Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: *Комп'ютеризована система USB аудіоплеєра з підтримкою веб-еквалайзера.*

Виконав: студент 4 курсу, групи СІ-42
спеціальності 123 «Комп'ютерна інженерія»
(шифр і назва спеціальності)

	(підпис)	*Пуляк М. Я.* (прізвище та ініціали)
Керівник	(підпис)	*Варавін А. В.* (прізвище та ініціали)
Нормоконтроль	(підпис)	*Тиш Є. В.* (прізвище та ініціали)
Завідувач кафедри	(підпис)	*Осухівська Г.М.* (прізвище та ініціали)
Рецензент	(підпис)	*Бойко І.В.* (прізвище та ініціали)

Тернопіль
2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет *комп'ютерно-інформаційних систем і програмної інженерії*
(повна назва факультету)

Кафедра *комп'ютерних систем та мереж*
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
Осухівська Г.М.
(підпис) (прізвище та ініціали)
«27» січня 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня *бакалавр*
(назва освітнього ступеня)

за спеціальністю *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

студенту *Пуляк Максим Ярославович*
(прізвище, ім'я, по батькові)

1. Тема роботи *Комп'ютеризована система USB аудіоплеєра з підтримкою веб-еквалайзера.*

Керівник роботи *Варавін Антон Валерійович, канд. фіз-мат. наук*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від *« 27 » січня 2025 року № 4/7-53*
2. Термін подання студентом завершеної роботи *19.06.2025р.*
3. Вихідні дані до роботи *Технічне завдання*

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз технічного завдання usb аудіоплеєра
2. Проєктування системи usb аудіоплеєра
3. Програмна реалізація та тестування комп'ютеризованої системи
4. Безпека життєдіяльності, основи охорони праці.
Висновки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Структурна схема системи
2. Модель тестового макету
3. Схема електричних з'єднань
4. Блок-схема алгоритму роботи програмного забезпечення

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	*д.т.н., професор Пилипець М. І.*		

7. Дата видачі завдання *27.01.2025 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	*Розробка технічного завдання*	*27.01 – 02.02*	*Викон.*
2.	*Аналіз технічного завдання usb аудіоплеєра*	*12.02 – 25.03*	*Викон.*
3.	*Проєктування системи usb аудіоплеєра*	*06.04 – 03.05*	*Викон.*
4.	*Програмна реалізація та тестування комп'ютеризованої системи*	*08.05 – 26.05*	*Викон.*
5.	*Безпека життєдіяльності, основи охорони праці*	*27.05 – 02.06*	*Викон.*
6.	*Оформлення пояснювальної записки і графічного матеріалу*	*03.06 – 08.06*	*Викон.*
7.	*Перевірка на академічний плагіат, перевірка керівником та консультантами*	*9.06 – 15.06*	*Викон.*
8.	*Попередній захист кваліфікаційної роботи бакалавра*	*16.06 – 22.06*	*Викон.*
9.	*Захист кваліфікаційної роботи бакалавра*	*26.06*	*Викон.*

Студент _____ *Пуляк Максим Ярославович*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Варавін Антон Валерійович*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Пуляк М.Я. Комп’ютеризована система USB аудіоплеєра з підтримкою веб-еквалайзера: робота на здобуття кваліфікаційного ступеня бакалавра: спец. 123 — комп’ютерна інженерія. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2025.

Ключові слова: комп’ютеризована система, USB аудіоплеєр, веб-еквалайзер, STM32F407VGT6, ESP32-Wrover-Dev, CS43L22, цифрова обробка сигналів, біквадратичні фільтри, CMSIS-DSP, I2S, USART, Wi-Fi, веб-сервер, MP3.

Кваліфікаційна робота присвячена розробці комп’ютеризованої системи USB аудіоплеєра з веб-керованим еквалайзером на базі мікроконтролера STM32F407VGT6 для аудіообробки та модуля ESP32-Wrover-Dev для веб-інтерфейсу.

У першому розділі проведено аналіз технічного завдання: визначено функціональне призначення та основні задачі системи, розглянуто альтернативні апаратні платформи та компоненти, обґрунтовано вибір ключових елементів та програмних засобів розробки.

У другому розділі описано проектну частину: розроблено структурну схему комп’ютеризованої системи, наведено схеми електричних з'єднань та принципові рішення для ключових вузлів, а також деталізовано використані шини та протоколи передачі даних.

У третьому розділі деталізовано програмну реалізацію для STM32 та ESP32, а також наведено результати тестування.

У четвертому розділі висвітлено аспекти безпеки життєдіяльності та охорони праці: розглянуто потенційні небезпеки при роботі з електронним обладнанням та заходи щодо їх мінімізації.

ANNOTATION

Puliak M.Y. Computerized USB audio player system with web equalizer support: Bachelor's Graduation Thesis: Spec. 123 — Computer Engineering. Ternopil: Ternopil Ivan Puluj National Technical University, 2025.

Keywords: computerized system, USB audio player, web equalizer, STM32F407VGT6, ESP32-Wrover-Dev, CS43L22, digital signal processing, biquad filters, CMSIS-DSP, I2S, USART, Wi-Fi, web server, MP3.

The qualification thesis is dedicated to the development of a computerized USB audio player system with a web-controlled equalizer based on the STM32F407VGT6 microcontroller for audio processing and the ESP32-Wrover-Dev module for the web interface.

The first section analyzes the technical task: the functional purpose and main tasks of the system are defined, alternative hardware platforms and components are considered, and the choice of key elements and software development tools is substantiated.

The second section describes the design part: the structural diagram of the computerized system is developed, diagrams of electrical connections and principal solutions for key nodes are presented, and the used buses and data transfer protocols are detailed.

The third section details the software implementation for STM32 and ESP32, and also presents the testing results.

The fourth section highlights aspects of life safety and labor protection: potential hazards when working with electronic equipment and measures to minimize them are considered.

ЗМІСТ

СПИСОК СКОРОЧЕНЬ

ARM – Advanced RISC Machine

DMA – Direct Memory Access

DSP – Digital Signal Processing

FAT32 – File Allocation Table 32

FPU – Floating-Point Unit

GPIO – General-Purpose Input/Output

HTTP – Hypertext Transfer Protocol

I2C – Inter-Integrated Circuit

I2S – Inter-IC Sound

MCU – Microcontroller Unit

PCM – Pulse-Code Modulation

RAM – Random Access Memory

USART - Universal Synchronous/Asynchronous Receiver/Transmitter

USB – Universal Serial Bus

USB OTG – USB On-The-Go

ВСТУП

Від портативних персональних пристроїв до складних професійних установок, вимоги до якості відтворення звуку, гнучкості керування та зручності користувацького інтерфейсу невпинно зростають. Сучасні користувачі прагнуть не лише пасивного споживання аудіоконтенту, але й активної взаємодії з ним, що включає можливість тонкого налаштування параметрів звучання відповідно до індивідуальних вподобань, акустичних особливостей середовища або специфіки відтворюваного матеріалу.

Одним із ключових елементів сучасних аудіосистем є еквалайзер, що дозволяє коригувати амплітудно-частотну характеристику (АЧХ) сигналу. Традиційні апаратні еквалайзери часто мають обмежену кількість смуг або незручні органи керування. Водночас, програмні рішення, інтегровані в операційні системи комп'ютерів чи мобільних пристроїв, не завжди доступні для спеціалізованих вбудованих систем. Тому створення автономних аудіопристроїв з розширеними можливостями еквалізації, керованими через інтуїтивно зрозумілі та доступні інтерфейси, становить значний інтерес. Використання веб-технологій для реалізації користувацького інтерфейсу дозволяє керувати пристроєм з будь-якого сучасного гаджета, оснащеного веб-браузером, без необхідності встановлення спеціалізованого програмного забезпечення.

Комп’ютеризована система USB аудіоплеєра, призначенна не лише для забезпечення високоякісного відтворення аудіофайлів, але й надає користувачеві потужний інструмент для налаштування звуку – багатосмуговий графічний еквалайзер, керований через веб-інтерфейс. Такий підхід поєднує переваги вбудованих систем, таких як компактність та енергоефективність, з гнучкістю та універсальністю веб-технологій. Розробка даної системи також має значну освітню цінність, оскільки охоплює широке коло питань, пов'язаних з мікроконтролерною технікою, цифровою обробкою сигналів, програмуванням вбудованих систем та протоколами передачі даних.

РОЗДІЛ 1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ USB АУДІОПЛЕЄРА

1.1 Визначення задач та призначення пристрою

У рамках даної кваліфікаційної роботи було поставлено завдання розробки комп'ютеризованої системи USB аудіоплеєра. Призначенням даної системи є забезпечення високоякісного відтворення аудіосигналів, отриманих через USB-інтерфейс, з наданням користувачеві можливості гнучкого налаштування частотних характеристик звуку в реальному часі.

