доступність у ціновій категорії.

В якості основної обчислювальної системи було обрано мікроконтролер ESP32, який є потужним мікроконтролером із вбудованим Wi-Fi та Bluetooth модулями, двоядерним процесором Xtensa LX6 та широким набором периферійних інтерфейсів (рис. 2.2).

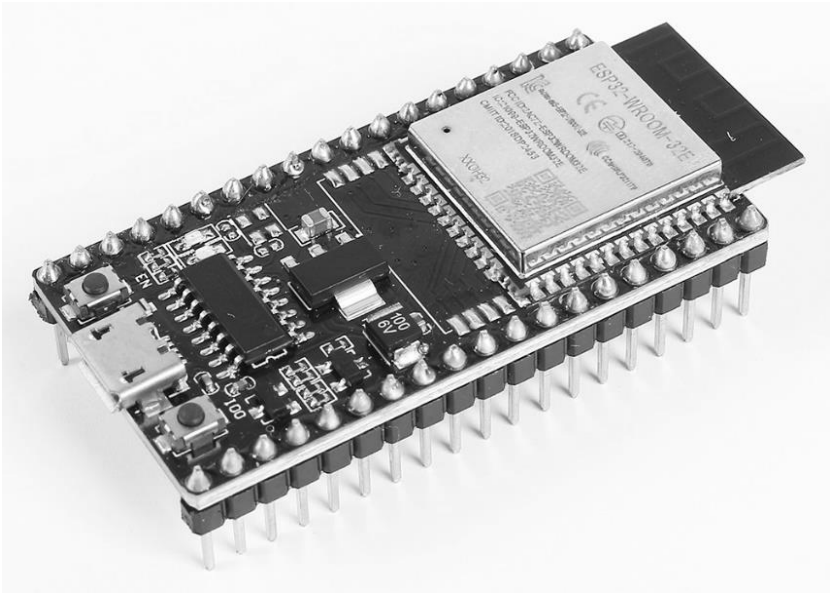

Рис. 2.2 — Мікроконтролер ESP32

Завдяки підтримці інтерфейсу I2S, ESP32 найкраще підходить для реалізації систем, які базуються на обробці звуку. Він дозволяє підключати цифрові аудіокомпоненти, такі як мікрофони та підсилювачі, для обробки сигналів із частотою дискретизації до сорока восьми кГц.

I2S — це інтерфейс послідовної шини, спеціально розроблений для передачі цифрових аудіоданих між інтегральними схемами. Протокол I2S надсилає аудіодані з імпульсно-кодовою модуляцією від контролера до цільового пристрою

[14]. Інтерфейс має щонайменше три шини. Дані передаються по лінії SD, стан лінії WS відповідає аудіоканалу (правому чи лівому), який передається в даний момент, а лінія тактової частоти несе послідовний тактовий сигнал. Як показано на рисунку 2.3, сигнали WS та SCK можуть генеруватися передавачем, приймачем або стороннім контролером.

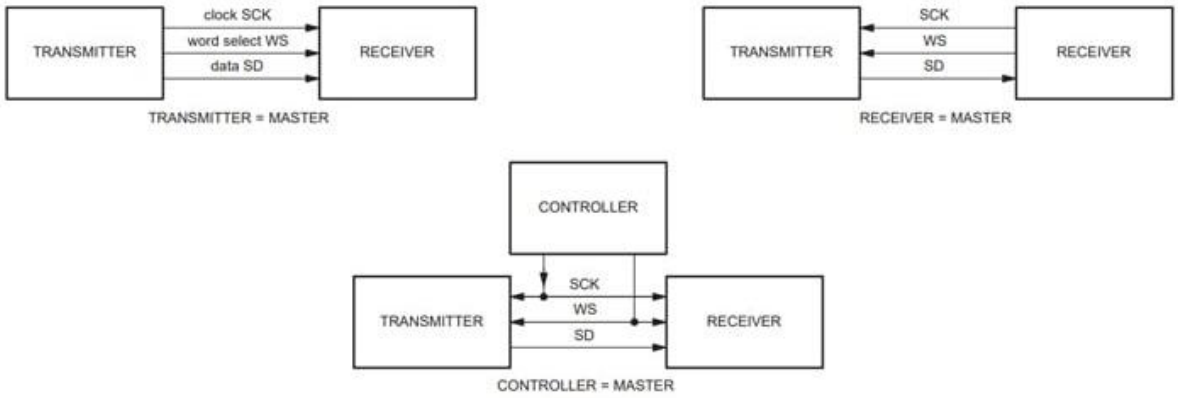

Рис. 2.3 — Передача сигналів по I2S

Цифровий звук має багато переваг для систем, яким потрібно багато працювати з аудіосигналом. Його можна легко обробити та передати бездротовому модулю. Наприклад, у розумній колонці звук передається та приймається через Wi-Fi для хмарної обробки та потокової передачі. Цифрові мікрофонні масиви краще підходять для захоплення аудіо на великій відстані, а підсилювачі з цифровим входом можуть обробляти потік I2S для внутрішнього вирівнювання та регулювання гучності, щоб невеликі колонки або динаміки звучали голосніше [14].

Ще одним важливим та необхідним інтерфейсом, який підтримує ESP32, є SPI, оскільки для реалізації функціональності збереження аудіофайлів використовується модуль MicroSD Card.

SPI — це послідовний синхронний інтерфейс, який є чотирьох-провідним, та призначений для повнодуплексного обміну даними між ведучим та підлеглими пристроями. Інтерфейсна шина SPI використовується для передачі даних між мікроконтролерами та периферійними пристроями, такими як регістри зсуву,

датчики, оперативна та flash пам'ять, АЦП, ЦАП, SD-карти тощо. Шина SPI використовує окрему лінію синхронізації та лінії даних [14].

У протоколі SPI є два типи пристроїв: Master (ведучий) та Slave (підлеглий). Ведучий пристрій є головним на шині, в даному випадку це мікроконтролер, а всі інші пристрої вважаються підлеглими. Ведучий пристрій генерує тактовий сигнал синхронізації (SCLK), по якому синхронізується передача вхідних та вихідних даних [14]. Вхідний та вихідний сигнали позначаються MOSI та MISO (рис. 2.3).

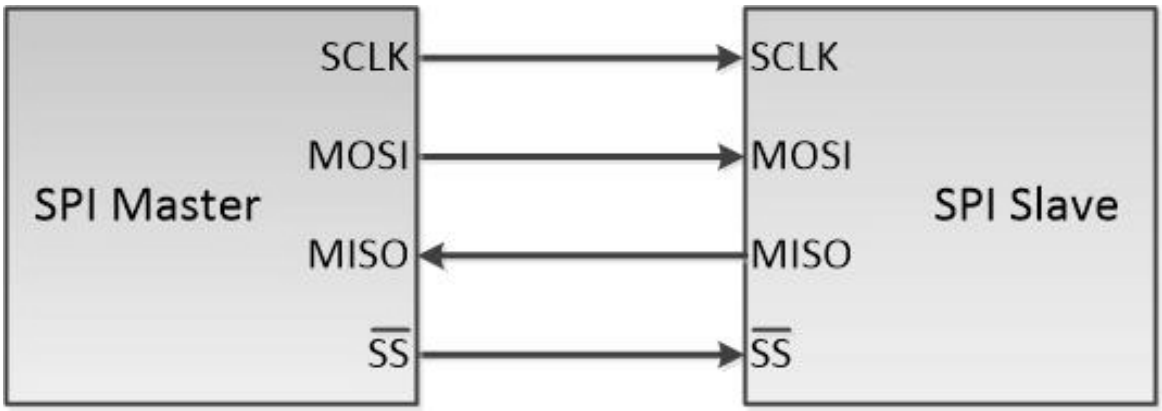

Рис. 2.3 — Передача сигналів по SPI

MOSI (англ. Master Out Slave In) — вихід ведучого, вхід підлеглого. Дані передаються від ведучого пристрою до підлеглого.

MISO (англ. Master In Slave Out) — вхід ведучого, вихід підлеглого. Дані передаються від підлеглого пристрою до ведучого.

Також на шині практично завжди присутній сигнал SS або CS (англ. Chip Select або Slave Select), який призначений для вибору підлеглого пристрою. Зазвичай це активний низький сигнал, який стає високим при завершенні сеансу зв'язку. Коли є кілька підлеглих пристроїв, то ведучий пристрій має окремий вихід цього сигналу для кожного підлеглого пристрою [14].

Не менш важливим критерієм до вибору ESP32 є наявність будованих модулів Wi-Fi і Bluetooth, які забезпечують бездротову передачу даних і керування, що усуває потребу в додаткових модулях. Енергоефективність ESP32, із споживанням 160-260 мА в активному режимі Wi-Fi і до п’яти мкА в режимі глибокого сну, робить його придатним для портативних пристроїв, що працюють

від акумулятора, а сумісність із Arduino IDE спрощує розробку програмного забезпечення, дозволяючи використовувати різноманітні бібліотеки для керування периферійними пристроями.

Для реалізації функції захоплення аудіосигналу було обрано цифровий MEMS мікрофон INMP44 (рис. 2.4).

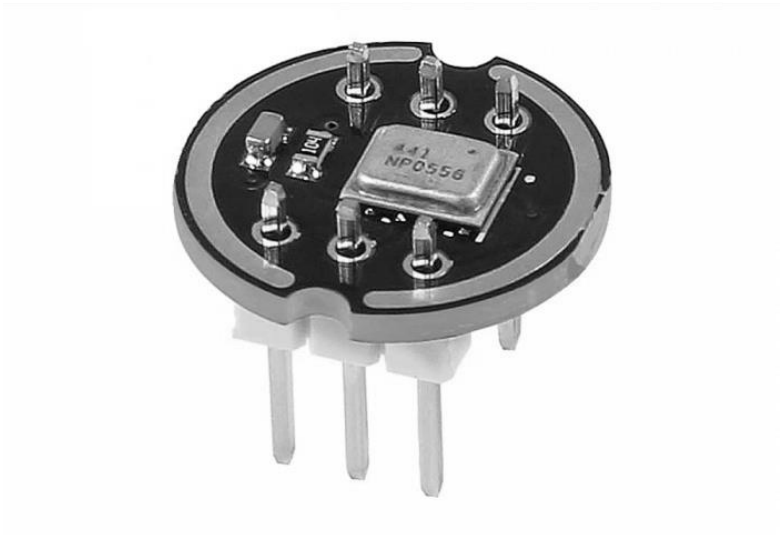

Рис. 2.4 — Цифровий MEMS мікрофон INMP441

INMP441 — це високопродуктивний MEMS-мікрофон з цифровим виходом I2S та низьким електроспоживанням. Він спеціально розроблений для передових застосувань розпізнавання аудіо та голосу. Його сумісність з інтерфейсом I2S, який використовується для передачі цифрового аудіо між пристроями, гарантує його можливість інтеграції в різноманітні системи обробки аудіо [15].

INMP441 забезпечує якісний запис звуку з частотою дискретизації від восьми до сорока восьми кГц і 24-бітною глибиною, що відповідає вимогам для голосового зв’язку та простого аудіо. Його діапазон частот 60 Гц-15 кГц охоплює спектр людської мови, а відношення сигнал/шум 61 дБ забезпечує чіткість запису навіть у шумному середовищі. Мікрофон працює при напрузі 1.8-3.3 В і споживає близько 1.5 мА, що сприяє енергоефективності системи [16]. INMP441 передає цифровий сигнал через I2S, що дозволяє легко інтегрувати його з ESP32, уникаючи додаткових аналогово-цифрових перетворювачів. Компактні розміри мікрофона задовольняють умову в розробці портативного пристрою, а його низька ціна

робить його економічно вигідним в порівнянні з аналогами.

На рисунку 2.5 показано функціональну блок-схему мікрофона INMP441. Він містить MEMS-датчик, систему обробки сигналу, АЦП, фільтри згладжування, систему керування живленням та стандартний 24-бітний інтерфейс I2S.

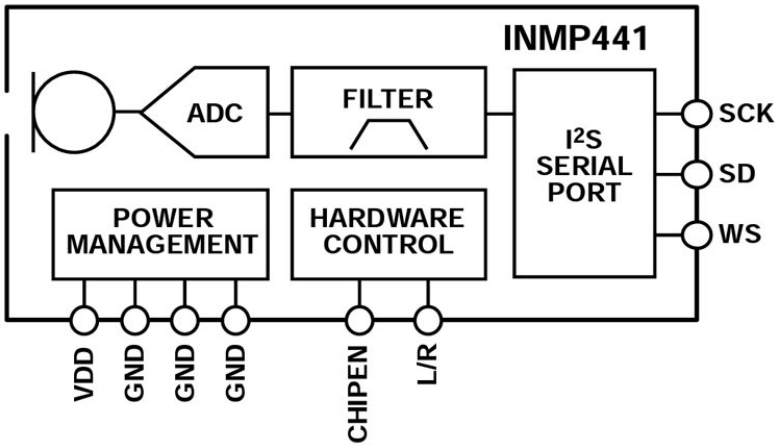

Рис. 2.5 — Функціональна блок-схема INMP441

Для реалізації функції відтворення аудіосигналу було обрано цифровий аудіопідсилювач MAX98357 (рис. 2.6).

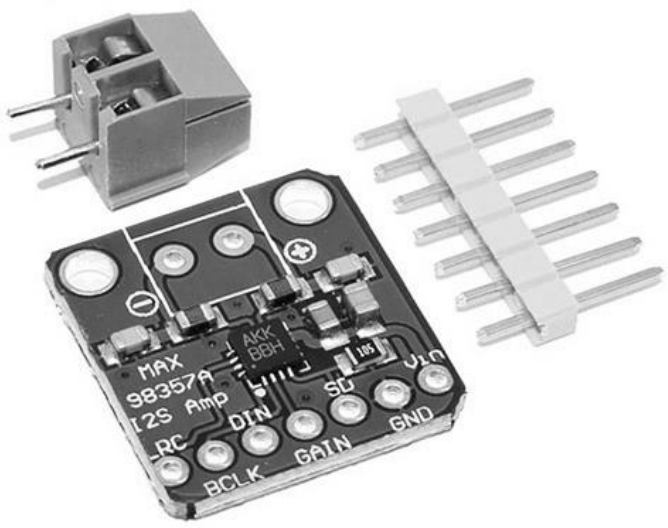

Рис. 2.6 — Аудіопідсилювач MAX98357

MAX98357 — це простий у використанні та недорогий цифровий підсилювач класу D, розроблений для вхідного сигналу PCM (імпульсно-кодованої модуляції).

Завдяки своєму універсальному цифровому аудіоінтерфейсу він автоматично розпізнає до тридцяти п’яти різних схем тактування PCM та TDM , що усуває необхідність програмування I2C. Підсилювач спрощує експлуатацію, усуваючи потребу в зовнішньому сигналі MCLK, який використовується в зв'язку PCM, зменшуючи електромагнітні перешкоди та потенційні проблеми зі зв'язком плат[17].

У контексті розроблюваної системи аудіопідсилювач MAX98357 використовується для відтворення аудіо через динамік, забезпечуючи високу якість звуку при низькому енергоспоживанні. Його вихідна потужність становить приблизно 3.2 Вт при навантаженні 4 Ом і напрузі 5 В. Він підтримує I2S для прямого підключення до ESP32, що дозволяє передавати цифровий аудіосигнал без додаткових перетворювачів. Підсилювач сумісний із частотами дискретизації 8-96 кГц і забезпечує ефективність до 92%, що зменшує теплові втрати та сприяє енергоефективності [18]. Низьке споживання струму в режимі очікування і компактний корпус роблять MAX98357 ідеальним для портативних аудіосистем.

На рисунку 2.7 показано функціональну блок-схему аудіопідсилювача MAX98357. Він містить вхід для цифрового PCM-сигналу, систему обробки сигналу, ЦАП, фільтри згладжування, вихідний каскад класу D, вхід для керування вибору каналу та систему керування підсиленням.

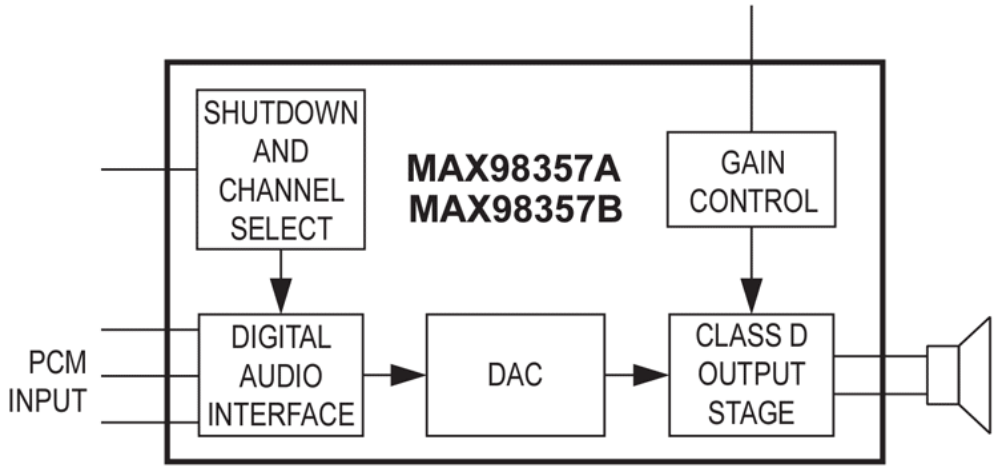

Рис. 2.7 — Функціональна блок-схема аудіопідсилювача MAX98357

За виведення аудіоданих відповідає широкосмуговий динамік з номінальною

потужністю 3 В та електричним опором 4 Ом, який добре поєднується із підсилювачем (рис. 2.8).

Рис. 2.8 — Широкосмуговий динамік

Для забезпечення відтворення низькочастотного діапазону динамік має фазоінверторний порт, прикритий захисною сіткою. Завдяки динаміку з неодимовим магнітом та якісною жорсткою мембраною цей динамік забезпечує цілком пристойну якість звучання без пригуків пластику. Мала вага дозволяє закріпити корпус практично в будь-якому місці, що є великою перевагою в умовах обмеженого простору на роботі.

Для збереження аудіофайлів використовується модуль MicroSD, який дозволяє записувати та відтворювати аудіо у форматі WAV (рис. 2.9). Це простий у використанні модуль з інтерфейсом SPI та вбудованим стабілізатором напруги 3,3 В для забезпечення належного живлення SD-карти. Модуль підключається до ESP32 через інтерфейс SPI, який забезпечує швидкість передачі даних до двадцяти МБ/с, достатню для роботи з аудіофайлами розміром кілька мегабайт. MicroSD карти підтримують ємність до тридцяти двох ГБ, що дозволяє зберігати значну кількість аудіоданих, наприклад, до кількох годин запису при частоті 8 кГц і 16-бітній глибині. Споживання модуля становить близько 20-30 мА під час читання/запису, що є прийнятним для портативних систем. Сумісність із бібліотеками, такими як SdFat, спрощує програмну обробку файлів, а компактні розміри модуля сприяють реалізації портативного комплексу.

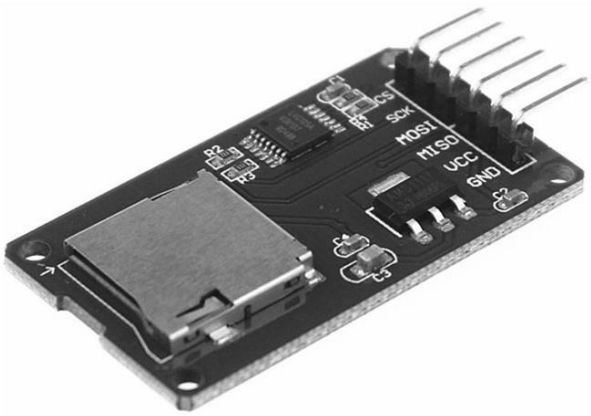

Рис. 2.9 — Модуль MicroSD

Взаємодія компонентів системи забезпечується через відповідні інтерфейси та програмне забезпечення. ESP32 виступає центральним контролером, який обробляє аудіосигнали від INMP441, передає аудіодані на MAX98357 для відтворення, використовуючи інтерфейс I2S, та записує файли на Micro SD картку через SPI. За допомогою вбудованого Wi-Fi модуля реалізовується потокова передача аудіо через UDP і керування усім функціоналом через HTTP-запити, дозволяючи взаємодіяти з зовнішнім пристроєм, а саме смартфоном. Блок-схема взаємодії компонентів системи зображена на рисунку 2.9.

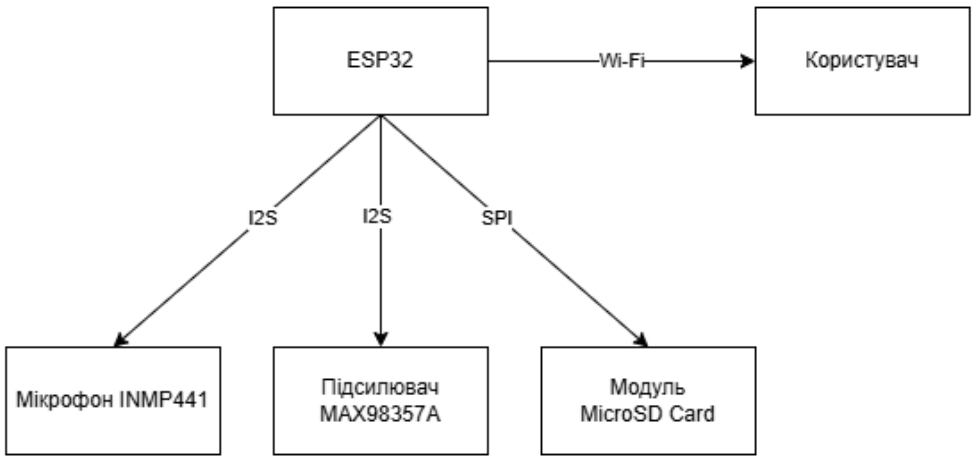

Рис. 2.9 — Блок-схема взаємодії компонентів системи

Оскільки система позиціонується як додатковий модуль до вже наявного пристрою і використовуватиме його джерело живлення, то оцінка енергоспоживання комплексу є критично важливою для забезпечення сталої роботи. Важливо, щоб модуль не створював надмірного навантаження на акумуляторну систему, водночас забезпечуючи повноцінну функціональність для запису, зберігання, передавання та відтворення аудіо. ESP32 споживає 160-260 мА при активному Wi-Fi, 20 мА без Wi-Fi/Bluetooth і до п’яти мкА в режимі глибокого сну. INMP441 споживає 1.5 мА під час запису, а MAX98357 — до п’ятдесяти мА при відтворенні на максимальній гучності та приблизно 1 мА в режимі очікування. Модуль Micro SD споживає 20-30 мА під час операцій читання/запису. Загальне енергоспоживання системи в активному режимі (з увімкненим Wi-Fi, записом і відтворенням) становить приблизно 230-340 мА при напрузі 3.3-5 В. Для акумулятора ємністю 2000 мА це забезпечує автономну роботу протягом 5-8 годин, що є прийнятним для цієї конфігурації. У режимі очікування (без Wi-Fi і аудіообробки) споживання знижується до десяти-двадцяти мкА, що дозволяє значно продовжити час роботи.

Таким чином, обрані апаратні компоненти дозволяють досягти оптимального балансу між функціональністю та енергоефективністю. Контролер ESP32 забезпечує обчислювальні ресурси та мережеві взаємодії, мікрофон INMP441 — якісний цифровий запис, аудіопідсилювач MAX98357 — потужне та чисте відтворення, а модуль MicroSD — надійне зберігання аудіофайлів. Усі компоненти мають малі габарити, що дозволяє легко інтегрувати модуль у корпус робота без суттєвого збільшення ваги чи зміни балансу.

2.3 Програмні засоби для реалізації та тестування системи

При розробці кіберфізичних систем важливим етапом є правильний вибір актуальних програмних засобів, які зможуть забезпечити повноцінне функціонування системи. Основними критеріями при виборі є забезпечення стабільної взаємодії між апаратними компонентами та можливість зручного тестування для вчасного виявлення помилок.

Для реалізації системи дистанційного передавання аудіо базовим середовищем для написання програмного коду було обрано Arduino IDE, оскільки воно підтримує широку вибірку різних мікроконтролерів, включаючи ESP32, має підтримку багатьох бібліотек та дозволяє швидко завантажувати прошивку на пристрій через USB.

Arduino IDE — містить текстовий редактор для написання коду, область повідомлень, текстову консоль, панель інструментів з кнопками для виконання поширених функцій та низку меню. Воно підключається до мікроконтролерів для завантаження програм та взаємодії з ними [20].

На рисунку 2.10 показано стартове вікно середовища Arduino IDE.

Рис. 2.10 — Стартове вікно середовища Arduino IDE

Програми, написані за допомогою програмного забезпечення Arduino IDE, називаються скетчами. Ці скетчі пишуться в текстовому редакторі та зберігаються з розширенням файлу «.ino». Редактор має функції для вирізання/вставки та пошуку/заміни тексту. Область повідомлень надає зворотний зв'язок під час збереження та експорту, а також відображає помилки. Консоль відображає текстовий вивід програмним забезпеченням Arduino IDE, включаючи повні повідомлення про помилки та іншу інформацію. У правому нижньому куті вікна

відображається налаштована плата та послідовний порт. Кнопки панелі інструментів дозволяють перевіряти та завантажувати програми, створювати, відкривати та зберігати скетчі, а також відкривати монітор послідовного порту [20].

Arduino IDE підтримує багато різноманітних бібліотек, які можна завантажити прямо в середовищі або імпортувати з комп'ютера. Вони надають додаткові функції для полегшення роботи із апаратними компонентами або маніпулювання даними. Щоб використовувати бібліотеку в проекті, необхідно знайти її в меню «Library Manager» або імпортувати за допомогою меню «Sketch» - «Імпорт бібліотеки». Далі у текстовому полі необхідно ввести назву бібліотеки за допомогою оператора «#include» у верхній частині скетчу. Це скомпілює бібліотеку зі скетчем. Варто зауважити, що бібліотеки завантажуються на плату разом із скетчем, тому це збільшує обсяг місця, яке займає загальна програма.

На рисунку 2.11 зображено приклад встановлення та використання бібліотеки.

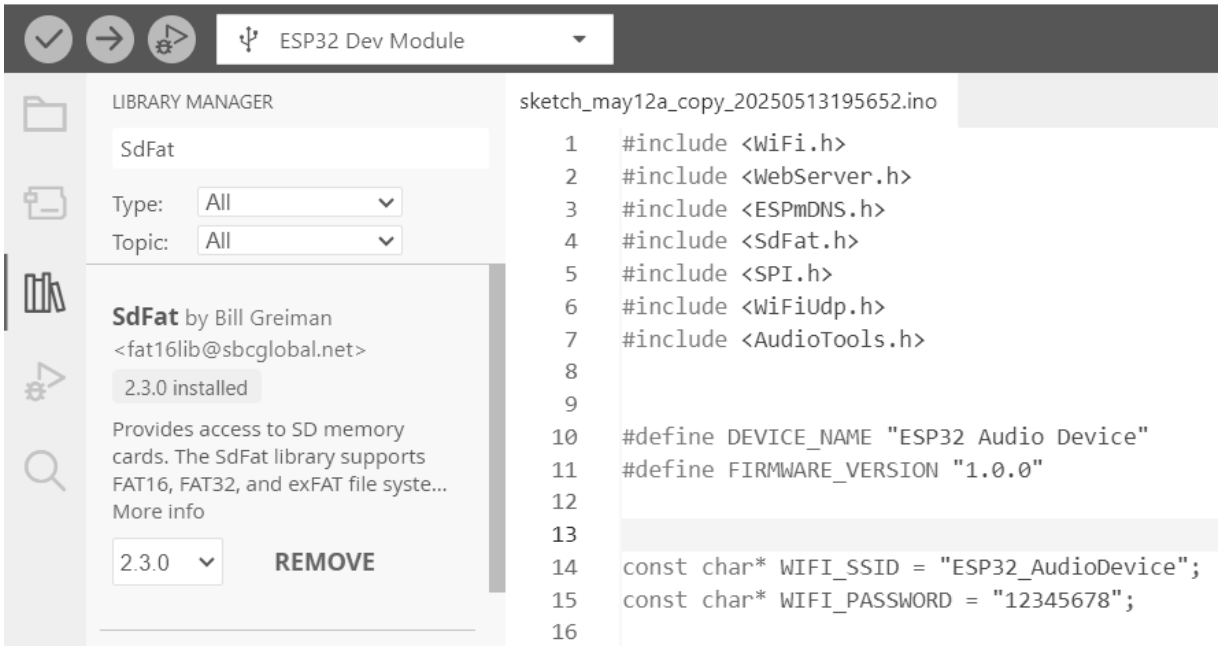

Рис. 2.11 — Встановлення та використання бібліотеки

Arduino IDE дозволяє створювати прошивки на основі C/C++, що полегшує програмування мікроконтролера без необхідності низькорівневих маніпуляцій із регістрами. Під час завантаження скетчу, він використовує завантажувач Arduino,

який надає можливість завантажувати код без використання додаткового обладнання. Завантажувач активний протягом кількох секунд після перезавантаження плати, а потім він запускає той скетч, який був останнім завантажений на мікроконтролер.

Не менш важливим функціоналом середовища є наявність серійного монітора, який забезпечує виведення діагностичних даних, таких як стан буферів, частота кадрів або інформацію про помилки при компілюванні, що дуже сильно спрощує відлагодження. Для коректного відображення повідомлень необхідно у випадаючому меню монітору вибрати швидкість передачі даних, яка відповідає швидкості, заданій у Serial.begin у конкретному скетчі. Завдяки цьому можна швидко тестувати код, змінюючи параметри, такі як частота дискретизації або розмір буфера, і перевіряти їхній вплив на продуктивність.

Для виконання технічних вимог до пристрою було використано декілька ключових бібліотек, які полегшили реалізацію складних функцій, таких як обробка аудіопотоків та обробка HTTP-запитів. А саме було використано бібліотеки AudioTools, SdFat, WiFiUdp, WebServer, WiFi та SPI.

Бібліотека AudioTools відіграє ключову роль у реалізації обробки аудіопотоків, забезпечуючи захоплення, передачу та відтворення аудіоданих. Вона працює з інтерфейсом I2S і підтримує PCM-аудіо з частотою дискретизації від восьми до сорока восьми кГц та глибиною 16 біт, що відповідає вимогам голосового зв’язку [21]. У проекті AudioTools використовується для створення об’єктів аудіообробки, наприклад як I2SStream, який захоплює аудіо з мікрофона INMP441 і передає його в мережу через UDP. Бібліотека дозволяє налаштувати параметри потоку, наприклад, розмір буфера в 512 байт, що забезпечує затримку обробки до тридцяти двох мс при восьми кГц. AudioTools також підтримує обробку WAV-файлів, що використовується для запису та відтворення на MicroSD-карті. Для тестування результату бібліотека має інструменти з моніторингу аудіопотоку, дозволяючи перевірити якість сигналу та виявити переповнення буфера.

Для роботи з файловою системою на MicroSD-карті була обрана бібліотека SdFat. Вона значно продуктивніша за стандартну SD.h, особливо в умовах роботи з

великими обсягами даних у режимі реального часу. SdFat підтримує створення WAV-файлів з параметрами 8 кГц / 16 біт, а через інтерфейс SPI забезпечує швидкість запису до п’ятисот КБ/с [22]. У проекті SdFat використовується для відкриття файлів, буферизації даних і синхронного запису потоку з I2S, уникаючи переривань, які могли б викликати втрату даних або зависання. Для тестування стабільності SdFat проводилися цикли запису/читання файлів розміром до п’ятдесяти МБ. У результаті усі файли були успішно записані на MicroSD-картку, що підтверджує коректність вибору цієї бібліотеки.