Ключовими задачами, що вирішувалися при розробці, стали: прийом та декодування цифрового аудіопотоку з USB-джерела; цифрова обробка сигналу, що включає застосування біквадратичних фільтрів для реалізації функціоналу багатосмугового еквалайзера; подальше цифро-аналогове перетворення обробленого сигналу для його виведення на зовнішні аудіопристрої. Важливою задачею також було забезпечення інтуїтивного інтерфейсу для керування параметрами еквалайзера та загальним рівнем гучності хї [1].

Для реалізації зазначених задач було спроектовано двокомпонентну архітектуру. Функції безпосереднього відтворення аудіо, включаючи застосування алгоритмів фільтрації, покладено на мікроконтролер STM32F407VGT6. У свою чергу, мікроконтролер ESP32-Wrover-Dev використано для організації точки доступу Wi-Fi та хостингу веб-сервера. Через веб-інтерфейс, доступний з будь-якого пристрою з веб-браузером, користувач отримує змогу динамічно змінювати налаштування еквалайзера. Дані про змінені параметри передаються від модуля ESP32 до модуля STM32 у вигляді пакетів даних через послідовний інтерфейс USART, що призводить

до негайної модифікації коефіцієнтів біквадратичних фільтрів на STM32.

Таким чином, розроблюваний пристрій створює повноцінну аудіосистему з розширеними можливостями користувацького налаштування звуку. Це дозволяє не тільки відтворювати аудіоконтент, але й адаптувати його звучання під індивідуальні вподобання слухача або акустичні умови приміщення. Подібні системи можуть знайти застосування при створенні спеціалізованих аудіопрогравачів, в інтерактивних мультимедійних установках, або як модулі для систем, що вимагають керованої обробки аудіосигналу [2].

1.2 Аналіз альтернативних апаратних платформ та компонентів для реалізації системи

В ході проектування комп’ютеризованої системи USB аудіоплеєра з веб-еквалайзером було здійснено аналіз альтернативних апаратних платформ, здатних реалізувати поставлені завдання. Розгляд аналогів проводився для ключових функціональних вузлів системи: мікроконтролера для обробки аудіосигналу та мікроконтролера або модуля, відповідального за мережеву взаємодію та користувацький веб-інтерфейс, а також аудіокодеків.

Для виконання завдань, пов'язаних з обробкою аудіосигналу, таких як прийом цифрового потоку, його декодування, застосування алгоритмів цифрової фільтрації (зокрема, біквадратичних фільтрів для реалізації функціоналу еквалайзера) та взаємодія з аудіокодеком, на ринку існує широкий спектр мікроконтролерів. Мікроконтролери сімейства STM32 від STMicroelectronics, зокрема серії F4, F7 та H7, що базуються на ядрах ARM Cortex-M4/M7, є популярним вибором для аудіозастосувань завдяки високій продуктивності, наявності FPU (блоку обробки чисел з плаваючою комою), спеціалізованих периферійних інтерфейсів (I2S, SAI, DFSDM) та розвиненій екосистемі. Наприклад, мікроконтролери серії STM32H7 пропонують значно

вищу тактову частоту та більший обсяг пам'яті, що дозволило б реалізувати складніші алгоритми обробки або підтримувати аудіо вищої роздільної здатності без суттєвих обмежень. Альтернативою можуть слугувати мікроконтролери від NXP Semiconductors, наприклад, серії LPC (зокрема, LPC55Sxx з DSP-розширеннями ARM Cortex-M33) або i.MX RT, які є кросоверними процесорами, що поєднують продуктивність прикладних процесорів з простотою використання мікроконтролерів. Вони також мають розвинені аудіоінтерфейси та достатню обчислювальну потужність. Компанія Microchip Technology пропонує мікроконтролери серій SAM (на базі ARM Cortex-M) та PIC32 (на базі MIPS). Деякі з них, наприклад, SAM E70/S70/V70/V71, оснащені ядром Cortex-M7, FPU та периферією, придатною для аудіообробки. Для проектів, де критичним є наднизьке енергоспоживання, могли б розглядатися мікроконтролери з оптимізованим енергоспоживанням, проте це часто йде врозріз з вимогами до високої продуктивності для обробки аудіо в реальному часі [4].

Для реалізації функціоналу веб-сервера, точки доступу Wi-Fi та забезпечення користувацького інтерфейсу через веб-еквалайзер, а також для передачі команд управління на основний аудіопроцесор, також існує декілька підходів. Модулі на базі мікроконтролерів ESP32 від Espressif Systems є дуже поширеним рішенням завдяки інтегрованим Wi-Fi та Bluetooth, двоядерній архітектурі та доступній ціні. Існують різні модифікації ESP32 (ESP32-S3, ESP32-C3 з ядром RISC-V), що пропонують різний набір периферії та продуктивності. Як альтернатива, могли б використовуватися мікроконтролери з зовнішніми Wi-Fi модулями. Наприклад, мікроконтролери STM32 можуть працювати в парі з модулями на базі чіпів Inventek eS-WiFi або Murata Wi-Fi/Bluetooth modules, що вимагало б додаткової розробки драйверів та інтеграції. Іншим варіантом є використання одноплатних комп'ютерів (SBC), таких як Raspberry Pi (наприклад, Pi Zero 2 W, Pi 3/4/5) або BeagleBone. Вони працюють під управлінням повноцінних операційних систем (зазвичай

Linux), що значно спрощує розробку веб-сервера та мережевих додатків, надаючи доступ до широкого стеку програмного забезпечення (Node.js, Python/Flask/Django, Apache/Nginx). Однак, SBC зазвичай мають вище енергоспоживання, більші габарити та вищу вартість порівняно з мікроконтролерними модулями, а також можуть вносити затримки, якщо використовуються для критичних по часу завдань управління. Для даного проекту, де основна обробка аудіо винесена на окремий МК, SBC міг би слугувати потужною платформою для користувацького інтерфейсу. Також існують інші мікроконтролери з інтегрованим Wi-Fi, наприклад, деякі моделі від Realtek (Ameba) або Cypress (тепер Infineon) PSoC 6 Wi-Fi BT, які пропонують свої переваги в плані периферії або енергоефективності.

Щодо аудіокодеків, які виконують цифро-аналогове перетворення (ЦАП) та часто аналого-цифрове (АЦП), вибір залежить від необхідної якості звуку, кількості каналів, наявності вбудованих підсилювачів та інтерфейсів. На ринку представлені кодеки від Cirrus Logic (наприклад, серії CS43xx, CS42xx), відомі своєю якістю звуку в споживчій та професійній аудіотехніці. Вони пропонують різні рівні інтеграції, включаючи підсилювачі для навушників та лінійні виходи. Texas Instruments пропонує широкий асортимент аудіокодеків під брендом Burr-Brown (наприклад, серія PCM) та інші (TLV320AIC, TASxxxx). Деякі з них мають вбудовані DSP, що дозволяє виконувати частину обробки сигналу безпосередньо на кодеку, або підсилювачі класу D для прямого підключення динаміків. Analog Devices також є значним гравцем на ринку аудіокомпонентів, пропонуючи високоякісні АЦП/ЦАП та кодеки (наприклад, серії AD19xx, ADAUxxxx). Деякі з їхніх продуктів, особливо серія SigmaDSP (наприклад, ADAU1701), містять потужні програмовані DSP, що дозволило б реалізувати всю логіку еквалайзера та іншої обробки безпосередньо на чіпі кодека, розвантаживши основний мікроконтролер. Компанія NXP Semiconductors (раніше Freescale) пропонує кодеки, такі як SGTL5000, що є популярним рішенням для

вбудованих систем завдяки хорошому співвідношенню ціна/якість та набору функцій. Вибір конкретного кодека визначається вимогами до розрядності (16, 24, 32 біт), частоти дискретизації (44.1 кГц, 48 кГц, 96 кГц, 192 кГц), співвідношення сигнал/шум (SNR), коефіцієнту нелінійних спотворень (THD) та наявності необхідних інтерфейсів (I2S, TDM, SPI, I2C).

Таким чином, ринок електронних компонентів надає широкий вибір альтернативних рішень для кожного з вузлів розроблюваної системи, що дозволяє гнучко підходити до проектування залежно від пріоритетів: продуктивності, вартості, енергоспоживання чи специфічних функціональних вимог.

1.3 Принципи функціонування та засоби забезпечення якості USB аудіоплеєра

Забезпечення заданих технічних характеристик комп'ютеризованої системи USB аудіоплеєра з підтримкою веб-еквалайзера було досягнуто шляхом комплексного підходу до вибору компонентів та реалізації програмно-апаратних рішень.

Ключовим аспектом функціонування пристрою є обробка аудіосигналу. Для прийому цифрового аудіопотоку з USB-накопичувача було інтегровано програмні бібліотеки для роботи з файловою системою FATFS, що дозволило зчитувати аудіофайли поширених форматів. Подальша обробка сигналу на мікроконтролері STM32F407VGT6 включає декодування та, що найважливіше, застосування цифрової фільтрації. Для реалізації функціоналу веб-еквалайзера було використано каскад біквадратичних фільтрів другого порядку. Коефіцієнти цих фільтрів динамічно розраховуються та оновлюються на STM32 на основі команд, отриманих від модуля ESP32-Wrover-Dev. Така архітектура дозволила забезпечити гнучке налаштування амплітудно-частотної характеристики аудіосигналу в реальному часі.

Передача даних між основними обчислювальними модулями системи була організована наступним чином: конфігураційні параметри для біквадратичних фільтрів (значення підсилення для обраних частотних смуг та загальний рівень гучності), встановлені користувачем через веб-інтерфейс, передаються з ESP32 на STM32 через послідовний інтерфейс USART (USART2). Для ефективної передачі аудіоданих між периферійними блоками мікроконтролера STM32 (наприклад, з USB-контролера в пам'ять та з пам'яті на інтерфейс I2S) було задіяно контролер прямого доступу до пам'яті (DMA). Це дозволило суттєво розвантажити центральний процесор, забезпечивши стабільну обробку аудіопотоку, особливо при роботі з стандартними частотами дискретизації, такими як 44.1 кГц або 48 кГц.

Для виведення обробленого аудіосигналу було передбачено використання зовнішнього аудіокодека, підключеного до STM32 через шину I2S для передачі цифрових аудіоданих та шину I2C для конфігурування параметрів кодека (наприклад, рівня гучності на виході, режимів роботи). Після цифро-аналогового перетворення на аудіокодеку, для усунення високочастотних артефактів квантування та формування кінцевого аналогового сигналу, як правило, застосовуються аналогові фільтри низької частоти, що часто інтегровані в сам кодек або реалізуються зовнішньою простою RC- чи LC-ланкою. Забезпечення належного рівня вихідного сигналу для підключення навушників або зовнішніх підсилювальних пристроїв покладається на вбудований в аудіокодек підсилювач або, за потреби, на зовнішні підсилювальні каскади.

Стабільність роботи всієї системи та якість аналогового сигналу значною мірою залежать від системи живлення. Було приділено увагу забезпеченню стабільного та малошумного живлення для всіх компонентів, особливо для аналогової частини аудіокодека, шляхом застосування відповідних стабілізаторів напруги (наприклад, LDO-регуляторів для чутливих вузлів). Точність тактування цифрових компонентів, зокрема

мікроконтролера STM32 та інтерфейсу I2S, забезпечується за допомогою вбудованого фазового автопідстроювання частоти (PLL), що є критичним для коректної обробки та відтворення аудіо з заданою частотою дискретизації та бітрейтом.

Інтерфейс користувача реалізовано на базі веб-сервера, розміщеного на модулі ESP32-Wrover-Dev, який також функціонує як точка доступу Wi-Fi. Це дозволило створити зручний та універсальний спосіб керування параметрами еквалайзера та гучністю з будь-якого пристрою, що має веб-браузер та можливість підключення до Wi-Fi.

На етапі проектування друкованої плати було враховано необхідність мінімізації перехресних завад та впливу електромагнітних випромінювань шляхом раціонального розміщення компонентів, екранування чутливих сигнальних ліній та організації належного заземлення. Ці заходи спрямовані на досягнення високої якості звучання та надійності роботи пристрою.

1.4 Обґрунтування вибору ключових апаратних компонентів та архітектури їх взаємодії

Для реалізації комп’ютеризованої системи USB аудіоплеєра з підтримкою веб-еквалайзера було обрано комбінацію з мікроконтролера STM32F407VGT6, модуля ESP32-Wrover-Dev та аудіокодека CS43L22. Такий вибір обґрунтовано специфічними завданнями, покладеними на кожен з компонентів, їхніми технічними характеристиками, доступністю засобів розробки та можливістю ефективної взаємодії.

Центральним елементом підсистеми обробки аудіосигналу виступає мікроконтролер STM32F407VGT6 (рис. 1.1). Його вибір зумовлений наявністю високопродуктивного ядра ARM Cortex-M4, що функціонує на тактовій частоті до 168 МГц та містить блок обробки чисел з плаваючою комою (FPU) і DSP-інструкції. Ці особливості є критично важливими для

ефективної реалізації алгоритмів цифрової обробки сигналів, зокрема, для розрахунку та застосування біквадратичних фільтрів, що формують основу веб-еквалайзера. Обсяг вбудованої Flash-пам'яті (1 МБ) є достатнім для зберігання програмного коду, включаючи драйвери USB, файлову систему та алгоритми обробки, а 192 КБ SRAM дозволяють буферизувати аудіодані та зберігати проміжні обчислення. Наявність периферійних інтерфейсів, таких як I2S для зв'язку з аудіокодеком, USB OTG для читання аудіофайлів з зовнішніх накопичувачів, та USART для комунікації з модулем ESP32, робить цей мікроконтролер оптимальним для поставлених задач. Використання контролера прямого доступу до пам'яті (DMA) для операцій з аудіобуферами (наприклад, передача даних з USB в пам'ять та з пам'яті на I2S) дозволяє значно знизити навантаження на центральний процесор [4].

Рисунок 1.1 – Мікроконтролер STM32F407VGT6

За реалізацію користувацького інтерфейсу, мережевої взаємодії та передачу налаштувань еквалайзера відповідає модуль ESP32-Wrover-Dev (рис. 1.2). Цей вибір обґрунтований інтегрованими можливостями Wi-Fi, що дозволяють модулю функціонувати як точка доступу та хостити веб-сервер.

На цьому веб-сервері розміщено сторінку з елементами керування веб-еквалайзером (повзунки для регулювання частотних смуг та загальної гучності). ESP32 обробляє запити користувача з веб-інтерфейсу, формує відповідні пакети даних з конфігурацією для фільтрів і надсилає їх на мікроконтролер STM32F407VGT6 через послідовний порт USART. Достатня обчислювальна потужність ESP32 та великий обсяг доступної пам'яті (завдяки PSRAM у версії Wrover) забезпечують стабільну роботу веб-сервера та швидку реакцію інтерфейсу [5].

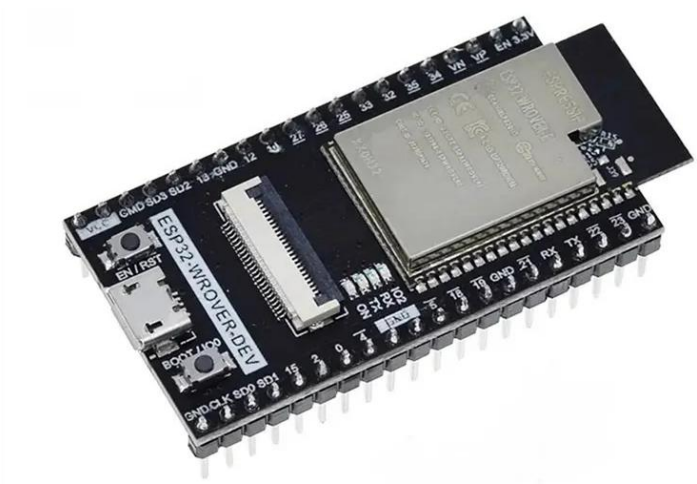

Рисунок 1.2 – Мікроконтролер ESP32-Wrover-Dev

Для перетворення цифрового аудіосигналу, обробленого на STM32, в аналоговий вихідний сигнал було обрано аудіокодек CS43L22. Цей компонент підтримує роботу з 24-бітним аудіо та частотами дискретизації до 96 кГц, що забезпечує високу якість відтворення. Наявність вбудованого підсилювача для навушників спрощує схемотехніку вихідного каскаду пристрою. Взаємодія між STM32F407VGT6 та CS43L22 відбувається через стандартні аудіоінтерфейси: I2S для передачі потоку аудіоданих та I2C для конфігурування параметрів кодека, таких як рівень гучності, режими роботи та налаштування вбудованих блоків [6].

Таким чином, обрана архітектура та компоненти дозволяють розділити

задачі: STM32F407VGT6 концентрується на критичній за часом обробці аудіо та роботі з USB, тоді як ESP32-Wrover-Dev забезпечує гнучкий та сучасний користувацький інтерфейс і мережеву комунікацію. Така комбінація забезпечує необхідну продуктивність, функціональність та можливість реалізації всіх поставлених у проекті завдань [7].

1.5 Обґрунтування вибору програмних засобів та бібліотек для розробки системи

Для розробки програмного забезпечення комп'ютеризованої системи USB аудіоплеєра з підтримкою веб-еквалайзера було застосовано відповідні середовища розробки та спеціалізовані бібліотеки для кожного з ключових мікроконтролерів системи – STM32F407VGT6 та ESP32-Wrover-Dev.

Програмне забезпечення для мікроконтролера STM32F407VGT6, на який покладено завдання обробки аудіосигналу, розроблялося з використанням офіційного інтегрованого середовища розробки STM32CubeIDE. Це середовище від STMicroelectronics надало комплексний набір інструментів, що включає конфігуратор STM32CubeMX для ініціалізації периферії та генерації початкового коду, а також компілятор, налагоджувач та редактор коду. Застосування STM32CubeIDE дозволило ефективно налаштувати периферійні модулі мікроконтролера, такі як USB OTG, I2S, I2C, USART та DMA, що є критично важливим для функціонування аудіоплеєра. В рамках розробки для STM32F407VGT6 була інтегрована бібліотека FAT_FS для роботи з файловою системою на USB-накопичувачі, що дозволило реалізувати зчитування аудіофайлів. Для забезпечення функціоналу USB Host, необхідного для підключення флеш-накопичувачів, були задіяні відповідні стеки USB Host Library з пакету STM32Cube Middleware [8].