Для реалізації функцій дистанційного передавання аудіо використовується бібліотека WiFiUdp, яка налаштовує ESP32 для відправлення та прийому потокових аудіопакетів. У проекті ця бібліотека реалізовує UDP-сервер, який передає дані з бітрейтом 128 кбіт/с у пакетах по 512 байт. Для перевірки коректності роботи у системі було проведено тестування двостороннього зв’язку, де ESP32 надсилав і приймав аудіопотоки, перевіряючи синхронізацію та стабільність. У результаті усі аудіосигнали були успішно відтворені, що підтверджує коректність вибору цієї бібліотеки.

Для реалізації керування функціями системи через окермий веб-інтерфейс було обрано бібліотеку WebServer. У проекті бібліотека обробляє HTTP-запити з часом відповіді 10-50 мс, що перевірялося шляхом надсилання кількох запитів із клієнтських пристроїв. Налаштування ESP32, як точки доступу для підключення пристроїв, реалізовується за допомогою бібліотеки WiFi.

У підсумку, використання Arduino IDE, як програмного середовища, дозволило повноцінно реалізувати програмну частину системи. Застосування вищевказаних бібліотек дозволило спростити роботу із апратними компонентами та забезпечило гнучку діагностику, налаштування та керування пристроєм.

3 КОНСТРУКТОРСЬКА РОЗРОБКА КІБЕРФІЗИЧНОГО ПРИСТРОЮ

3.1 Побудова апаратної частини системи

Конструкторська розробка апаратно-програмного комплексу для дистанційного передавання аудіо передбачає створення апаратної частини, яка забезпечує ефективну обробку, передачу та збереження аудіоданих. Основним компонентом пристрою є мікроконтролер ESP32, який дозволяє інтегрувати цифровий MEMS мікрофон INMP441, аудіопідсилювач MAX98357 і модуль MicroSD в єдину функціонуючу систему. Ці компоненти обрано з огляду на їхню сумісність, енергоефективність і компактність, що відповідає заданим технічним вимогам.

Мікроконтролер ESP32 має 34 виводи для підключення периферійних пристроїв (рис. 3.1). При розробці системи використовується лише 10 із них, які підтримують інтерфейси I2S та SPI.

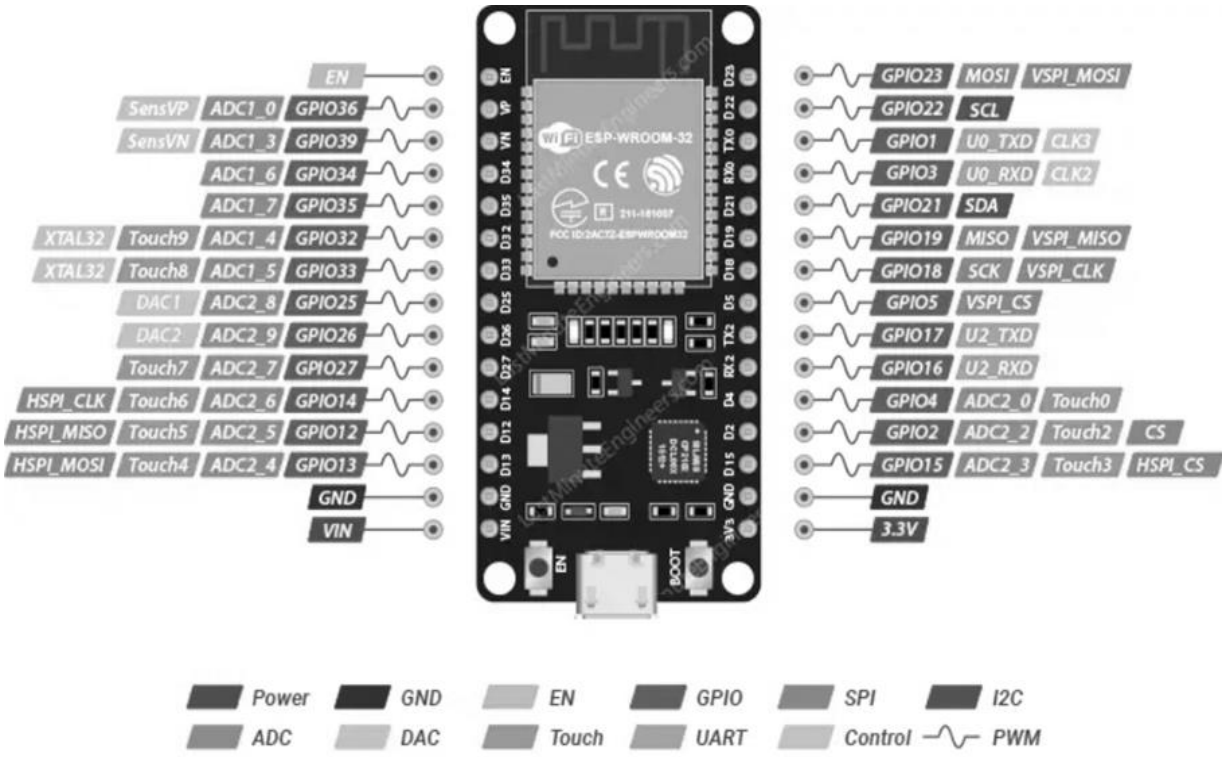

Рис. 3.1 — Опис пінів мікроконтролера ESP32

У таблицях 3.1-3.3 зображено підключення периферійних компонентів до мікроконтролера ESP32.

Табл. 3.1 — Схема підключення мікрофону INMP441 з ESP32

Контакт INMP441	Призначення	Пін ESP32
VDD	Живлення 3.3V	3.3V
GND	Земля	GND
SCK (BCLK)	Тактова синхронізація	GPIO14
WS (LRCK)	Вибір каналу	GPIO15
SD (Data)	Аудіо-дані	GPIO32

Табл. 3.2 — Схема підключення аудіопідсилювача MAX98357 з ESP32

Контакт MAX98357	Призначення	Пін ESP32
VIN	Живлення 5V	5V
GND	Земля	GND
DIN	Аудіо-дані	GPIO25
BCLK	Тактова синхронізація	GPIO27
LRC	Вибір каналу	GPIO26

Табл. 3.3 — Схема підключення модуля MicroSD-карти з ESP32

Контакт MAX98357	Призначення	Пін ESP32
VCC	Живлення 5V	5V
GND	Земля	GND
CS	Вибір пристрою	GPIO5
MOSI	Передача даних	GPIO23
MISO	Прийом даних	GPIO19
SCK	Тактова синхронізація	GPIO18

Інтерфейс I2S, який забезпечує синхронну передачу цифрових аудіоданих із частотою дискретизації до сорока восьми кГц, використовується для під’єднання

мікрофона INMP441 і аудіопідсилювача MAX98357, тоді як SPI застосовується для зв’язку з модулем Micro SD. Такий розподіл виводів дозволяє ефективно обробляти аудіопотоки, зберігати дані та забезпечувати стабільну роботу системи. На рисунку 3.2 зображена принципова схема підключення.

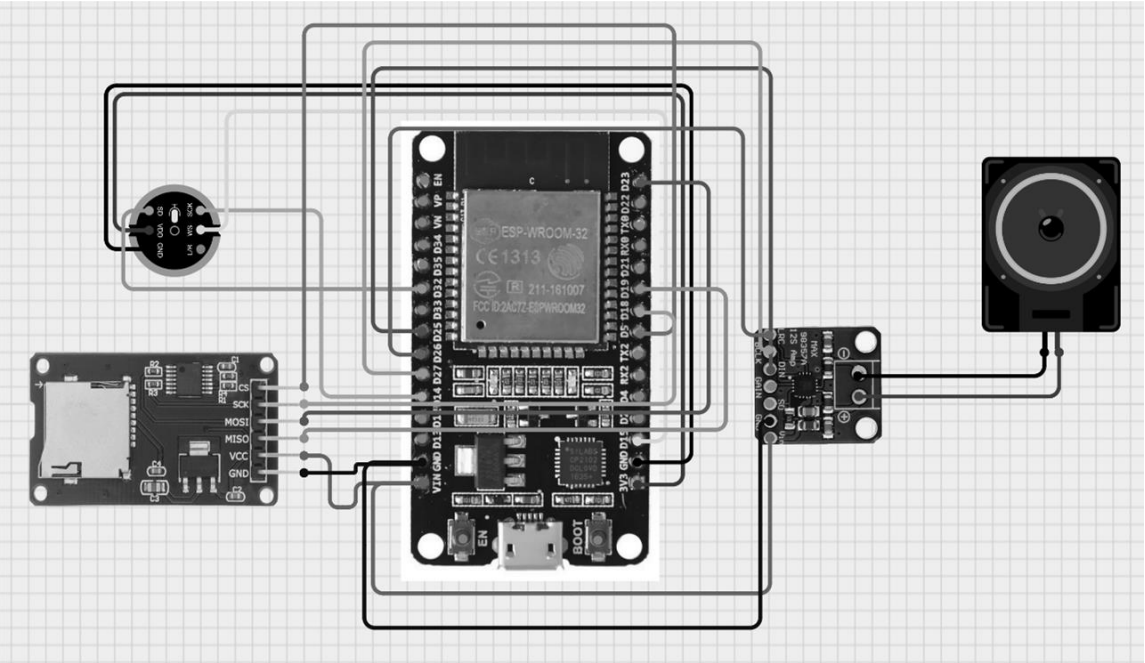

Рис. 3.2 — Принципова схема підключення

INMP441 є цифровим MEMS-мікрофоном, який забезпечує якісний запис із частотою дискретизації 8 кГц і 24-бітною глибиною. Він захоплює аудіо з навколишнього середовища у форматі PCM і передає його через інтерфейс I2S до ESP32.

MAX98357 є підсилювачем класу D із вихідною потужністю 3.2 Вт при навантаженні 4 Ом і напрузі 5 В. Він підтримує I2S для прямого підключення до ESP32, забезпечуючи ефективність до 92%. Усередині MAX98357 PCM-дані, отримані через I2S, модулюються в PWM-сигнал (імпульсно-широтно-модульований), який є послідовністю імпульсів із змінною шириною, що відповідає амплітуді вихідного аудіосигналу. Цей PWM-сигнал підсилюється і подається на вихід для динаміка, де низькочастотний фільтр динаміку згладжує імпульси, формуючи аналоговий аудіосигнал, який сприймається як звук.

Модуль Micro SD підтримує карти ємністю до тридцяти двох ГБ, дозволяючи

зберігати кілька годин аудіо у форматі WAV при частоті 8 кГц, і споживає 20-30 мА під час операцій читання/запису.

Архітектура ESP32 дозволяє розподіляти обчислювальні задачі між ядрами, що забезпечує одночасну обробку аудіопотоків, керування мережею та взаємодію з модулем зберігання, мінімізуючи затримки. З'єднання між компонентами реалізовано за допомогою перемичок, що надає можливість легко та швидко монтувати пристрій або змінювати використання виводів за потреби.

Схема підключення компонентів забезпечує їхню безперебійну взаємодію. ESP32 керує всіма периферійними пристроями, використовуючи I2S для аудіообробки та SPI для роботи з Micro SD. Мікрофон розташовано подалі від Wi-Fi модуля ESP32 та аудіопідсилювача MAX98357, щоб мінімізувати електромагнітні перешкоди.

На рисунку 3.3 зображено макет реалізованої апаратної частини.

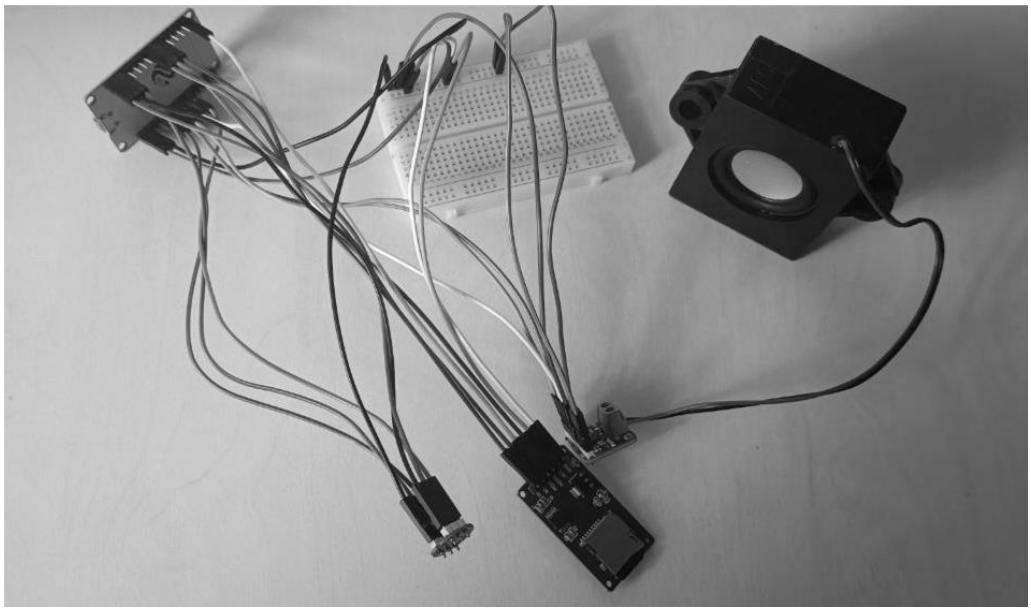

Рис. 3.3 — Макет пристрою

3.2 Програмна реалізація функціоналу пристрою

3.2.1 Огляд архітектури програми

Технічні вимоги до реалізації пристрою передбачають розробку фунціоналу запису, відтворення, потокової передачі аудіоданих і мережевої взаємодії з

клієнтами, тому програмна частина системи має модульну структуру, що включає підсистеми ініціалізації, мережевих операцій, обробки аудіо та діагностики.

Ініціалізація виконується у функції setup(), яка налаштовує мережеві сервери, аудіоінтерфейси та файлову систему. Основний цикл програми loop() координує роботу системи, викликаючи функції для обробки HTTP-запитів, виявлення пристрою через UDP, передачі файлів через TCP, потокової передачі аудіо, відтворення WAV-файлів і запису аудіо.

Багатозадачність забезпечується викликами функції yield(), яка передає керування планувальнику задач ESP32. Стан системи контролюється глобальними змінними, такими як isAudioStreaming, isPlaying і isRecording, які визначають активні режими роботи. Періодична діагностика виводить статус системи через серійний порт кожні 5 секунд.

Архітектура програми зображена на рисунку 3.4, де показано взаємодію основного циклу з модулями мережі, аудіо та діагностики.

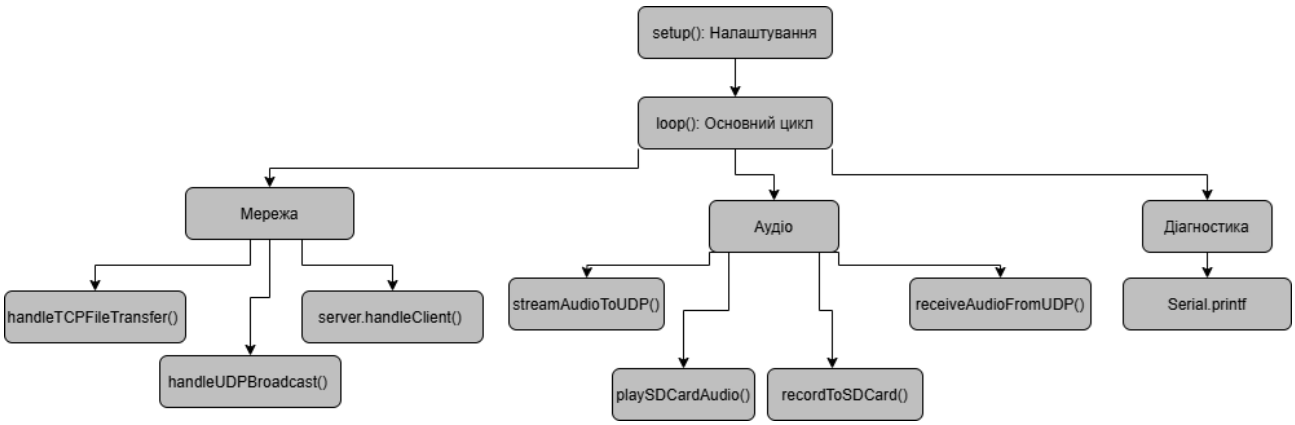

Рис. 3.4 — Блок-схема загальної структури програми

3.2.2 Ініціалізація та глобальні компоненти

Для реалізації функціоналу системи було використано бібліотеки WebServer для обробки HTTP-запитів, WiFi і WiFiUDP для роботи в режимі точки доступу і UDP-зв’язку, AudioTools для керування I2S-аудіо та SdFat для доступу до microSD через SPI-протокол.