Декодування аудіофайлів формату MP3 здійснювалося за допомогою бібліотеки HELIX MP3 Decoder, оптимізованої для вбудованих систем, що

забезпечило ефективне перетворення стиснених аудіоданих у PCM-потік. Взаємодія з аудіокодеком CS43L22 реалізовувалася через спеціалізований драйвер, який забезпечував конфігурування кодека через шину I2C та управління передачею аудіоданих через шину I2S.

Розробка програмного забезпечення для модуля ESP32-Wrover-Dev, що виконує функції веб-сервера та точки доступу Wi-Fi для веб-еквалайзера, здійснювалася в середовищі Arduino IDE з використанням відповідного пакету підтримки для плат ESP32 (ESP32 Arduino Core). Такий вибір був зумовлений простотою налаштування та великою кількістю доступних бібліотек для роботи з мережевими функціями. Для реалізації веб-сервера та обробки HTTP-запитів на ESP32 була використана стандартна бібліотека <WiFi.h> для створення точки доступу Wi-Fi. Робота з TCP-з'єднаннями забезпечувалася бібліотекою <AsyncTCP.h>, яка надає асинхронні функції, необхідні для функціонування веб-сервера без блокування основного циклу програми. Для створення самого асинхронного веб-сервера була задіяна бібліотека <ESPAsyncWebServer.h>. Її використання дозволило ефективно обробляти HTTP-запити, віддавати статичні файли веб-інтерфейсу еквалайзера та приймати дані від користувача для подальшої передачі на STM32, забезпечуючи швидку реакцію інтерфейсу [9].

Вибір зазначених програмних засобів та бібліотек був продиктований їхньою відповідністю функціональним вимогам проекту, стабільністю та доступністю. STM32CubeIDE забезпечило глибоке налаштування апаратних ресурсів STM32 для обробки аудіо, тоді як Arduino IDE з відповідними бібліотеками дозволило швидко реалізувати мережевий функціонал та користувацький веб-інтерфейс на ESP32.

РОЗДІЛ 2 ПРОЄКТУВАННЯ СИСТЕМИ USB АУДІОПЛЕЄРА

2.1 Структурна схема комп’ютеризованої системи USB аудіоплеєра та опис її функціональних блоків

Розробка комп’ютеризованої системи USB аудіоплеєра з підтримкою веб-еквалайзера передбачає взаємодію кількох основних функціональних блоків, що забезпечують обробку аудіосигналів та керування параметрами відтворення. Структурна схема розробленої системи представлена на рисунку 2.1.

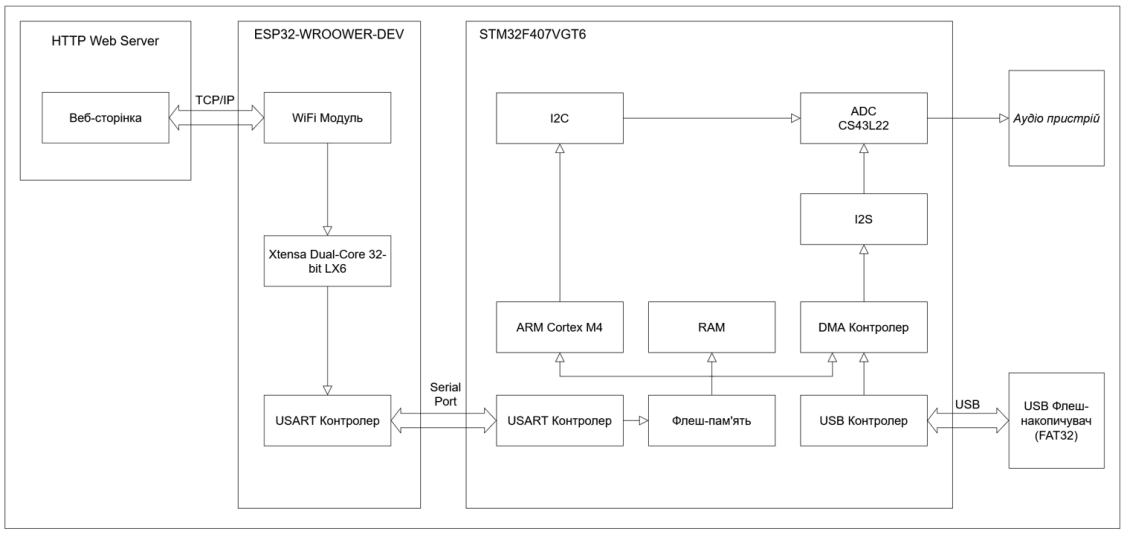

Рисунок 2.1 – Структурна схема комп’ютеризованої системи

В основі системи лежать два ключові мікроконтролерні модулі: STM32F407VGT6, відповідальний за безпосереднє відтворення та обробку аудіо, та ESP32-Wrover-Dev, що забезпечує функціонал веб-сервера та користувацького інтерфейсу еквалайзера.

Процес відтворення аудіо починається з підключення USB флеш-накопичувача, відформатованого у файловій системі FAT32, до USB-

інтерфейсу мікроконтролера STM32F407VGT6. Вбудований USB контролер STM32, використовуючи технологію прямого доступу до пам'яті (DMA), здійснює передачу даних з накопичувача безпосередньо до оперативної пам'яті (RAM) мікроконтролера. Такий підхід, особливо ефективний при використанні режиму USB High Speed, дозволяє мінімізувати навантаження на центральний процесор ARM Cortex-M4. Одночасно з завантаженням даних в RAM, ядро ARM Cortex-M4 розпочинає процес декодування аудіофайлів (наприклад, формату MP3), які порційно завантажуються в спеціальний буфер в RAM. Внутрішня флеш-пам'ять мікроконтролера STM32F407VGT6 використовується для зберігання програмного коду прошивки.

Після декодування та застосування цифрових фільтрів еквалайзера, аудіодані у форматі PCM (імпульсно-кодова модуляція) також за допомогою DMA контролера передаються з RAM на інтерфейс I2S. Далі, через шину I2S, цифровий аудіопотік надходить на аудіокодек CS43L22. Цей аудіокодек виконує цифро-аналогове перетворення сигналу та виводить його на стандартний аудіороз'єм 3.5 мм для підключення зовнішнього аудіопристрою (наприклад, навушників або активних колонок). Конфігурування параметрів аудіокодека CS43L22, таких як рівень гучності чи інші специфічні налаштування, здійснюється мікроконтролером STM32F407VGT6 через інтерфейс I2C. Використання окремих інтерфейсів для передачі даних (I2S) та управління (I2C) сприяє зменшенню взаємних завад та забезпечує стабільність аудіопотоку.

Паралельно, модуль ESP32-Wrover-Dev, що оснащений процесором Xtensa Dual-Core 32-bit LX6 та вбудованим Wi-Fi модулем, функціонує як точка доступу та хостить HTTP веб-сервер. Користувачі підключаються до цієї Wi-Fi мережі та через веб-браузер отримують доступ до веб-сторінки, яка є інтерфейсом веб-еквалайзера. Зміни положення повзунків на цій веб-сторінці, що відповідають за налаштування частотних смуг та загальної гучності, обробляються на стороні ESP32. Сформовані дані з новими параметрами

еквалайзера передаються з USART контролера модуля ESP32 через послідовний порт (Serial Port) на USART контролер мікроконтролера STM32F407VGT6. Отримавши ці дані, ядро ARM Cortex-M4 на STM32 оновлює коефіцієнти біквадратичних фільтрів, що призводить до зміни частотної характеристики аудіосигналу, який відтворюється в реальному часі.

Запропонована структурна схема забезпечує розділення завдань між двома мікроконтролерами, де STM32F407VGT6 виконує інтенсивні операції з обробки аудіо, а ESP32-Wrover-Dev реалізує зручний бездротовий користувацький інтерфейс, що дозволяє гнучко керувати процесом відтворення [10].

2.2 Побудова та опис схеми електричної принципової та схеми з’єднань USB аудіоплеєра

Ретельне схемотехнічне проектування є фундаментальним, оскільки саме на ньому закладаються основи для коректної та стабільної взаємодії всіх апаратних компонентів, забезпечується належне живлення, точне тактування та безперешкодна передача сигналів. Для даної системи, що поєднує інтенсивну обробку аудіоданих, мережеву взаємодію та користувацький інтерфейс, особливої уваги потребувала інтеграція двох ключових мікроконтролерних модулів – STM32F407VGT6 та ESP32-Wrover-Dev, а також їх взаємодія з периферійними пристроями, такими як USB-накопичувач та аудіокодек. Центральним вузлом, відповідальним за основні обчислювальні операції, пов'язані з аудіо, є мікроконтролер STM32F407VGT6, інтегрований на плату розробника STM32F407G-DISC1. Важливими є і вбудовані периферійні модулі: USB OTG для роботи з зовнішніми накопичувачами, інтерфейс I2S для високоякісної передачі аудіоданих на кодек, I2C для конфігурування аудіокодека та USART для комунікації з модулем ESP32. Забезпечення стабільного живлення для ядра мікроконтролера та його

периферії, правильне розведення ліній тактування для уникнення джиттера, а також коректне підключення ліній даних усіх використовуваних інтерфейсів є першочерговими завданнями схемотехнічного проектування. Для розуміння реалізації цих аспектів необхідно детально розглянути відповідні фрагменти електричної принципової схеми самої плати STM32F407G-DISC1, а також принципову схему модуля ESP32-Wrover-Dev, що дозволить оцінити особливості їх інтеграції в єдину систему. Лише після аналізу цих принципових рішень можна переходити до розгляду загальної схеми електричних з'єднань, яка візуалізує фізичну взаємодію всіх компонентів пристрою.

На рисунку 2.2 представлено фрагмент електричної принципової схеми, що стосується безпосередньо мікроконтролерного вузла (MCU) на платі STM32F407G-DISC1. Для потенційної взаємодії з користувачем, наприклад, для керування відтворенням, може бути задіяний вивід PA0, до якого зазвичай підключається кнопка.

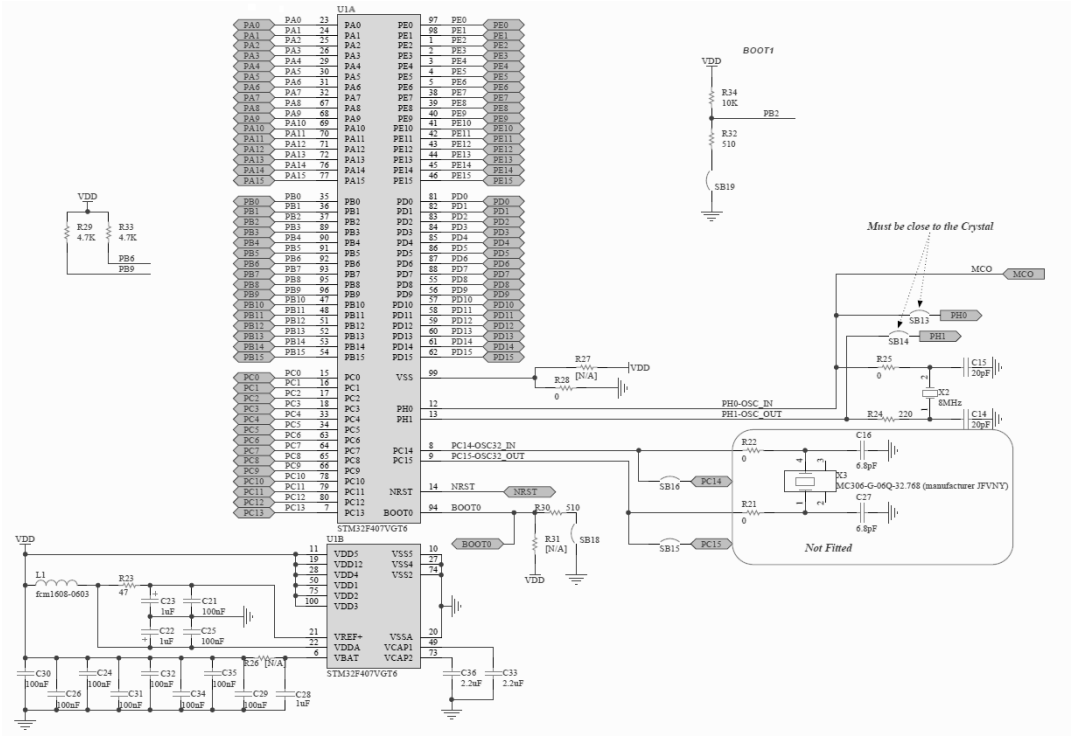

Рисунок 2.2 – Схема електрична принципова MCU
(фрагмент плати STM32F407G-DISC1)

Процес обробки та виведення аудіосигналу реалізовано за допомогою інтегрованого на плату STM32F407G-DISC1 аудіокодека CS43L22. Принципова схема підключення аудіокодека наведена на рисунку 2.3. Передача цифрового аудіосигналу, на аудіокодек відбувається через спеціалізований послідовний інтерфейс I2S3. Для цього використовуються виводи мікроконтролера PC7, що слугує для передачі основного тактового сигналу (MCLK – Master Clock), PC10 для послідовного тактового сигналу (SCK – Serial Clock), PC12 для передачі самих аудіоданих (SD – Serial Data) та PA4 для сигналу вибору слова (WS – Word Select). Коректна конфігурація виводу PC7, зокрема його тактової частоти, є критично важливою, оскільки саме цей сигнал використовується аудіокодеком для внутрішньої синхронізації процесів цифро-аналогового перетворення. Конфігурація аудіокодека CS43L22 реалізовано через двопровідну послідовну шину I2C. Для цього задіяні виводи мікроконтролера PB9 (SDA – Serial Data) та PB6 (SCL – Serial Clock). Після цифро-аналогового перетворення, сформований аналоговий аудіосигнал з виходів кодека, позначених як HP/LINE_OUTA та HP/LINE_OUTB, подається на стандартний 3.5 мм аудіороз'єм.

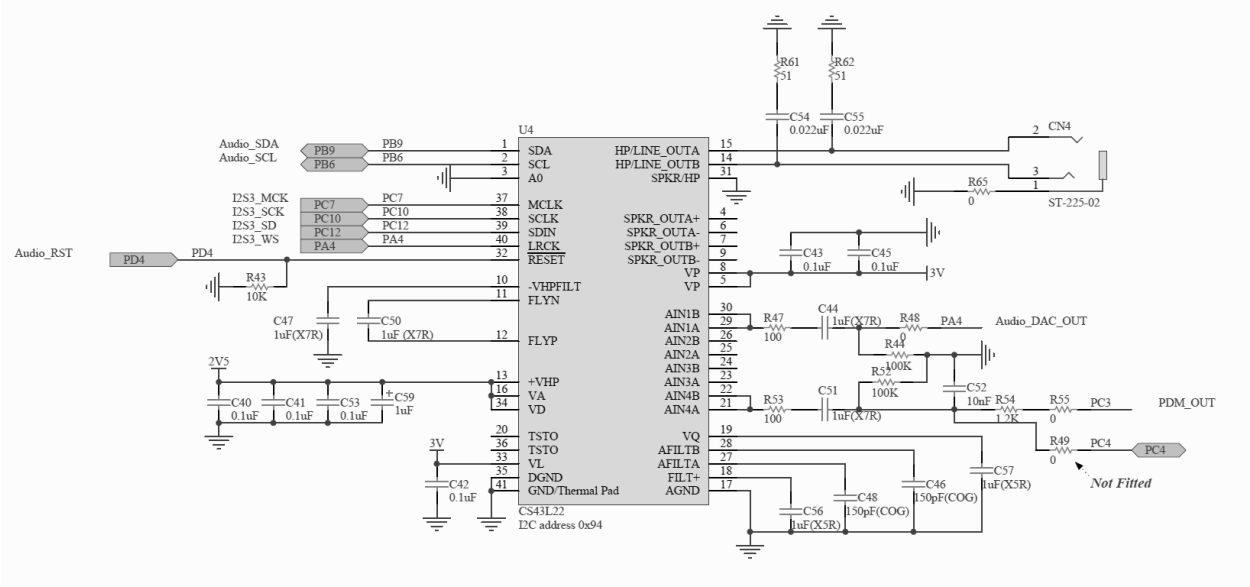

Рисунок 2.3 – Схема електрична принципова аудіокодеку CS43L22 (фрагмент плати STM32F407G-DISC1)

Для забезпечення можливості зчитування аудіофайлів з зовнішніх USB-накопичувачів в системі на базі мікроконтролера STM32F407VGT6 було реалізовано функціонал USB On-The-Go (OTG). Ця технологія дозволяє пристрою, в даному випадку мікроконтролеру, виступати як в ролі USB-хоста , так і в ролі USB-пристрою. У контексті даного проекту ключовою є саме роль USB-хоста, яка дає змогу підключати флеш-накопичувачі та отримувати доступ до їх файлової системи. На рисунку 2.4 детально показано схемотехнічне рішення для підключення роз'єму micro-USB, що використовується для реалізації цього OTG-інтерфейсу.