У функції setup() ініціалізується WiFi у режимі точки доступу з SSID і паролем та призначається статична IP-адреса. I2S-інтерфейс налаштовується для мікрофона та підсилювача із частотою дискретизації 8 кГц, моно каналом і розрядністю 32 біти на вхід та 16 біт на вихід. Модуль microSD ініціалізується через інтерфейс SPI. HTTP-сервер запускається на порту 80 із запитами для керування аудіо та файлами, UDP-сервер — на портах 5555 для отримання і 5556 для відправки аудіопотоку, а TCP-сервер — на порту 8080 для передачі файлів. Серійний порт налаштовується зі швидкістю 115200 бод для виведення діагностичних повідомлень.

Глобальні компоненти включають константи, такі як частота дискретизації, значення портів, а також піни для апаратних компонентів. Змінні стану isAudioStreaming, isPlaying, isRecording керують режимами роботи, а об’єкти server, udpServer, tcpFileServer, i2s_in, i2s_out і audioBuffer забезпечують функціонал мережевих і аудіоопераций.

Детальний опис роботи блоку програми, який відповідає за ініціалізацію та глобальні компоненти:

#include <WiFi.h> — підключення бібліотеки для роботи з WiFi на ESP32, яка дозволяє налаштувати точку доступу і керувати мережевими з'єднаннями.

#include <WebServer.h> — підключення бібліотеки для створення вебсервера на порту 80, який обробляє HTTP-запити з телефону.

#include <SdFat.h> — підключення бібліотеки для роботи з MicroSD-картою через SPI.

#include <SPI.h> — підключення бібліотеки для роботи з SPI-інтерфейсом, який необхідний для зв'язку з MicroSD-картою.

#include <WiFiUdp.h> — підключення бібліотеки для роботи з UDP-протоколом, який використовується для потокової передачі аудіо.

#include <AudioTools.h> — підключення бібліотеки для обробки аудіо.

#define DEVICE_NAME "ESP32 Audio Device" — ініціалізація константи, яка задає назву пристрою для ідентифікації в мережі

#define FIRMWARE_VERSION "1.0.0" — ініціалізація константи, яка вказує

версію прошивки пристрою.

const char* WIFI_SSID = "ESP32_AudioDevice"; — ініціалізація константи, яка визначає ім'я точки доступу WiFi.

const char* WIFI_PASSWORD = "12345678"; — ініціалізація константи, яка задає пароль для точки доступу WiFi.

#define HTTP_PORT 80 — ініціалізація константи, яка вказує порт, на якому працює HTTP веб-сервер.

#define UDP_AUDIO_PORT 5555 — ініціалізація константи, яка вказує UDP-порт, який використовується для прийому аудіоданих із клієнта на ESP32 у режимі потокової передачі.

#define UDP_OUT_AUDIO_PORT 5556 — ініціалізація константи, яка вказує UDP-порт, який використовується для відправлення аудіоданих із клієнта на ESP32 у режимі потокової передачі.

#define UDP_BROADCAST_PORT 12345 — ініціалізація константи, яка вказує UDP-порт для широкомовних повідомлень, які використовуються для виявлення пристрою в мережі.

#define TCP_FILE_PORT 8080 — ініціалізація константи, яка вказує TCP-порт для передачі файлів.

#define SAMPLE_RATE 8000 — ініціалізація константи, яка визначає частоту дискретизації аудіо в герцах.

#define CHANNELS 1 — ініціалізація константи, яка вказує кількість аудіоканалів.

#define BITS_PER_SAMPLE_IN 32 — ініціалізація константи, яка визначає розрядність для вхідного аудіопотоку з мікрофона INMP441.

#define BITS_PER_SAMPLE_OUT 16 — ініціалізація константи, яка визначає розрядність для вихідного аудіопотоку на динамік через MAX98357.

#define I2S_BCLK_PIN 14 — ініціалізація константи, що визначає пін ESP32, який використовується для синхронізації передачі бітів даних між ESP32 і мікрофоном.

#define I2S_WS_PIN 15 — ініціалізація константи, що визначає пін ESP32,

яка визначає лівий або правий канал.

#define I2S_DATA_IN_PIN 32 — ініціалізація константи, що визначає пін ESP32, для прийому даних від мікрофона INMP441 через I2S.

#define I2S_DOUT_PIN 25 — ініціалізація константи, що визначає пін ESP32, який використовується для відправлення даних до підсилювача MAX98357 через I2S.

#define I2S_BCLK_OUT_PIN 27 — ініціалізація константи, що визначає пін ESP32, який синхронізує передачу даних до підсилювача.

#define I2S_LRC_OUT_PIN 26 — ініціалізація константи, що визначає пін ESP32, який визначає початок нового аудіосемплу для відтворення.

#define SD_CS_PIN 5 — ініціалізація константи, що визначає пін ESP32, який активує MicroSD-карту для зв’язку через SPI.

#define SD_MOSI_PIN 23 — ініціалізація константи, що визначає пін ESP32, який передає дані від ESP32 до MicroSD-карти.

#define SD_MISO_PIN 19 — ініціалізація константи, що визначає пін ESP32, який передає дані від MicroSD-карти до ESP32.

#define SD_SCK_PIN 18 — ініціалізація константи, що визначає пін ESP32, який забезпечує синхронізацію даних між ESP32 і SD-картою.

WebServer server(HTTP_PORT); — оголошення глобального об’єкту, який слухає HTTP-запити на порту 80 та обробляє запити від клієнта для керування пристроєм.

WiFiUDP udpServer; — оголошення глобального об’єкту, який використовується для відправки аудіоданих клієнту через порт 5555.

WiFiUDP udpAudioReceiver; — оголошення глобального об’єкту який, використовується для отримання UDP-пакетів з аудіоданими від клієнта на порту 5556 у режимі стрімінгу.

WiFiServer tcpFileServer(TCP_FILE_PORT); — оголошення глобального об’єкту, що створює TCP-сервер для прийому файлів, які клієнт завантажує на MicroSD-карту ESP32.

WiFiClient tcpClient; — оголошення глобального об’єкту, який

використовується для обробки з’єднання з телефоном під час передачі файлів.

SdFat SD; — оголошення глобального об’єкту, який надає доступ до файлової системи MicroSD-карти через бібліотеку SdFat.

SPIClass sdSPI(HSPI); — оголошення глобального об’єкту, який налаштовує SPI-інтерфейс для зв’язку з microSD-кардридером.

I2SStream i2s_out; — оголошення глобального об’єкту, що налаштовує вихідний I2S-потік для передачі аудіоданих на підсилювач MAX98357, який підключений до динаміка.

I2SStream i2s_in; — оголошення глобального об’єкту, який налаштовує вхідний I2S-потік для отримання аудіоданих із мікрофона INMP441.

WAVDecoder wavDecoder; — оголошення глобального об’єкту, що використовується для декодування аудіофайлів у форматі WAV, які зберігаються на SD-карті, перед їх відтворенням через I2S-вихід.

EncodedAudioStream wavStream(&i2s_out, &wavDecoder); — оголошення глобального об’єкту, який зв’язує декодер WAV із I2S-виходом, щоб аудіодані з SD-карти відтворювалися через MAX98357.

StreamCopy copier; — використовується для копіювання даних із одного потоку в інший, зокрема для передачі декодованих WAV-даних із SD-карти в I2S-вихід під час відтворення.

FsFile audioFile; — глобальний об’єкт, що ініціалізує файл на SD-карті, який відкривається для читання або запису.

RingBuffer<uint8_t> audioBuffer(8192); — створення буферу для тимчасового зберігання аудіоданих, отриманих через UDP.

String currentAudioFile = ""; — зберігає шлях до поточного аудіофайлу, який відтворюється або записується на SD-карту

bool isAudioStreaming = false; — визначає активний режим потокової передачі аудіо з мікрофона на клієнт через UDP. Якщо true, ESP32 надсилає аудіодані клієнту.

bool isAudioListening = false; — визначає активний режим потокового отримання аудіо, відправленого клієнтом через UDP, для відтворення через

динамік. Якщо true, ESP32 приймає аудіопакети та виводить їх.

bool isPlaying = false; — визначає чи відтворюється аудіофайл із SD-карти через динамік. Якщо true, ESP32 читає WAV-файл і передає його на I2S-вихід.

bool isRecording = false; — визначає активний режим запису аудіо з клієнту на SD-карту. Якщо true, дані записуються у WAV-файл.

String pendingFilename = ""; — зберігає ім’я файлу, який планується завантажити на SD-карту через TCP.

bool hasSDCard = false; — визначає чи успішно ініціалізована SD-карта.

uint64_t sdCardCapacity = 0; — зберігає загальну ємність SD-карти в байтах.

uint64_t sdCardFreeSpace = 0; — зберігає обсяг вільного місця на SD-карті в байтах.

IPAddress lastClientIP; — зберігає IP-адресу останнього клієнта, який ініціював стрімінг або прослуховування аудіо через UDP.

uint16_t lastClientPort = 0; — зберігає порт клієнта для UDP-стримінгу.

int currentChannels = CHANNELS; — зберігає поточну кількість аудіоканалів для відтворення.

char fileList[128] = "[]"; — зберігає список імен аудіофайлів у директорії /AudioFiles у форматі JSON.

Також у скетчі були визначені прототипи функцій, які оголошують основні функції, що використовуються для налаштування, обробки аудіо, роботи з мережею та SD-картою. Прототипи функцій потрібні для того, щоб компілятор знав про існування цих функцій перед їх викликом, особливо якщо вони визначені нижче в коді.

3.2.3 Мережева взаємодія

Мережева взаємодія аудіопристрою забезпечує керування функціями, потокову передачу аудіоданих і обмін файлами через WiFi у режимі точки доступу. Для цього використовуються протоколи HTTP для керування, UDP для виявлення пристрою та стрімінгу і TCP для передачі файлів, які реалізовані через бібліотеки WebServer, WiFi і WiFiUDP.

HTTP-сервер, створений за допомогою бібліотеки WebServer, працює на порту 80 і обробляє запити синхронно через метод server.handleClient(), викликаний у циклі loop(). Сервер підтримує API-маршрути, такі як /api/audio/files/play для відтворення WAV-файлів, /api/audio/files/upload для передачі файлів через TCP та /api/audio/streaming/startsending для запуску стрімінгу. Синхронна обробка HTTP-запитів може викликати затримки в операціях реального часу, таких як аудіострімінг, що компенсується викликами функції yield() для забезпечення багатозадачності.

Виявлення пристрою клієнтом реалізовано через функцію handleUDPBroadcast(), яка на UDP-запити надсилає IP-адресу клієнту. Потокова передача аудіо виконується функціями streamAudioToUDP() і receiveAudioFromUDP(). Перша надсилає дані з I2S-входу до клієнта, використовуючи буфер audioBuffer для стабільності, тоді як друга приймає аудіопотік для відтворення. Передача WAV-файлів здійснюється через TCP функцією handleTCPFileTransfer(), яка зберігає отримані дані в директорії /AudioFiles на microSD. Процес ініціюється HTTP-запитом до /api/audio/files/upload.

Мережева взаємодія пристрою зображена на рисунку 3.5, де показано взаємодію клієнта з HTTP, UDP і TCP-серверами, а також використання audioBuffer і microSD для стрімінгу та збереження файлів.

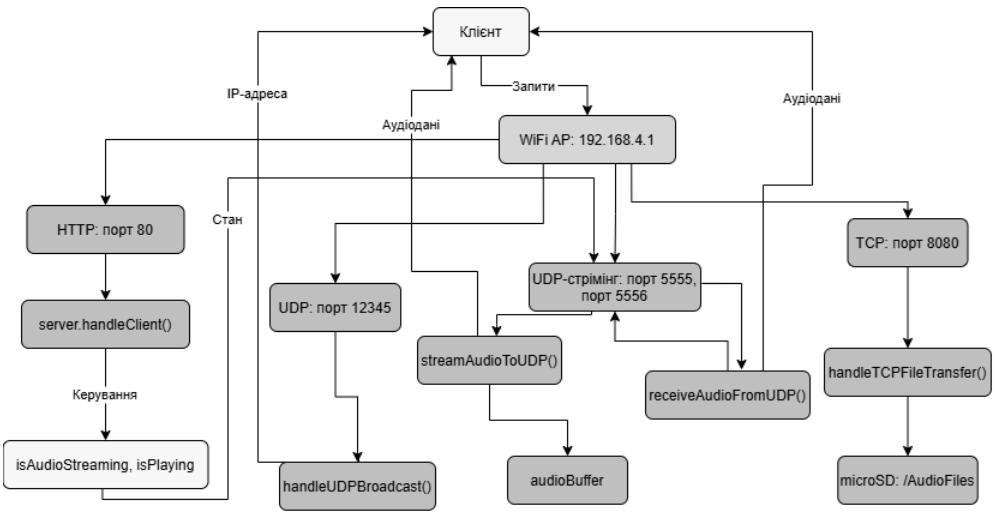

Рис. 3.5 — Блок-схема мережевої взаємодії пристрою

Детальний опис роботи блоку програми, який відповідає за реалізацію мережевих взаємодій:

void setupWiFi() — оголошення функції, яка налаштовує ESP32 як точку доступу WiFi, щоб телефон міг підключитися до нього для керування.

WiFi.mode(WIFI_AP); — встановлення режиму роботи WiFi-модуля ESP32 у режим точки доступу.

WiFi.softAP(WIFI_SSID, WIFI_PASSWORD); — створення точки доступу WiFi з ім’ям і паролем.

if (!MDNS.begin("esp32audio")) { Serial.println("Error setting up MDNS responder!"); } — ініціалізує mDNS, який дозволяє клієнту знаходити ESP32 у локальній мережі та виводить повідомлення "Error setting up MDNS responder!" у серійний порт, якщо ініціалізація mDNS не вдалася.

udpServer.begin(UDP_BROADCAST_PORT); — налаштовує ESP32 для прийому UDP-пакетів, які використовуються для виявлення пристрою в мережі.

void setupUDPAudio() — функція, яка налаштовує два UDP-сервери для потокового аудіо: один для надсилання аудіо з мікрофона на клієнт, інший для прийому аудіо від клієнта для відтворення через динамік.

udpServer.begin(UDP_AUDIO_PORT); — налаштовує ESP32 для надсилання аудіоданих із мікрофона через UDP на клієнт.

udpAudioReceiver.begin(UDP_OUT_AUDIO_PORT); — налаштовує ESP32 для прийому аудіоданих від клієнта через UDP для відтворення через динамік.

void setupTCPFileServer() — функція, яка налаштовує TCP-сервер для прийому файлів від клієнта для запису на SD-карту.

tcpFileServer.begin(); — налаштовує ESP32 для прийому TCP-з’єднань від клієнта, через які передаються файли

void setupWebServer() — функція, яка налаштовує HTTP-сервер для обробки запитів від клієнта, забезпечуючи API для керування пристроєм, стримінгу, відтворення, запису та роботи з файлами.

server.on("/api/status", HTTP_GET, []() — визначає маршрут /api/status для GET-запитів. Лямбда-функція повертає текстовий рядок "ok" із кодом статусу 200.