Центральними лініями для передачі даних у будь-якому USB-інтерфейсі є диференціальна пара сигналів D- та D+. У випадку мікроконтролера STM32F407VGT6, для повношвидкісного (Full-Speed) режиму USB OTG, ці функції виконують виводи PA11 (OTG_FS_DM) та PA12 (OTG_FS_DP). Ці виводи мікроконтролера безпосередньо підключені до відповідних контактів D- та D+ на фізичному micro-USB роз'ємі, забезпечуючи шлях для обміну інформацією з підключеним USB-пристроєм. Для коректної роботи в режимі USB-хоста, а також для забезпечення живлення підключеного периферійного пристрою, наприклад, флеш-накопичувача, якому для функціонування зазвичай необхідне стандартне живлення 5В по шині USB, передбачено спеціальний механізм керування лінією VBUS. Для активації VBUS, тобто для подачі живлення на підключений USB-пристрій, вивід PC0 мікроконтролера STM32F407VGT6 повинен бути встановлений у низький логічний рівень. Така можливість програмного керування живленням USB-периферії є важливою для енергозбереження та для правильної послідовності ініціалізації пристроїв.

Також на схемі роз'єму micro-USB можна побачити контакт ID. Цей контакт відіграє ключову роль у функціоналі USB OTG, дозволяючи пристрою автоматично визначати свою роль при підключенні. Залежно від того, як цей пін підключений у кабелі (закорочений на землю для ідентифікації пристрою як хоста або залишений вільним), мікроконтролер може автоматично

перемикатися між режимами USB-хоста та USB-пристрою. Хоча в рамках даного проекту мікроконтролер використовується переважно в режимі хоста для читання даних з флеш-накопичувача, наявність цього функціоналу на апаратному рівні забезпечує потенційну гнучкість для майбутніх розширень. Належне схемотехнічне підключення цих ліній (D+, D-, VBUS_Control, ID) та їх коректна програмна конфігурація є запорукою стабільної та надійної роботи USB-інтерфейсу в режимі хоста.

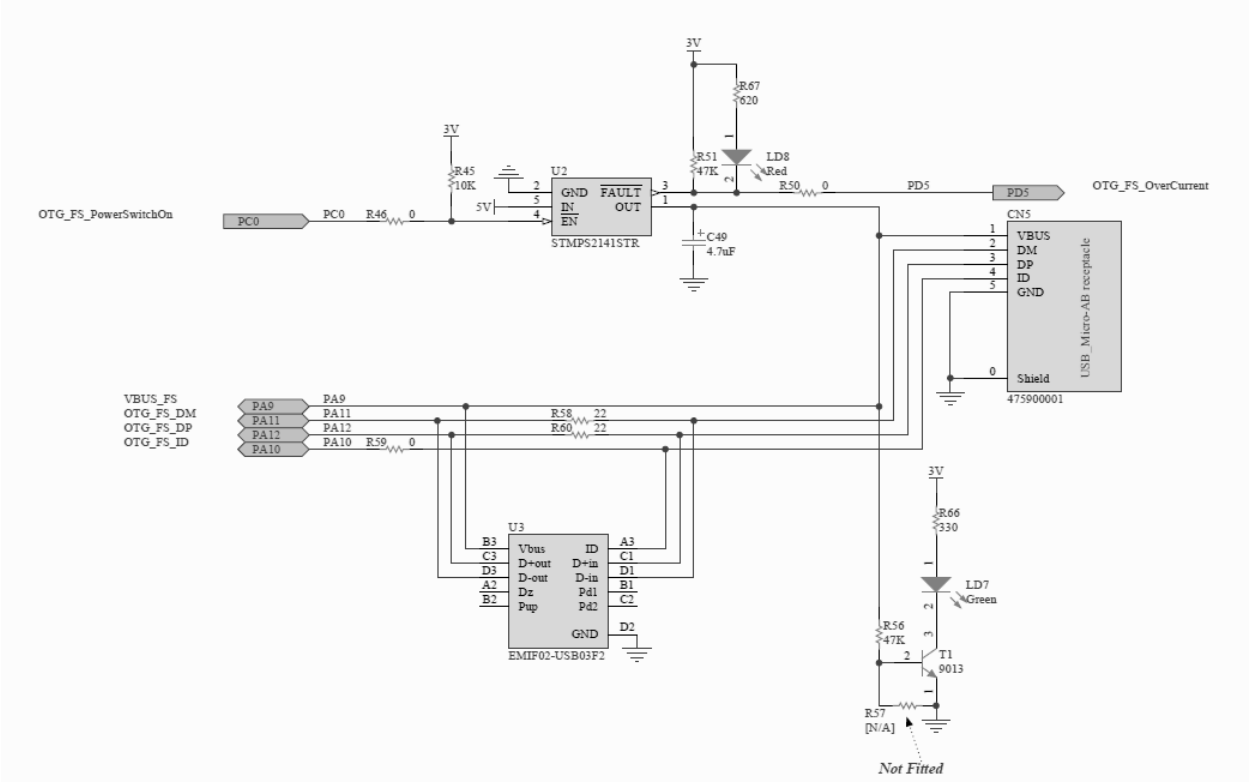

Рисунок 2.4 – Схема електрична принципова micro-USB
(фрагмент плати STM32F407G-DISC1)

Модуль ESP32-Wrover-Dev, який виконує функції веб-сервера та забезпечує користувацький інтерфейс для керування еквалайзером, також має власну електричну принципову схему, ключові аспекти якої представлені на рисунку 2.5. Ця схема деталізує підключення мікроконтролера ESP32, інтегрованого модуля Wi-Fi, мікросхем пам'яті (Flash та PSRAM) та основних інтерфейсних виводів. Для взаємодії з мікроконтролером STM32F407VGT6 з

метою передачі налаштувань еквалайзера та команд керування гучністю використовуються виводи послідовного порту USART. Зокрема, вивід передавача (TX) модуля ESP32 з'єднується з виводом приймача PA3 (RX) мікроконтролера STM32, а вивід приймача (RX) модуля ESP32, відповідно, підключається до виводу передавача PA2 (TX) мікроконтролера STM32. Живлення самого модуля ESP32-Wrover-Dev здійснюється через його власний USB-роз'єм, який також може використовуватися для програмування та налагодження.

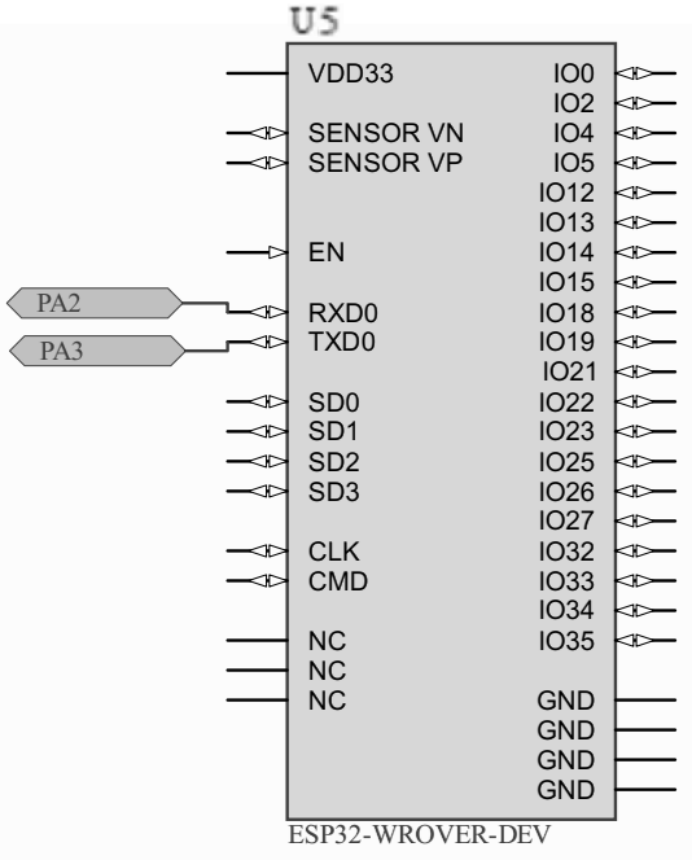

Рисунок 2.5 – Схема електрична принципова ESP32-Wrover-Dev

Після детального розгляду принципових аспектів функціонування окремих вузлів системи, доцільно перейти до аналізу загальної схеми електричних з'єднань, яка наочно демонструє фізичну інтеграцію всіх перерахованих компонентів в єдиний функціональний пристрій. Ця схема, представлена на рисунку 2.6, візуалізує шляхи проходження сигналів та

живлення між модулем обробки аудіо A2 (плата STM32F407G-DISC1), модулем веб-інтерфейсу A3 (плата ESP32-Wrover-Dev), джерелом аудіофайлів A1 (флеш-накопичувач), аудіопристроєм B1 (динамік або навушники) та джерелом живлення G1 (батарейний блок, павербанк). Флеш-накопичувач A1 підключається до плати A2 за допомогою кабелю-адаптера USB OTG W4. Плата A2, в свою чергу, з'єднується з аудіопристроєм B1 через кабель W3, що підключається до її 3.5 мм аудіороз'єму. Модуль A3 (ESP32-Wrover-Dev) комунікує з платою A2 через лінії USART. Живлення модуля A3 може бути забезпечене від зовнішнього джерела, наприклад, батарейного блоку G1, через один з USB-роз'ємів X1 (USB Type-A breakout) або X2 (Micro-USB breakout) та відповідні кабелі W1 або W2. Така комплексна схема з'єднань є основою для фізичної реалізації розробленого USB аудіоплеєра. Важливим аспектом при практичній реалізації є забезпечення якісного та стабільного живлення для всіх компонентів, а також мінімізація можливих електромагнітних завад шляхом раціонального розміщення елементів та, за потреби, екранування сигнальних ліній, що є запорукою надійного функціонування пристрою та високої якості відтворюваного звуку.