Це надає простий спосіб перевірки, чи пристрій увімкнений і вебсервер працює. Клієнт може надіслати запит, щоб переконатися, що ESP32 доступний.

server.on("/api/config", HTTP_GET, []() {String configResponse = "{\"DeviceName\":\"" + String(DEVICE_NAME) +"\",\"FirmwareVersion\":\"" + String(FIRMWARE_VERSION) "\",\"HasSDCard\":" + String(hasSDCard ? "true" : "false" ",\"SDCardCapacity\":" + String(sdCardCapacity) + ",\"SDCardFreeSpace\":" + String(sdCardFreeSpace) + "}"; — визначає маршрут /api/config для GET-запитів. Формує JSON-рядок із конфігурацією пристрою та повертає його з кодом 200. Це надає клієнту інформацію про пристрій: ім’я, версію прошивки, наявність SD-карти, її загальну ємність і вільний простір.

server.on("/api/audio/streaming/startsending", HTTP_GET, []() — визначає маршрут для запуску потокової передачі аудіо з мікрофона на клієнт через UDP. Він перевіряє, чи стрімінг ще не активний, і якщо не активний, то зупиняє відтворення, запис і прослуховування. Встановлює isAudioStreaming = true. Далі зберігає IP-адресу клієнта і порт 5555 для надсилання аудіо. Вмикає I2S-вхід для мікрофона. Повертає JSON із статусом "streaming_started". Якщо стрімінг уже активний, повертає "already_streaming".Це запускає режим рації, де ESP32 надсилає аудіо з мікрофона на телефон через UDP та зупиняє інші операції для уникнення конфліктів.

server.on("/api/audio/streaming/stopsending", HTTP_GET, []() — визначає маршрут для зупинки потокового мовлення. Якщо стрімінг активний, то встановлює isAudioStreaming = false, вимикає I2S-вхід. Далі повертає JSON із статусом "streaming_stopped". Якщо стрімінг не активний, повертає "not_streaming". Це зупиняє надсилання аудіо з мікрофона на клієнт.

server.on("/api/audio/streaming/startlistening", HTTP_GET, []() — визначає маршрут для запуску прослуховування аудіо від клієнта через UDP. Якщо прослуховування не активне, то зупиняє стрімінг, запис і відтворення. Встановлює isAudioListening = true. Далі зберігає IP-адресу клієнта і порт 5556 для прийому аудіо. Вмикає I2S-вихід для динаміка. Повертає JSON із статусом "listening_started". Якщо прослуховування вже активне, повертає

"already_listening". Це запускає режим потокової передачі, де ESP32 приймає аудіо від телефону і відтворює його через MAX98357 і зупиняє інші операції.

server.on("/api/audio/files/list", HTTP_GET, []() — визначає маршрут для отримання списку аудіофайлів із SD-карти. Він перевіряє наявність SD-карти. Якщо карти немає, повертає помилку 500 із JSON "no_sd_card". Якщо карта є то повертає кешований список файлів у форматі JSON із кодом 200. Це надає клієнту список WAV-файлів у директорії /AudioFiles для відображення в інтерфейсі.

server.on("/api/audio/files/play", HTTP_GET, []() — визначає маршрут для відтворення аудіофайлу з SD-карти. Перевіряє наявність параметра name. Якщо відсутній, повертає помилку 400. Формує шлях до файлу. Якщо файл не існує, повертає 404. Зупиняє інші операції: стрімінг, прослуховування, запис, відтворення. Відкриває файл у режимі читан. Читає 44-байтний WAV-заголовок, витягує частоту дискретизації, кількість каналів і розрядність. Якщо канали або частота не збігаються з поточними, то переініціалізовує I2S-вихід із новими параметрами. Далі ініціалізує wavStream і copier для декодування та копіювання WAV-даних. Встановлює isPlaying = true, зберігає ім’я файлу в currentAudioFile, увімкнює I2S-вихід. Повертає JSON із статусом "playing". Це реалізовує відтворення WAV-файл із SD-карти через MAX98357.

3.2.4 Обробка аудіоданих та керування файлами

Обробка аудіоданих забезпечує запис, відтворення, потокову передачу та збереження звукових даних на аудіопристрої. Для цього використовується I2S-інтерфейс із частотою дискретизації 8 кГц у моно режимі, реалізований через бібліотеку AudioTools для роботи з мікрофоном INMP441 і підсилювачем MAX98357, а також бібліотеку SdFat для доступу до WAV-файлів на microSD.

Запис аудіо здійснюється функцією recordToSDCard(), яка отримує дані від клієнта і зберігає їх у WAV-файлі в директорії /AudioFiles. Для сумісності із стандартом WAV виконується конвертація 32-бітних даних у 16-бітний формат із додаванням заголовка, що містить частоту, кількість каналів і розрядність. Відтворення аудіо реалізовано функцією playSDCardAudio(), яка читає WAV-файл

із microSD і надсилає його на MAX98357 через I2S-вихід. Буферизація даних забезпечує стабільне відтворення без переривань.

Потокова передача аудіо виконується функціями streamAudioToUDP() і receiveAudioFromUDP(). Перша надсилає дані з INMP441 до клієнта через UDP, використовуючи буфер audioBuffer для зменшення затримок, тоді як друга приймає UDP-потік і направляє його на MAX98357. Змінні стану isRecording, isPlaying і isAudioStreaming координують режими роботи, уникаючи конфліктів між операціями.

Потоки аудіоданих зображено на рисунку 3.6, де показано рух даних між INMP441, MAX98357, microSD і UDP-клієнтом.

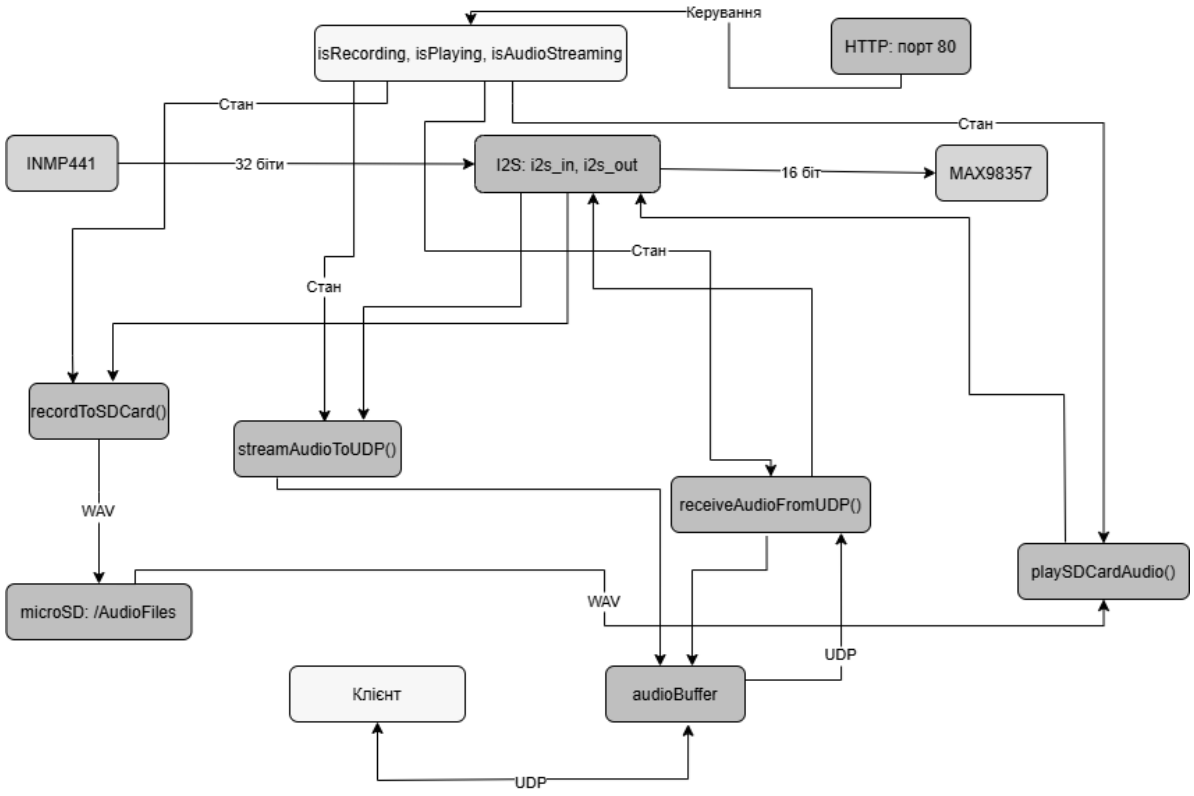

Рис. 3.6 — Блок-схема потоків аудіоданих

Детальний опис роботи блоку програми, який відповідає обробку аудіоданих та керування файлами:

void setupAudio() — функція, яка налаштовує I2S-інтерфейси для вхідного і вихідного аудіопотоків, готуючи пристрій до запису, відтворення та стрімінгу аудіо.

auto config_out = i2s_out.defaultConfig(TX_MODE); — отримує стандартну конфігурацію для вихідного I2S-потоку. TX_MODE вказує, що цей потік призначений для відправлення даних на зовнішній пристрій.

config_out.port_no = I2S_NUM_0; — вказує, що вихідний потік використовує I2S-порт 0 на ESP32. ESP32 має два I2S-порти, і порт 0 тут використовується для вихідного аудіо, задля уникнення переривань.

config_out.pin_bck = I2S_BCLK_OUT_PIN; — встановлює пін ESP32, який генерує тактовий сигнал для синхронізації бітів даних, що передаються до MAX98357. BCLK — це тактовий сигнал, який синхронізує передачу аудіоданих.

config_out.pin_ws = I2S_LRC_OUT_PIN; — встановлює пін ESP32, який вказує початок нового аудіосемплу. У монорежимі WS визначає початок кожного семплу.

config_out.pin_data = I2S_DOUT_PIN; — встановлює пін ESP32, через який передаються цифрові аудіодані до підсилювача MAX98357.

config_in.buffer_count = 8; — визначає кількість внутрішніх буферів у драйвері I2S для зберігання аудіоданих перед їх передачею.

void playSDCardAudio() — Відтворює WAV-файл із SD-карти через MAX98357, використовуючи об’єкти copier і wavStream для декодування та передачі аудіо на I2S-вихід.

if (isPlaying) — перевіряє, чи активне відтворення та виконує його, лише якщо воно активне.

size_t bytesCopied = copier.copy(); — читає дані з WAV-файлу, декодує їх за допомогою wavStream і передає на I2S-вихід для відтворення через MAX98357.

if (!bytesCopied && !audioFile.available()) — визначає, чи завершилося відтворення файлу.

stopPlayback(); — завершує відтворення, очищаючи ресурси, якщо файл закінчився.

void writeWAVHeader(FsFile &file, int sampleRate, int channels, int bitsPerSample) — функція, що формує 44-байтний заголовок WAV-файлу, який описує формат аудіо, і записує його в початок файлу на SD-карті.

uint8_t header[44]; — оголошення масиву, який використовується для зберігання заголовка WAV-файлу перед його записом у файл.

memset(header, 0, 44); — очищає масив, щоб уникнути невизначених даних у неініціалізованих полях заголовка.

memcpy(header, "RIFF", 4); — вказує, що файл є RIFF (Resource Interchange File Format), який містить WAV-данні. Це ідентифікатор першого блоку.

(uint32_t)&header[4] = 36; — встановлює розмір усього файлу мінус 8 байтів. Значення 36 є заповнювачем, оскільки реальний розмір файлу невідомий на момент запису заголовка.

memcpy(header + 8, "WAVE", 4); — вказує, що RIFF-файл є WAV-файлом.

memcpy(header + 12, "fmt ", 4); — ідентифікує підблок fmt, який описує формат аудіо.

(uint32_t)&header[16] = 16; — вказує розмір підблоку fmt (16 байтів для PCM).

(uint16_t)&header[20] = 1; — вказує формат аудіо: 1 означає PCM без стиснення.

(uint16_t)&header[22] = channels; — вказує кількість аудіоканалів.

(uint32_t)&header[24] = sampleRate; — вказує частоту дискретизації в герцах.

(uint16_t)&header[34] = bitsPerSample; — вказує розрядність одного семплу 16 біт.

memcpy(header + 36, "data", 4); — ідентифікує підблок data, який містить аудіодані.

(uint32_t)&header[40] = 0; — встановлює розмір аудіоданих у байтах як заповнювач, оскільки обсяг даних невідомий на момент запису заголовка.

file.write(header, 44); — зберігає сформований WAV-заголовок у початок файлу на SD-карті.

У підсумку, реалізована програмна частина системи виконує усі поставлені функції, а саме дозволяє ефективно передавати аудіо через UDP і керувати файлами на SD-карті та забезпечує стабільну роботу.

4 ОЦІНКА РЕЗУЛЬТАТІВ ТА ПЕРСПЕКТИВИ РОЗВИТКУ

4.1 Тестування та верифікація системи

Для перевірки працездатності розробленого комплексу було проведено тестування його апаратних і програмних компонентів на відповідність технічним вимогам. Тестувалися функції запису аудіо через мікрофон INMP441, відтворення через підсилювач MAX98357 із динаміком, потокової передачі через UDP, збереження та відтворення файлів на MicroSD через TCP і керування загальним функціоналом через HTTP.

Для перевірки коректності роботи мікрофона INMP441 було розроблено тестовий скетч та записано тестовий запис голосу із частотою 8 кГц, моно, 16 біт. Якість звуку оцінювалася за допомогою онлайн спектрального аналізатора, який показав основну частоту в районі 4 кГц, що є доволі хорошим результатом, задовольняючи теорему Котельникова-Шеннона (рис. 4.1).

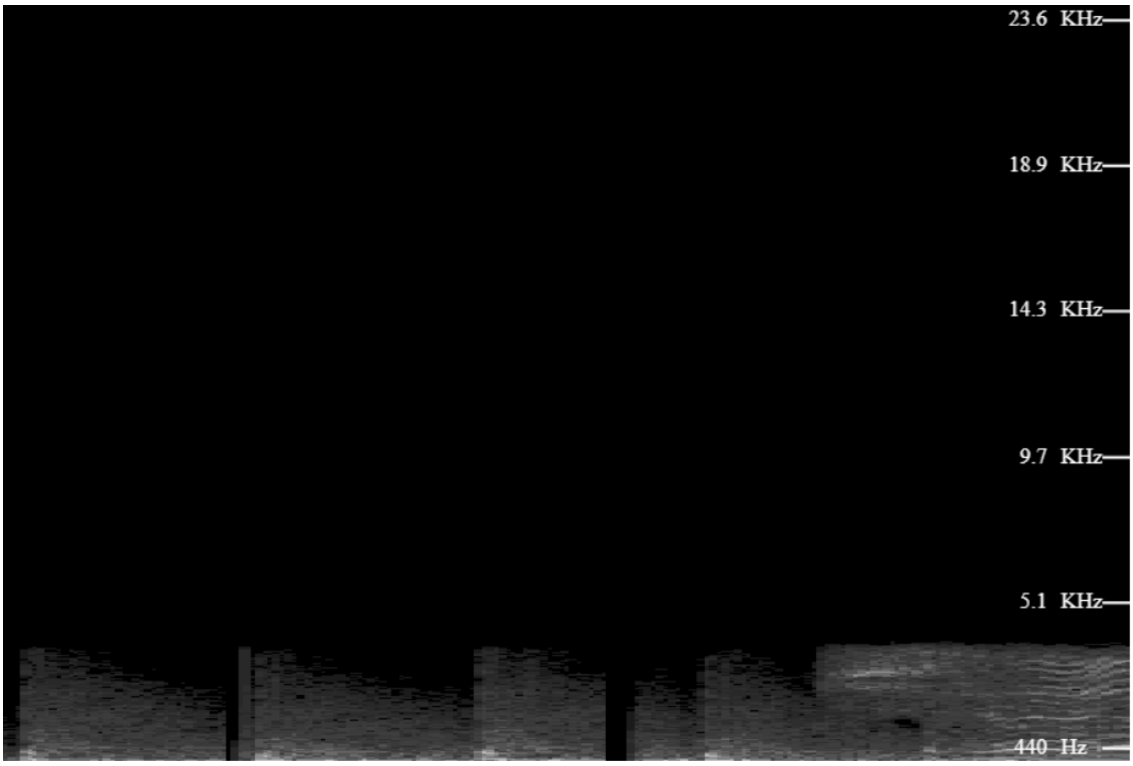

Рис. 4.1 — Тестування файлу за допомогою спектроаналізатора

Підсилювач MAX98357 із динаміком тестувався за допомогою відтворення WAV-файлів із microSD і UDP-потоку. Для перевірки якості та гучності звуку було

відтворено тестову музичну мелодію та запис людського голосу. Гучність вимірювалася смартфоном із програмою-шумоміром на відстані 1 м, досягнувши 80-84 дБ при максимальній гучності (рис. 4.2). Спектральний аналіз відтвореного сигналу зафіксував частотний діапазон в районі 4 кГц, із незначними спотвореннями. Стабільність I2S-сигналу перевірялася серійним монітором протягом 10 хвилин. Особливих відхилень виявлено не було. Споживання струму MAX98357 у режимі відтворення склало ≈100 мА, у спокої — 1 мА.

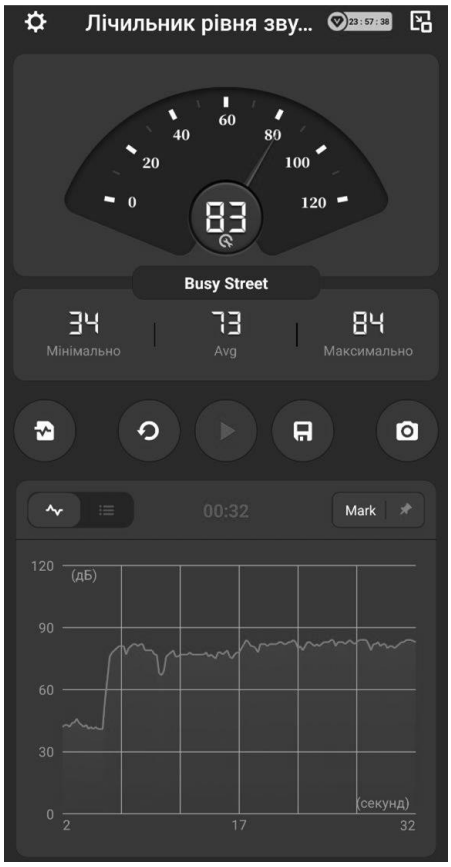

Рис. 4.2 — Тестування гучності підсилювача за допомогою шумоміру

Для перевірки роботи модуля MicroSD було проведено запис двадцяти файлів розбігом до п’ятдесяти МБ. Усі операції зчитування та запису відбувалися без помилок та затримок, а цілісність файлів було перевірено шляхом повторного зчитування та прослуховування.

Програмне тестування проводилося за допомогою серійного монітора 115200 бод і клієнтського застосунка на смартфоні. Воно охоплювало перевірку працездатності обробки HTTP-запитів та відтворення основних функцій

recordToSDCard(), playSDCardAudio(), streamAudioToUDP() і handleTCPFileTransfer(), перевіряючи коректність їх виконання. За результатами логування зі сторони серійного монітора та клієнтського застосунку, помилок у роботі системи виявлено не було (рис. 4.3).

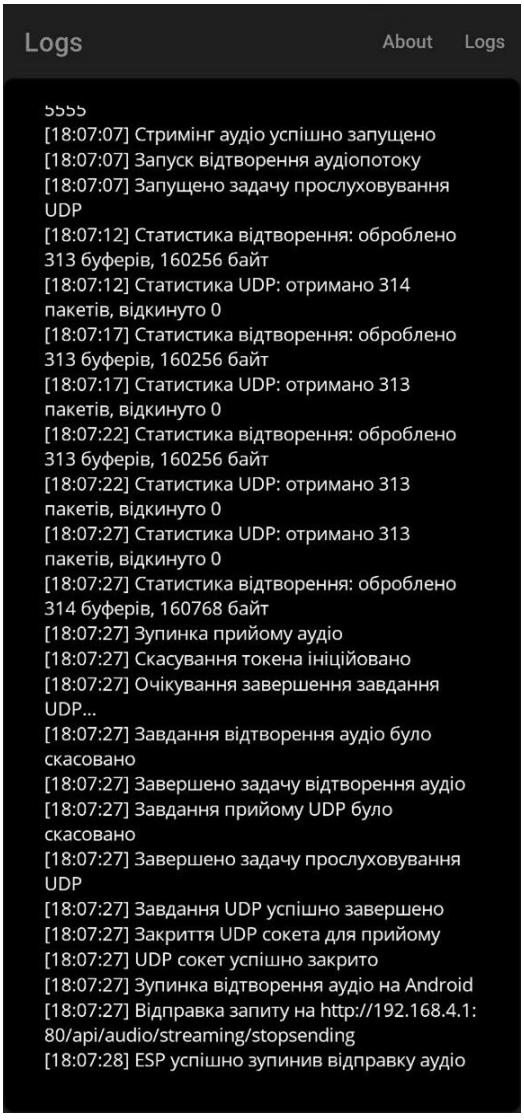

Рис. 4.3 — Логування роботи системи зі сторони застосунку

Отже, за результатами проведеного тестування, можна заявити, що розроблена система повністю відповідає технічним вимогам до якості звуку, затримок, стабільності, енергоефективності і компактності, та може бути інтегрована в розумного робота. Комплекс успішно виконує функції відтворення файлу із MicroSD-карти, стрімінгу у реальному часі та передачі файлів через TCP. Це підтверджує досягнення цілей проєкту, описаних у розділі 2.

4.2 Перспективи розвитку комплексу

Розроблений аудіопристрій забезпечує запис через INMP441, відтворення через MAX98357, стрімінг через UDP, збереження та передачу файлів на MicroSD і керування через HTTP, що підтверджено тестуванням. Однак обмеження, такі як затримки до ста мс при одночасній обробці HTTP і UDP, частотний діапазон до чотирьох кГц через частоту дискретизації 8 кГц, гучність 70-75 дБ і втрата UDP-пакетів, вказують на потенціал для вдосконалення. Тому необхідно розглянути деякі покращення, які змогли б забезпечити підвищення продуктивності, якості звуку та розширення функціональності.

Для розширення частотного діапазону до восьми кГц пропонується збільшити частоту дискретизації до шістнадцяти кГц, що покращить якість голосу та музики. Заміна динаміка на модель із вищою потужністю дозволить досягти гучності вищої за 80 дБ, зменшивши спотворення. Додавання другого мікрофона INMP441 або стереоаналога забезпечить стереозапис, який покращить якість звуку при спілкуванні. Також гарним рішенням покращення звуку є застосування фільтрів високих та низьких частот та алгоритмів шумозаглушення, які можуть зменшувати або й взагалі нівелювати наявність шумів.

Для оптимізації обчислювальних потужностей та зменшення затримок можна здійснити перехід на асинхронний веб-сервер, який зменшить затримки HTTP-обробки до десяти мс, усунувши конфлікти зі стрімінгом. Застосування DMA для I2S-інтерфейсу знизить навантаження на CPU, підвищивши стабільність відтворення.

Підтримка додаткових аудіоформатів є також важливим напрямом розвитку, оскільки поточна конфігурація обмежується PCM-форматом у WAV-файлах, що забезпечує простоту обробки, але генерує великі обсяги даних і не підтримує стиснення. Впровадження кодеків, таких як MP3 або Opus, дозволить зменшити розмір аудіофайлів, що критично для зберігання на SD-карті з обмеженою ємністю і передачі через Wi-Fi в умовах низької пропускної здатності. Наприклад, MP3 із бітрейтом 64 кбіт/с зменшує обсяг даних удвічі порівняно з PCM при 128 кбіт/с,

зберігаючи прийнятну якість для голосового зв’язку. Opus, завдяки адаптивному стисненню, може забезпечити ще кращу якість при нижчому бітрейті, що ідеально для нестабільних мереж. Однак декодування MP3 або Opus потребує додаткових обчислювальних ресурсів, що є викликом для ESP32 із 520 КБ SRAM і частотою процесора 240 МГц. Для реалізації цього функціоналу можна використати оптимізовані бібліотеки, такі як libmad для MP3 або opuslib, адаптовані для ESP32, із розподілом задач між двома ядрами: одне для декодування, інше для обробки мережі та I2S. Інтеграція цих кодеків дозволить відтворювати стиснені файли з SD-карти, зменшуючи час запису та підвищуючи ефективність зберігання. Крім того, підтримка форматів із вищою частотою дискретизації, наприклад, 16-44.1 кГц, розширить можливості системи для відтворення музики, хоча це вимагатиме оптимізації пропускної здатності Wi-Fi та буферизації, щоб уникнути переривань.

Також у межах масштабування системи пропонується забезпечити інтеграцію з хмарними сервісами, що забезпечить віддалене збереження аудіофайлів. Впровадження AI-алгоритмів дасть змогу розпізнавати голосові команди або виявляти звукові події, що дозволить пристрою конкурувати із IoT-комплеками розумного будинку, які описані у розділі 1.

Розвиток системи в запропонованих напрямах підвищить її конкурентоспроможність, розширить функціональність і забезпечить адаптацію до сучасних вимог IoT і мультимедійних технологій.

ВИСНОВКИ

У ході виконання бакалаврської дипломної роботи було розроблено та протестовано кіберфізичну систему для дистанційного передавання аудіо. Система забезпечує відтворення аудіофайлів, запис аудіо на карту пам’яті, передачу стрімінгового звуку у режимі реального часу, а також керування всіма функціями через Android-застосунок по бездротовому з’єднанню у локальній мережі.

У процесі роботи були досягнуті такі результати:

Проведено порівняльний аналіз апаратних платформ ESP32, STM32 і Raspberry Pi для визначення найбільш придатної для аудіообробки в умовах обмежених ресурсів. Обґрунтовано вибір ESP32 як найбільш збалансованої платформи за критеріями енергоефективності, підтримки бездротових інтерфейсів, вартості та простоти інтеграції.

Досліджено доцільність обміну даними через протоколи UDP, TCP і HTTP, що дозволило визначити їхні переваги та недоліки для стрімінгу, передачі файлів і керування. Розроблені підходи до реалізації двостороннього аудіострімінгу через UDP забезпечили низькі затрики, а механізми TCP — надійну передачу файлів.

Реалізовано обмін інформацією між мікроконтролером та клієнтським застосунком з використанням протоколу HTTP для передачі команд та керування режимами роботи пристрою.

Впроваджено функцію запису та відтворення аудіо з MicroSD-карти.

Проведено тестування системи, яке підтвердило працездатність розробленого комплексу, адекватну якість звуку та коректну реакцію на команди з мобільного пристрою.

Отже, розроблена система цілком відповідає умовам технічного завдання і може бути використана, як додатковий самостійний елемент розумного робота.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бондарчук О. Б., Дацюк В. В., Богомолов С. В., Мартинюк Т. Б. Дистанційна передача аудіо в IoT. " LIV Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2025)": вебсайт. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/24413

2. Cyber-Physical Systems: An Overview : веб-сайт. URL: https://www.fortinet.com/resources/cyberglossary/cyber-physical-systems (дата звернення: 03.05.2025).

3. C. Greer, M. Burns, D. Wollman, and E. Griffor, «Cyber-Physical Systems and Internet of Things». *NIST Special Publication 1900-202*, Mar. 2019. P. 61.

4. Комп’ютерні мережі : підручник / [Азаров О. Д., Захарченко С. М., Кадук О. В. та ін.]. − Вінниця : ВНТУ, 2020. − 378 с.

5. WebRTC Wikipedia : веб-сайт. URL: https://uk.wikipedia.org/wiki/WebRTC (дата звернення: 03.05.2025).

6. WebRTC технологія для безпеки корпоративного спілкування : веб-сайт. URL: https://corewin.ua/blog/webrtc_nextcloud_talk (дата звернення: 03.05.2025).

7. ESP32: Характеристики та специфікації : веб-сайт. URL: http://esp32.net/ (дата звернення: 04.05.2025).

8. ESP32 Series Datasheet : веб-сайт. URL: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf (дата звернення: 04.05.2025).

9. What Is Raspberry Pi? Models, Features, and Uses : веб-сайт. URL: https://www.spiceworks.com/tech/networking/articles/what-is-raspberry-pi/ (дата звернення: 05.05.2025).

10. Що таке мікрокомп'ютери? Raspberry Pi: веб-сайт. URL: https://e-server.com.ua/uk/poradi/shho-take-mikrokompiuteri-rozpovidajemo-na-prikladi-raspberry-pi (дата звернення: 05.05.2025)

11. STM32 для новачків: обираємо плати, середовища та перші кроки : веб-

сайт. URL: https://dou.ua/forums/topic/52283/ (дата звернення: 06.05.2025).

12. Turchet, L., Fischione, C., Essl, G., & Keller, P. «Internet of Musical Things: Vision and Challenges». *IEEE 2169-3536*, Sep 2018. P. 61994–62017.

13. IEEE 802.11n Wikipedia : веб-сайт. URL: https://en.wikipedia.org/wiki/IEEE_802.11n-2009 (дата звернення: 07.05.2025).

14. The I2S Protocol and Why Digital Audio is Everywhere : веб-сайт. URL: https://www.keysight.com/blogs/en/tech/bench/2022/04/29/the-i2s-protocol-and-why-digital-audio-is-everywhere (дата звернення: 07.05.2025).

15. INMP441 : веб-сайт. URL: https://www.alldatasheet.com/datasheet-pdf/view/1244625/ETC1/INMP441.html (дата звернення: 08.05.2025)

16. INMP441 MEMS High Precision Omnidirectional Microphone Module : веб-сайт. URL: https://components101.com/modules/inmp441-mems-omnidirectional-microphone (дата звернення: 08.05.2025).

17. MAX98357 : веб-сайт. URL: https://www.alldatasheet.com/datasheet-pdf/view/623796/MAXIM/MAX98357A.html (дата звернення: 08.05.2025)

18. MAX98357 Low Cost PCM 3W Class D Amplifier : веб-сайт. URL: https://components101.com/ics/max98357-low-cost-pcm-3w-class-d-amplifier (дата звернення: 08.05.2025).

19. Micro SD Card Adapter Module : веб-сайт. URL: https://components101.com/modules/micro-sd-card-module-pinout-features-datasheet-alternatives (дата звернення: 09.05.2025).

20. Arduino Integrated Development Environment (IDE) : веб-сайт. URL: https://docs.arduino.cc/software/ide-v1/tutorials/arduino-ide-v1-basics/ (дата звернення: 09.05.2025).

21. Arduino Audio Tools : веб-сайт. URL: https://github.com/pschatzmann/arduino-audio-tools?tab=readme-ov-file (дата звернення: 10.05.2025).

22. SdFat Library : веб-сайт. URL: https://github.com/greiman/SdFat (дата звернення: 10.05.2025).

ДОДАТОК А

Технічне завдання

Міністерство освіти та науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н. проф._____Азаров О. Д.
«21» березня 2025 року

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання комплексної бакалаврської кваліфікаційної роботи
«Апаратно-програмний комплекс для дистанційного передавання аудіо. Частина 1. Кіберфізичний пристрій»

08-54.КБКР.036.00.000 ТЗ

Керівник роботи д.т.н., проф. каф. ОТ

_____Мартинюк Т. Б._____

Студент групи 1КІ-21б

_____Бондарчук О. Б._____

ВНТУ 2025

1 Підстава для виконання комплексної бакалаврської кваліфікаційної роботи (КБКР)

1.1 Актуальність теми полягає в зростанні попиту на системи дистанційного передавання аудіо, які застосовуються в розумних будинках, освіті, телекомунікаціях та мультимедійних додатках. Розробка таких систем на базі мікроконтролерів, зокрема ESP32, є актуальною завдяки потребі в надійних і енергоефективних рішеннях для обробки аудіоданих у локальних мережах. Дослідження проблематики та створення програмно-апаратних комплексів сприяють розвитку КФС-технологій.

1.2 Наказ ВНТУ №97 про затвердження теми БКР від 20.03.2025.

2 Мета КБКР і призначення розробки

2.1 Мета роботи — створення системи для дистанційної передачі аудіо в локальній мережі на базі мікроконтролера ESP32, яка забезпечує двосторонній аудіострімінг, передачу, запис та відтворення аудіофайлів і керування пристроєм за допомогою за допомогою Android-застосунку, а також дослідження проблематики реалізації таких систем.

2.2 Призначення розробки — створення програмно-апаратного комплексу для обробки та передачі аудіоданих, що може використовуватися в освітніх, наукових і комерційних проєктах для вивчення мережевих протоколів і розробки аудіосистем.