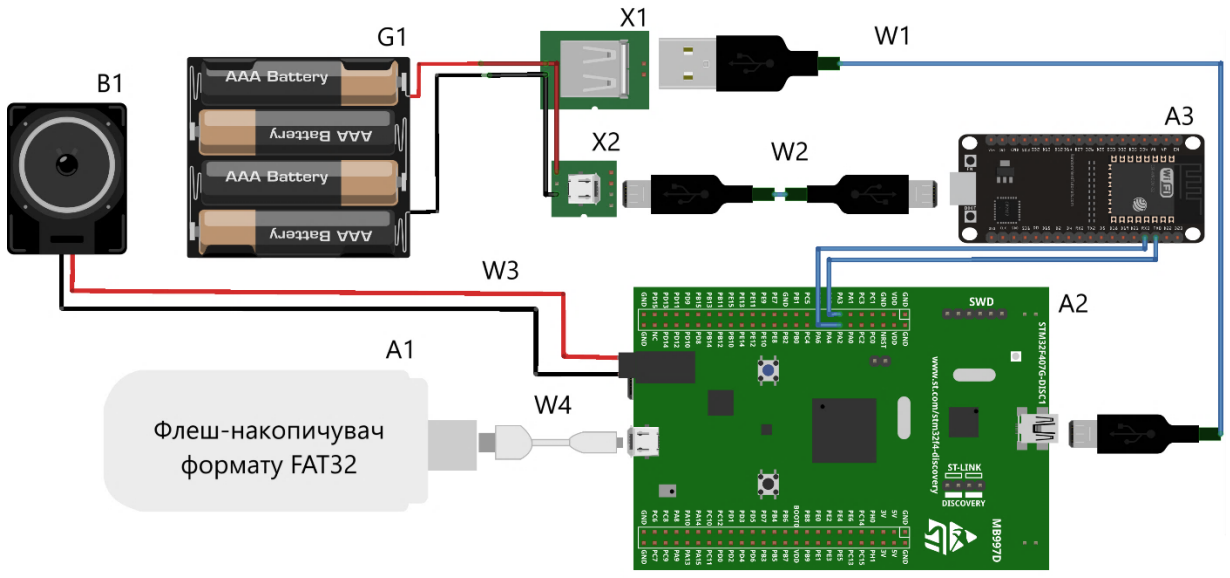

Рисунок 2.6 – Схема електричних з’єднань USB-аудіоплеєра

2.3 Інтерфейси передачі даних та керування в комп'ютеризованій системі USB аудіоплеєра

Функціонування комп'ютеризованої системи USB аудіоплеєра з підтримкою веб-еквалайзера, що поєднує в собі мікроконтролер STM32F407VGT6 для аудіообробки, модуль ESP32-Wrover-Dev для мережевої взаємодії та аудіокодек CS43L22 для якісного звуковідтворення, критично залежить від злагодженої та ефективної взаємодії цих компонентів. Ця взаємодія реалізується через ретельно підібраний набір стандартизованих шин та протоколів передачі даних.

Центральним елементом у тракті передачі аудіо є послідовний інтерфейс I2S (Inter-IC Sound), який було задіяно для транспортування цифрового аудіосигналу між основним оброблювальним мікроконтролером STM32F407VGT6 та аудіокодеком CS43L22. У даній конкретній реалізації системи, мікроконтролер STM32F407VGT6 (а саме його периферійний блок I2S3) виступає в ролі ведучого пристрою (master). Це означає, що STM32 генерує та надає всі необхідні тактові сигнали для синхронізації передачі. До цих сигналів належать: послідовний тактовий сигнал (SCK, також відомий як BCLK – Bit Clock), який тактує кожен біт даних, що передається; сигнал вибору слова (WS, також відомий як LRCK – Left/Right Clock або FS – Frame Sync), який вказує, для якого каналу (лівого чи правого у стереорежимі) передаються поточні дані, а також визначає початок кожного аудіосемплу (фрейму); та, за потреби, основний тактовий сигнал (MCLK – Master Clock) Часову діаграму для шини I2S модна переглянути на рисунку 2.7. MCLK часто є кратною частоті дискретизації і використовується аудіокодеком CS43L22 для внутрішньої синхронізації його цифро-аналогових забезпечуючи точне відновлення аналогового сигналу. Аудіодані передаються послідовно, біт за бітом, у форматі PCM від мікроконтролера STM32 до аудіокодека CS43L22. Протокол I2S за своєю природою забезпечує синхронну передачу даних, що є

абсолютно критичним для збереження цілісності, мінімізації спотворень та запобігання виникненню джиттера (фазового тремтіння тактових сигналів), що в кінцевому підсумку гарантує високу якість відтворюваного аудіосигналу. Правильне налаштування параметрів протоколу I2S на обох сторонах (STM32 та CS43L22), таких як формат даних (наприклад, I2S Philips standard, MSB-justified, LSB-justified), розрядність даних та частота дискретизації, є обов'язковою умовою для коректної роботи аудіотракту [11].

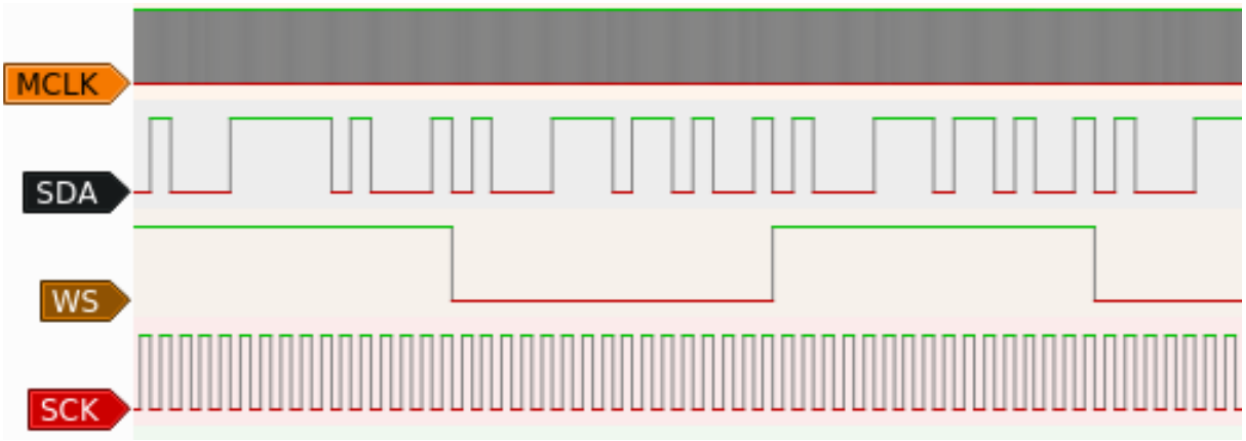

Рисунок 2.7 – Часова діаграма для протоколу передачі даних I2S

Для взаємодії з зовнішнім USB флеш-накопичувачем, з якого зчитуються аудіофайли, використовується універсальна послідовна шина USB. Мікроконтролер STM32F407VGT6, завдяки вбудованому контролеру USB OTG (On-The-Go), функціонує в режимі USB Host (рис. 2.8). Це дозволяє йому ініціювати зв'язок з підключеним накопичувачем, розпізнавати його файлову систему (FAT32) та зчитувати дані. Протокол USB забезпечує стандартизований механізм обміну даними, включаючи команди для доступу до файлів та секторів накопичувача. Також USB-інтерфейс на платі STM32F407G-DISC1 (через ST-LINK) та на модулі ESP32-Wrover-Dev використовується для програмування мікроконтролерів та їх налагодження з персонального комп'ютера [12].

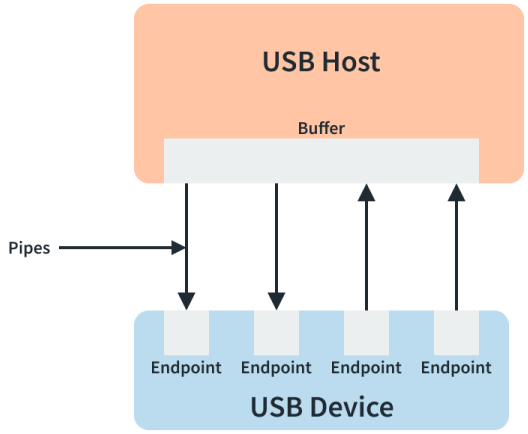

Рисунок 2.8 – Модель потоку даних протоколу передачі даних USB

Конфігурування параметрів аудіокодека CS43L22, таких як рівень гучності, частота дискретизації, режими роботи аналогових виходів та інші специфічні налаштування, здійснюється мікроконтролером STM32F407VGT6 через двопровідний послідовний інтерфейс I2C. Ця шина використовує дві лінії: лінію даних (SDA - Serial Data) та лінію тактування (SCL - Serial Clock). Обмін даними відбувається в режимі "ведучий-ведений", де STM32 виступає ведучим, а аудіокодек CS43L22 – веденим пристроєм з унікальною адресою на шині (рис. 2.9). Незважаючи на відносно нижчу швидкість порівняно з протоколом I2S, для передачі невеликих обсягів конфігураційних даних I2C є ефективним та простим у реалізації рішенням [13].