3 Вихідні дані для виконання КБКР

3.1 Базові знання мікроконтролерних систем, зокрема ESP32.

3.2 Принципи роботи аудіомодулів (мікрофонів, динаміків) і інтерфейсів.

3.3 Технічні характеристики мережевих протоколів UDP, TCP, HTTP.

3.4 Методи обробки аудіоданих.

3.5 Середовища розробки програмного забезпечення для ESP32.

3.6 Застосування систем дистанційного передавання аудіо.

4 Вимоги до виконання КБКР

4.1 Провести аналіз проблематики реалізації систем дистанційного передавання аудіо;

4.2 Дослідити архітектуру обміну даними між клієнтським застосунком і мікроконтролером ESP32 через протоколи UDP, TCP і HTTP;

4.3 Розробити апаратну частину системи на базі ESP32 для запису, відтворення та передачі аудіо;

4.4 Реалізувати функціональність керування аудіофайлами (відтворення, видалення, оновлення списку) на мікроконтролері та потокової передачі аудіо;

5 Етапи КБКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

Таблиця А.1 — Етапи КБКР

№ з/п	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Аналіз задачі	21.03.25	30.03.25	Огляд джерел, висновки із взаємодії викладачів та студентів
2	Аналітичний огляд стану сучасних технологій дистанційного передавання аудіо	31.03.25	10.04.25	Розділ 1
3	Проєктування апаратно-програмного комплексу	11.04.25	21.04.25	Розділ 2
4	Конструкторська розробка кіберфізичного пристрою	22.04.25	30.04.25	Розділ 3
5	Оцінка результатів та перспективи розвитку	31.04.25	02.05.25	Розділ 4
6	Оформлення пояснювальної записки та презентації	03.05.24	11.05.24	ПЗ, графічний матеріал, презентація

6 Матеріали, що подаються до захисту КБКР

До захисту подаються: пояснювальна записка КБКР, графічні і ілюстративні матеріали, лістинг програмного забезпечення, протокол попереднього захисту

КБКР на кафедрі, відгук наукового керівника, відгук рецензента, анотації до КБКР українською та іноземною мовами.

7 Порядок контролю виконання та захисту КБКР

Виконання етапів графічної та розрахункової документації КБКР контролюється науковим керівником згідно зі встановленими термінами. Захист КБКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання КБКР

8.1 При оформлюванні БКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 123 — «Комп’ютерна інженерія»;

— документи на які посилаються у вище вказаних.

8.2 Порядок виконання КБКР викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21.

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Апаратно-програмний комплекс для дистанційного передавання аудіо. Частина 1. Кіберфізичний пристрій

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 3 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

- ☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

- ☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Азаров О.Д., зав. каф. ОТ _____ _____
(прізвище, ініціали, посада) (підпис)

Крупельницький Л.В., доц. каф ОТ _____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ д.т.н., проф. каф. ОТ Мартинюк Т. Б.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Бондарчук О. Б.
(підпис) (прізвище, ініціали)

ДОДАТОК В

Графічна частина

АПАРАТНО-ПРОГРАМНИЙ КОМПЛЕКС ДЛЯ ДИСТАНЦІЙНОГО ПЕРЕДАВАННЯ АУДІО. ЧАСТИНА 1. КІБЕРФІЗИЧНИЙ ПРИСТРІЙ

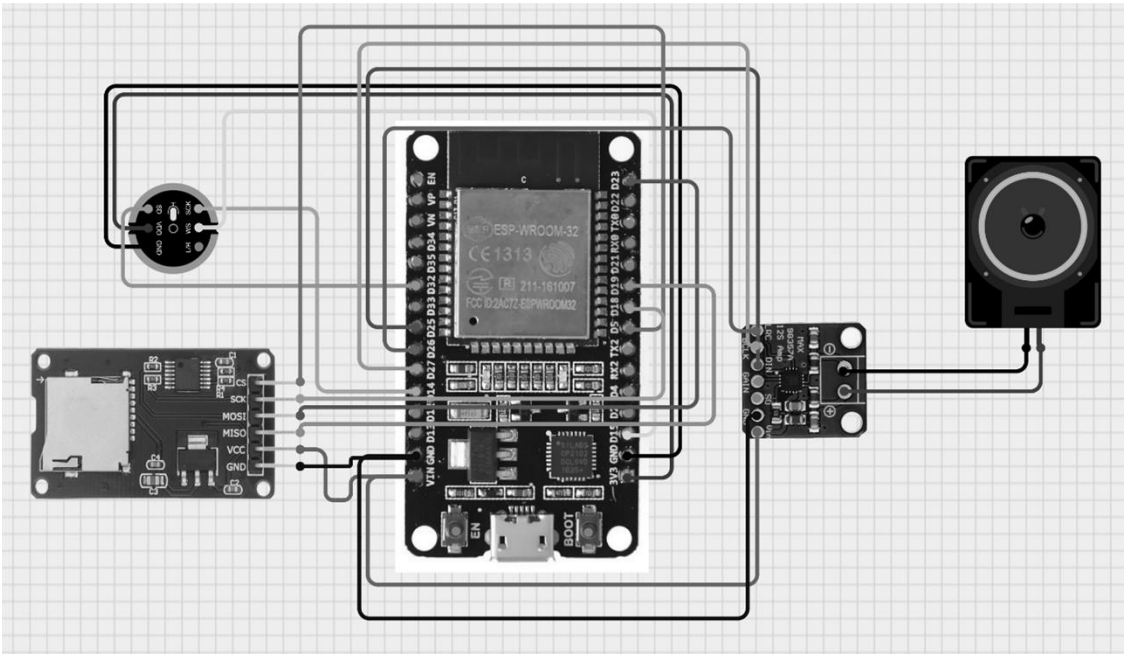

Рисунок В.1 — Принципова схема підключення

ДОДАТОК Г

Ілюстративна частина

АПАРАТНО-ПРОГРАМНИЙ КОМПЛЕКС ДЛЯ ДИСТАНЦІЙНОГО ПЕРЕДАВАННЯ АУДІО. ЧАСТИНА 1. КІБЕРФІЗИЧНИЙ ПРИСТРІЙ

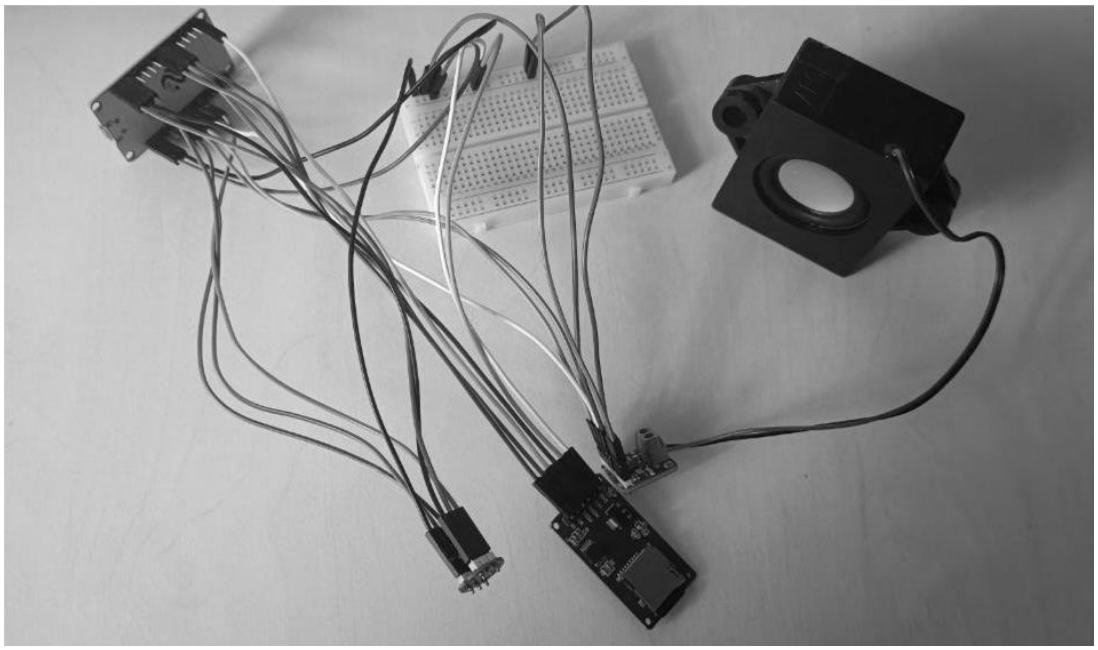

Рисунок Г.1 — Макет реалізованого пристрою

ДОДАТОК Д

Лістинг програмного забезпечення

```
#include <WiFi.h>
#include <WebServer.h>
#include <ESPmDNS.h>
#include <SdFat.h>
#include <SPI.h>
#include <WiFiUdp.h>
#include <AudioTools.h>
#define DEVICE_NAME "ESP32 Audio Device"
#define FIRMWARE_VERSION "1.0.0"
const char* WIFI_SSID = "ESP32_AudioDevice";
const char* WIFI_PASSWORD = "12345678";
#define HTTP_PORT 80
#define UDP_AUDIO_PORT 5555
#define UDP_OUT_AUDIO_PORT 5556
#define UDP_BROADCAST_PORT 12345
#define TCP_FILE_PORT 8080
#define SAMPLE_RATE 8000
#define CHANNELS 1
#define BITS_PER_SAMPLE_IN 32
#define BITS_PER_SAMPLE_OUT 16
#define I2S_BCLK_PIN 14
#define I2S_WS_PIN 15
#define I2S_DATA_IN_PIN 32
#define I2S_DOUT_PIN 25
#define I2S_BCLK_OUT_PIN 27
#define I2S_LRC_OUT_PIN 26
#define SD_CS_PIN 5
#define SD_MOSI_PIN 23
#define SD_MISO_PIN 19
#define SD_SCK_PIN 18
WebServer server(HTTP_PORT);
WiFiUDP udpServer;
WiFiUDP udpAudioReceiver;
WiFiServer tcpFileServer(TCP_FILE_PORT);
WiFiClient tcpClient;
SdFat SD;
SPIClass sdSPI(HSPI);
I2SStream i2s_out;
I2SStream i2s_in;
WAVDecoder wavDecoder;
EncodedAudioStream wavStream(&i2s_out, &wavDecoder);
StreamCopy copier;
FsFile audioFile;
RingBuffer<uint8_t> audioBuffer(8192);
String currentAudioFile = "";
bool isAudioStreaming = false;
bool isAudioListening = false;
bool isPlaying = false;
```

```
bool isRecording = false;
String pendingFilename = "";
bool hasSDCard = false;
uint64_t sdCardCapacity = 0;
uint64_t sdCardFreeSpace = 0;
IPAddress lastClientIP;
uint16_t lastClientPort = 0;
int currentChannels = CHANNELS;
char fileList[128] = "[]";
void setupWiFi();
void setupAudio();
void setupSD();
void setupWebServer();
void setupUDPAudio();
void setupTCPFileServer();
void streamAudioToUDP();
void receiveAudioFromUDP();
void playSDCardAudio();
void recordToSDCard();
void enableAudioOutput(bool enable);
void enableAudioInput(bool enable);
void handleUDPBroadcast();
void handleTCPFileTransfer();
void updateFileList();
void stopPlayback();
void stopRecording();
void writeWAVHeader(FsFile &file, int sampleRate, int channels, int bitsPerSample);
void setup() {
  Serial.begin(115200);
  delay(1000);
  Serial.println("\n\nESP32 Audio Device Starting...");
  setupSD();
  delay(200);
  setupAudio();
  delay(200);
  setupWiFi();
  setupWebServer();
  setupUDPAudio();
  setupTCPFileServer();
  Serial.println("System initialization complete");}
void setupWiFi() {
  WiFi.mode(WIFI_AP);
  WiFi.softAP(WIFI_SSID, WIFI_PASSWORD);
  if (!MDNS.begin("esp32audio")) {
    Serial.println("Error setting up MDNS responder!");  }
  Serial.print("AP IP address: ");
  Serial.println(WiFi.softAPIP());
  udpServer.begin(UDP_BROADCAST_PORT);}
void setupAudio() {
  Serial.println("Initializing Audio...");
  auto config_out = i2s_out.defaultConfig(TX_MODE);
  config_out.port_no = I2S_NUM_0;
```

```
  config_out.pin_bck = I2S_BCLK_OUT_PIN;
  config_out.pin_ws = I2S_LRC_OUT_PIN;
  config_out.pin_data = I2S_DOUT_PIN;
  config_out.sample_rate = SAMPLE_RATE;
  config_out.bits_per_sample = BITS_PER_SAMPLE_IN;
  config_out.channels = CHANNELS;
  config_out.buffer_count = 8;
  config_out.buffer_size = 64;
  if (!i2s_out.begin(config_out)) {
    Serial.println("Failed to initialize I2S output");
  } else {
    Serial.println("I2S output initialized"); }
  auto config_in = i2s_in.defaultConfig(RX_MODE);
  config_in.port_no = I2S_NUM_1;
  config_in.pin_bck = I2S_BCLK_PIN;
  config_in.pin_ws = I2S_WS_PIN;
  config_in.pin_data = I2S_DATA_IN_PIN;
  config_in.sample_rate = SAMPLE_RATE;
  config_in.bits_per_sample = BITS_PER_SAMPLE_OUT;
  config_in.channels = CHANNELS;
  config_in.buffer_count = 8;
  config_in.buffer_size = 64;
  if (!i2s_in.begin(config_in)) {
    Serial.println("Failed to initialize I2S input");
  } else {
    Serial.println("I2S input initialized");  }}
void setupSD() {
  sdSPI.begin(SD_SCK_PIN, SD_MISO_PIN, SD_MOSI_PIN, SD_CS_PIN);
  if (!SD.begin(SdSpiConfig(SD_CS_PIN, DEDICATED_SPI, SD_SCK_MHZ(16), &sdSPI))){
    Serial.println("SD Card Mount Failed");
    hasSDCard = false;
    return;}
  Serial.println("SD Card mounted successfully");
  hasSDCard = true;
  sdCardCapacity = SD.card()->sectorCount() * 512ULL;
  sdCardFreeSpace = SD.freeClusterCount() * SD.sectorsPerCluster() * 512ULL;
  if (!SD.exists("/AudioFiles")) {
    SD.mkdir("/AudioFiles");
    Serial.println("/AudioFiles directory created"); }
  updateFileList();}
void setupUDPAudio() {
  udpServer.begin(UDP_AUDIO_PORT);
  udpAudioReceiver.begin(UDP_OUT_AUDIO_PORT);
  Serial.print("UDP audio receiver started on port: ");
  Serial.println(UDP_OUT_AUDIO_PORT);}
void setupTCPFileServer() {
  tcpFileServer.begin();
  Serial.println("TCP file server started");}
void setupWebServer() {
  server.on("/api/status", HTTP_GET, []() {
    server.send(200, "text/plain", "ok");  });
  server.on("/api/config", HTTP_GET, []() {
```

```
  String configResponse = "{\"DeviceName\":\"" + String(DEVICE_NAME) +
                          "\",\"FirmwareVersion\":\"" + String(FIRMWARE_VERSION) +
                          "\",\"HasSDCard\":" + String(hasSDCard ? "true" : "false") +
                          ",\"SDCardCapacity\":" + String(sdCardCapacity) +
                          ",\"SDCardFreeSpace\":" + String(sdCardFreeSpace) + "}";
  server.send(200, "application/json", configResponse);  });
server.on("/api/audio/streaming/startsending", HTTP_GET, []() {
  if (!isAudioStreaming) {
    stopPlayback();
    stopRecording();
    if (isAudioListening) {
      isAudioListening = false;
      enableAudioOutput(false);
      delay(100);   }
    isAudioStreaming = true;
    lastClientIP = server.client().remoteIP();
    lastClientPort = 5555;
    enableAudioInput(true);
    Serial.print("Streaming to IP: ");
    Serial.println(lastClientIP);
    server.send(200, "application/json", "{\"status\":\"streaming_started\"}");
  } else {
    server.send(200, "application/json", "{\"status\":\"already_streaming\"}"); }
});
server.on("/api/audio/streaming/stopsending", HTTP_GET, []() {
  if (isAudioStreaming) {
    isAudioStreaming = false;
    enableAudioInput(false);
    Serial.println("Streaming stopped");
    server.send(200, "application/json", "{\"status\":\"streaming_stopped\"}");
  } else {
    server.send(200, "application/json", "{\"status\":\"not_streaming\"}");}
});
server.on("/api/audio/streaming/startlistening", HTTP_GET, []() {
  if (!isAudioListening) {
    if (isAudioStreaming) {
      isAudioStreaming = false;
      enableAudioInput(false);
      delay(100);  }
    stopRecording();
    stopPlayback();
    isAudioListening = true;
    lastClientIP = server.client().remoteIP();
    lastClientPort = 5556;
    enableAudioOutput(true);
    Serial.print("Listening to IP: ");
    Serial.println(lastClientIP);
    server.send(200, "application/json", "{\"status\":\"listening_started\"}");
  } else {
    server.send(200, "application/json", "{\"status\":\"already_listening\"}"); }  });
server.on("/api/audio/streaming/stoplistening", HTTP_GET, []() {
  if (isAudioListening) {
```

```
    isAudioListening = false;
    enableAudioOutput(false);
    server.send(200, "application/json", "{\"status\":\"listening_stopped\"}");
  } else {
    server.send(200, "application/json", "{\"status\":\"not_listening\"}"); }  });
server.on("/api/audio/files/list", HTTP_GET, []() {
  if (!hasSDCard) {
    server.send(500, "application/json", "{\"error\":\"no_sd_card\"}");
    return; }
  Serial.print("Returning cached file list: ");
  Serial.println(fileList);
  server.send(200, "application/json", fileList); });
server.on("/api/audio/files/play", HTTP_GET, []() {
  if (!server.hasArg("name")) {
    server.send(400, "application/json", "{\"error\":\"name_missing\"}");
    return;  }
  String filename = "/AudioFiles/" + server.arg("name");
  if (!SD.exists(filename)) {
    server.send(404, "application/json", "{\"error\":\"file_not_found\"}");
    return; }
  if (isAudioStreaming) {
    isAudioStreaming = false;
    enableAudioInput(false);
    delay(100);}
  if (isAudioListening) {
    isAudioListening = false;
    enableAudioOutput(false);
    delay(100); }
  stopRecording();
  stopPlayback();
  audioFile = SD.open(filename, O_RDONLY);
  if (!audioFile) {
    server.send(500, "application/json", "{\"error\":\"failed_to_open_file\"}");
    return; }
  uint8_t header[44];
  audioFile.read(header, 44);
  int sampleRate = *(int*)&header[24];
  short channels = *(short*)&header[22];
  short bitsPerSample = *(short*)&header[34];
  audioFile.seek(0);
  if (channels != currentChannels || sampleRate != SAMPLE_RATE) {
    i2s_out.end();
    auto config_out = i2s_out.defaultConfig(TX_MODE);
    config_out.port_no = I2S_NUM_0;
    config_out.sample_rate = SAMPLE_RATE;
    config_out.channels = channels;
    config_out.bits_per_sample = BITS_PER_SAMPLE_OUT;
    config_out.pin_bck = I2S_BCLK_OUT_PIN;
    config_out.pin_ws = I2S_LRC_OUT_PIN;
    config_out.pin_data = I2S_DOUT_PIN;
    config_out.buffer_count = 8;
    config_out.buffer_size = 64;
```

```
        if (!i2s_out.begin(config_out)) {
          Serial.println("Failed to reinitialize I2S output");
          server.send(500, "application/json", "{\"error\":\"i2s_reinit_failed\"}");
          return;  }
        currentChannels = channels; }
      wavStream.begin();
      copier.begin(wavStream, audioFile);
      isPlaying = true;
      currentAudioFile = filename;
      enableAudioOutput(true);
      server.send(200, "application/json", "{\"status\":\"playing\"}") });
    server.on("/api/audio/files/delete", HTTP_GET, []() {
      if (!server.hasArg("name")) {
        server.send(400, "application/json", "{\"error\":\"name_missing\"}");
        return;  }
      String filename = "/AudioFiles/" + server.arg("name");
      if (!SD.exists(filename)) {
        server.send(404, "application/json", "{\"error\":\"file_not_found\"}");
        return; }
      if (isPlaying && currentAudioFile == filename) {
        stopPlayback(); }
      if (SD.remove(filename)) {
        updateFileList();
        server.send(200, "application/json", "{\"status\":\"deleted\"}");
      } else {
        server.send(500, "application/json", "{\"error\":\"failed_to_delete\"}"); } });
    server.on("/api/audio/files/upload", HTTP_GET, []() {
      if (!server.hasArg("filename")) {
        server.send(400, "application/json", "{\"error\":\"filename_missing\"}");
        return;}
      pendingFilename = server.arg("filename");
      if (tcpClient && tcpClient.connected()) {
        tcpClient.stop();}
      server.send(200, "application/json", "{\"status\":\"listening\",\"port\":" +
String(TCP_FILE_PORT) + "}"); });
    server.on("/api/audio/record/start", HTTP_GET, []() {
      if (!server.hasArg("filename")) {
        server.send(400, "application/json", "{\"error\":\"filename_missing\"}");
        return; }
      if (isRecording) {
        server.send(400, "application/json", "{\"status\":\"already_recording\"}");
        return; }
      if (isAudioStreaming) {
        isAudioStreaming = false;
        enableAudioInput(false);
        delay(100);    }
      if (isAudioListening) {
        isAudioListening = false;
        enableAudioOutput(false);
        delay(100);    }
      stopPlayback();
      String filename = "/AudioFiles/" + server.arg("filename");
```

```
    audioFile = SD.open(filename, O_WRONLY | O_CREAT | O_TRUNC);
    if (!audioFile) {
      server.send(500, "application/json", "{\"error\":\"failed_to_open_file\"}");
      return;}
    writeWAVHeader(audioFile, SAMPLE_RATE, CHANNELS, BITS_PER_SAMPLE_OUT);
    isRecording = true;
    currentAudioFile = filename;
    enableAudioInput(true);
    server.send(200, "application/json", "{\"status\":\"recording_started\"}");});
  server.on("/api/audio/record/stop", HTTP_GET, []() {
    if (!isRecording) {
      server.send(400, "application/json", "{\"status\":\"not_recording\"}");
      return;}
    stopRecording();
    updateFileList();
    server.send(200, "application/json", "{\"status\":\"recording_stopped\"}");  });
  server.begin();
  Serial.println("HTTP server started");}
void updateFileList() {
  FsFile dir = SD.open("/AudioFiles");
  if (!dir || !dir.isDirectory()) {
    strcpy(fileList, "[]");
    if (dir) dir.close();
    return;  }
  char tempList[128] = "[";
  size_t len = 1;
  bool first = true;
  int fileCount = 0;
  const int maxFiles = 5;
  while (FsFile entry = dir.openNextFile()) {
    char name[64];
    entry.getName(name, sizeof(name));
    if (!entry.isDirectory()) {
      size_t nameLen = strlen(name);
      if (fileCount >= maxFiles) break;
      if (!first) {
        if (len + 1 < sizeof(tempList) - 1) {
          strcat(tempList, ",");
          len++;
        } else break;  }
      if (len + nameLen + 3 < sizeof(tempList) - 1) {
        strcat(tempList, "\"");
        strcat(tempList, name);
        strcat(tempList, "\"");
        len += nameLen + 2;
        first = false;
        fileCount++;
      } else break;   }
    entry.close();
    yield();}
  dir.close();
  if (len + 2 < sizeof(tempList)) {
```

```
    strcat(tempList, "]");
    strcpy(fileList, tempList);
  } else {
    strcpy(fileList, "[]");}
  Serial.print("Updated file list: ");
  Serial.println(fileList);}
void enableAudioOutput(bool enable) {
  static bool outputActive = false;
  if (enable && !outputActive) {
    i2s_out.end();
    auto config_out = i2s_out.defaultConfig(TX_MODE);
    config_out.port_no = I2S_NUM_0;
    config_out.pin_bck = I2S_BCLK_OUT_PIN;
    config_out.pin_ws = I2S_LRC_OUT_PIN;
    config_out.pin_data = I2S_DOUT_PIN;
    config_out.sample_rate = SAMPLE_RATE;
    config_out.bits_per_sample = BITS_PER_SAMPLE_OUT;
    config_out.channels = CHANNELS;
    config_out.buffer_count = 8;
    config_out.buffer_size = 64;
    if (!i2s_out.begin(config_out)) {
      Serial.println("Failed to enable I2S output");
    } else {
      outputActive = true;
      Serial.println("I2S output enabled"); }
    delay(100);
  } else if (!enable && outputActive && !isPlaying && !isAudioListening) {
    i2s_out.end();
    outputActive = false;
    Serial.println("I2S output disabled");
    delay(100);  }}
void enableAudioInput(bool enable) {
  static bool inputActive = false;
  if (enable && !inputActive) {
    i2s_in.end();
    auto config_in = i2s_in.defaultConfig(RX_MODE);
    config_in.port_no = I2S_NUM_1;
    config_in.pin_bck = I2S_BCLK_PIN;
    config_in.pin_ws = I2S_WS_PIN;
    config_in.pin_data = I2S_DATA_IN_PIN;
    config_in.sample_rate = SAMPLE_RATE;
    config_in.bits_per_sample = BITS_PER_SAMPLE_IN;
    config_in.channels = CHANNELS;
    config_in.buffer_count = 8;
    config_in.buffer_size = 64;
    if (!i2s_in.begin(config_in)) {
      Serial.println("Failed to enable I2S input");
    } else {
      inputActive = true;
      Serial.println("I2S input enabled"); }
    delay(100);
  } else if (!enable && inputActive && !isAudioStreaming && !isRecording) {
```

```
    i2s_in.end();
    inputActive = false;
    Serial.println("I2S input disabled");
    delay(100);  }}
void streamAudioToUDP() {
  if (lastClientIP == IPAddress(0, 0, 0, 0) || lastClientPort == 0) {
    Serial.println("No valid client IP or port for streaming");
    return;  }
  enableAudioInput(true);
  uint8_t audioBuffer[512];
  size_t bytesRead = i2s_in.readBytes(audioBuffer, sizeof(audioBuffer));
  if (bytesRead > 0) {
    udpServer.beginPacket(lastClientIP, lastClientPort);
    udpServer.write(audioBuffer, bytesRead);
    udpServer.endPacket();
    Serial.printf("Streamed %d bytes to UDP\n", bytesRead);
  } else {
    Serial.println("No data read from I2S input");}
  yield();}
void receiveAudioFromUDP() {
  static unsigned long lastPacketTime = 0;
  static const unsigned long TIMEOUT = 1000;
  int packetSize = udpAudioReceiver.parsePacket();
  if (packetSize > 0) {
    lastPacketTime = millis();
    uint8_t* buffer = (uint8_t*)malloc(packetSize);
    if (buffer == NULL) {
      Serial.println("Failed to allocate memory for UDP packet");
      return;}
   int len = udpAudioReceiver.read(buffer, packetSize);
    if (len > 0) {
      for (int i = 0; i < len; i++) {
        audioBuffer.write(buffer[i]);}}
    free(buffer);}
  while (audioBuffer.available() > 0 && isAudioListening) {
    uint8_t sample;
    if (audioBuffer.read(sample)) {
      static uint8_t audioReadBuffer[512];
      static int audioReadPos = 0;
      audioReadBuffer[audioReadPos++] = sample;
      if (audioReadPos >= 512) {
        i2s_out.write(audioReadBuffer, audioReadPos);
        audioReadPos = 0;}}}
  if (isAudioListening && (millis() - lastPacketTime > TIMEOUT)) {
    audioBuffer.reset();}}
void playSDCardAudio() {
  if (isPlaying) {
    size_t bytesCopied = copier.copy();
    if (!bytesCopied && !audioFile.available()) {
      stopPlayback();
      Serial.println("Playback finished"); }}}
void recordToSDCard() {
```

```
  if (isRecording) {
    uint8_t buffer[512];
    size_t bytesRead = i2s_in.readBytes(buffer, sizeof(buffer));
    if (bytesRead > 0) {
      audioFile.write(buffer, bytesRead);
      Serial.printf("Recorded %d bytes to SD\n", bytesRead);
    } else {
      Serial.println("No data read for recording");}}}
void stopPlayback() {
  if (isPlaying) {
    copier.end();
    wavStream.end();
    i2s_out.flush();
    audioFile.close();
    isPlaying = false;
    currentAudioFile = "";
    enableAudioOutput(false);}}
void stopRecording() {
  if (isRecording) {
    audioFile.close();
    isRecording = false;
    currentAudioFile = "";
    enableAudioInput(false);}}
void writeWAVHeader(FsFile &file, int sampleRate, int channels, int bitsPerSample) {
  uint8_t header[44];
  memset(header, 0, 44);
  memcpy(header, "RIFF", 4);
  *(uint32_t*)&header[4] = 36;
  memcpy(header + 8, "WAVE", 4);
  memcpy(header + 12, "fmt ", 4);
  *(uint32_t*)&header[16] = 16;
  *(uint16_t*)&header[20] = 1;
  *(uint16_t*)&header[22] = channels;
  *(uint32_t*)&header[24] = sampleRate;
  *(uint32_t*)&header[28] = sampleRate * channels * bitsPerSample / 8;
  *(uint16_t*)&header[32] = channels * bitsPerSample / 8;
  *(uint16_t*)&header[34] = bitsPerSample;
  memcpy(header + 36, "data", 4);
  *(uint32_t*)&header[40] = 0;
  file.write(header, 44);}
void handleTCPFileTransfer() {
  if (!tcpClient) {
    tcpClient = tcpFileServer.available();
    if (tcpClient) Serial.println("New TCP client connected");}
  if (tcpClient && tcpClient.connected() && pendingFilename != "") {
    String filename = "/AudioFiles/" + pendingFilename;
    FsFile file = SD.open(filename, O_WRONLY | O_CREAT | O_TRUNC);
    if (!file) {
      Serial.println("Failed to open file for writing");
      tcpClient.stop();
      pendingFilename = "";
      return;}
```

```
            unsigned long startTime = millis();
            size_t totalBytes = 0;
            while (tcpClient.connected()) {
              if (tcpClient.available()) {
                uint8_t buffer[1024];
                int bytesRead = tcpClient.read(buffer, sizeof(buffer));
                if (bytesRead > 0) {
                  file.write(buffer, bytesRead);
                  totalBytes += bytesRead;}
                yield();
                delay(1);
              } else {
                delay(10);
                if (millis() - startTime > 5000) break;}}
            file.close();
            tcpClient.stop();
            updateFileList();
            pendingFilename = "";
            Serial.printf("File received: %u bytes\n", totalBytes);}}
          void handleUDPBroadcast() {
            int packetSize = udpServer.parsePacket();
            if (packetSize) {
              char packetBuffer[255];
              int len = udpServer.read(packetBuffer, 255);
              if (len > 0) {
                packetBuffer[len] = 0;
                if (strcmp(packetBuffer, "ESP32 scanning") == 0) {
                  String response = "ESP32 device " + WiFi.softAPIP().toString() + ":" +
String(HTTP_PORT);
                  udpServer.beginPacket(udpServer.remoteIP(), udpServer.remotePort());
                  udpServer.write((uint8_t*)response.c_str(), response.length());
                  udpServer.endPacket();
                  Serial.println("Sent discovery response");}}}}
          void loop() {
            server.handleClient();
            handleUDPBroadcast();
            handleTCPFileTransfer();
            if (isAudioStreaming) {
              streamAudioToUDP();}
            if (isAudioListening) {
              receiveAudioFromUDP();}
            if (isPlaying) {
              playSDCardAudio();}
            if (isRecording) {
              recordToSDCard();}
            static unsigned long lastStatus = 0;
            if (millis() - lastStatus > 5000) {
              lastStatus = millis();
              Serial.printf("Status: Streaming=%d, Listening=%d, Playing=%d, Recording=%d\n",
                            isAudioStreaming, isAudioListening, isPlaying, isRecording);}
            yield();}
```