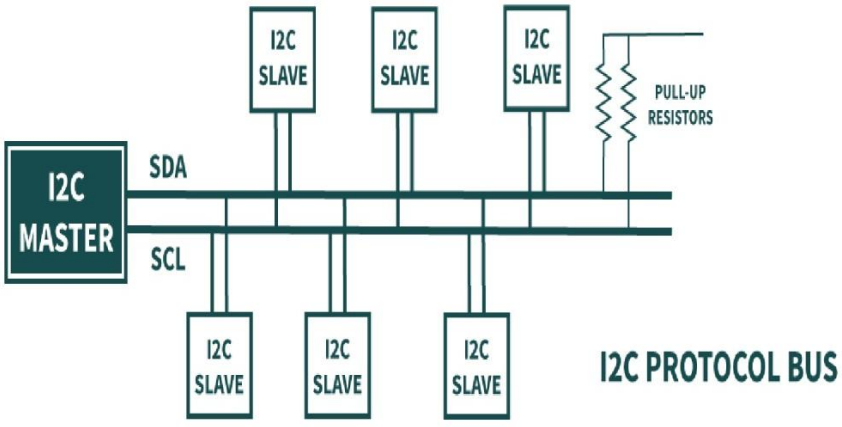

Рисунок 2.9 – Схема підключення пристроїв за допомогою протоколу передачі даних I2C

Обмін даними між мікроконтролером STM32F407VGT6 та модулем ESP32-Wrover-Dev, зокрема передача налаштувань еквалайзера та команд керування гучністю, реалізовано за допомогою універсального синхронного/асинхронного приймально-передавача USART. Цей послідовний інтерфейс використовує дві основні лінії: RX (Receive Data) для прийому даних та TX (Transmit Data) для передачі. В даному проекті було налаштовано асинхронний режим роботи, де дані передаються у вигляді пакетів, що містять стартовий біт, біти даних, опціональний біт парності та стоп-біти. Враховуючи, що в проекті підключені лише порти RX та TX, то можна вважати, що USART працює в режимі UART. Різницю можна побачити на рисунку 2.10. Швидкість передачі (baud rate) та інші параметри USART були узгоджені між обома пристроями для забезпечення коректної комунікації.

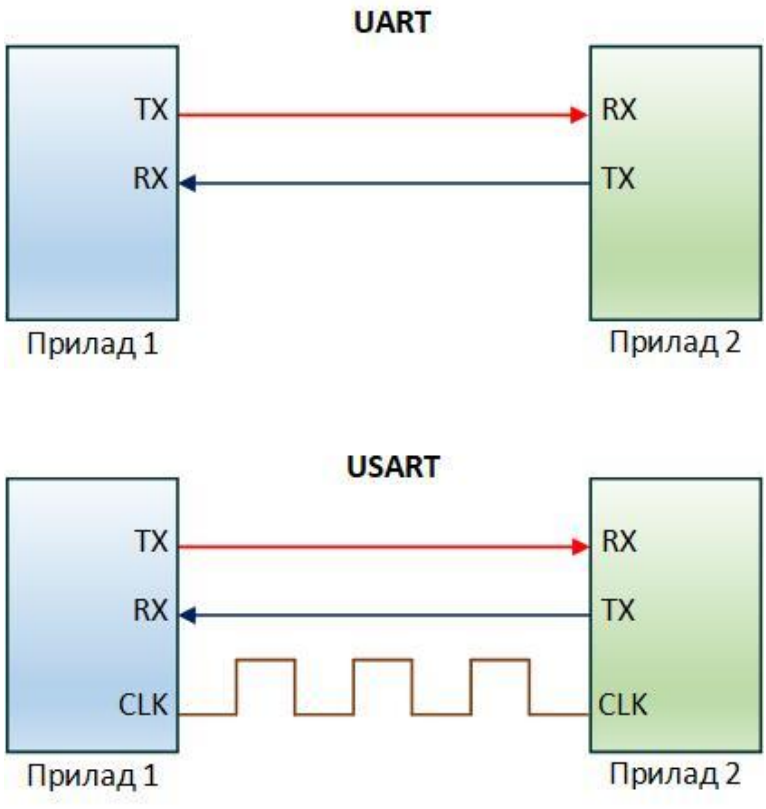

Рисунок 2.10 – Схема підключення пристроїв за допомогою інтерфейсів USART та UART

Для реалізації бездротового користувацького інтерфейсу веб-еквалайзера модуль ESP32-Wrover-Dev використовує вбудований контролер Wi-Fi. ESP32 функціонує як точка доступу, створюючи локальну бездротову мережу. Користувацькі пристрої (смартфони, планшети, комп'ютери) підключаються до цієї мережі. Взаємодія з веб-сервером, розміщеним на ESP32, відбувається за допомогою стеку протоколів TCP/IP, зокрема протоколу HTTP для обміну даними між веб-браузером користувача та веб-сервером на ESP32.

Для обробки сигналів від простих елементів керування, наприклад, кнопки перемикання треків, використовуються лінії введення/виведення загального призначення GPIO.

Важливу роль у підвищенні продуктивності системи відіграє контролер DMA, наявний в мікроконтролері STM32F407VGT6. Технологія DMA дозволяє передавати великі обсяги даних між периферійними пристроями (такими як USB-контролер або інтерфейс I²S) та оперативною пам'яттю безпосередньо, минаючи центральний процесор (рис. 2.11). У даному проекті DMA активно використовується для зчитування даних з USB флеш-накопичувача в буфери RAM, а також для передачі оброблених аудіоданих з RAM на інтерфейс I2S для відправки на аудіокодек. Це значно розвантажує CPU, дозволяючи йому виконувати інші задачі, такі як декодування аудіо, обчислення коефіцієнтів фільтрів та обробка команд з ESP32, забезпечуючи при цьому безперервний та плавний потік аудіоданих.

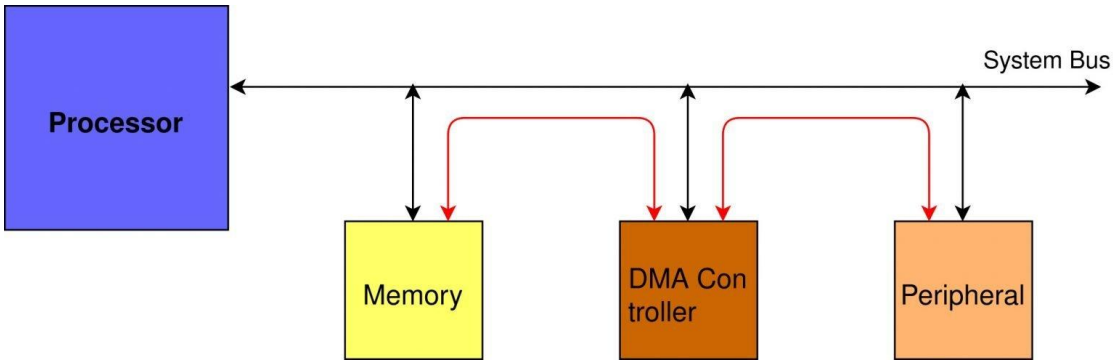

Рисунок 2.11 – Схема роботи DMA

Комплексне використання зазначених шин та протоколів забезпечує ефективну взаємодію всіх апаратних та програмних компонентів системи, дозволяючи реалізувати весь запланований функціонал комп’ютеризованого USB аудіоплеєра з веб-еквалайзером.

2.4 Теоретичні основи цифрової фільтрації та біквадратичні фільтри для реалізації еквалайзера

Цифрова фільтрація є невід'ємною частиною сучасних систем обробки сигналів, зокрема аудіосистем, де вона використовується для зміни частотного спектру сигналу з метою покращення якості звучання, усунення небажаних компонентів або створення специфічних звукових ефектів. Цифрові фільтри обробляють дискретизовані в часі сигнали і можуть бути реалізовані програмно на мікроконтролерах або спеціалізованих процесорах цифрових сигналів (DSP).

Залежно від характеру їх імпульсної характеристики, цифрові фільтри поділяються на два основні класи: фільтри зі скінченною імпульсною характеристикою (СІХ-фільтри, англ. Finite Impulse Response, FIR) та фільтри з нескінченною імпульсною характеристикою (НІХ-фільтри, англ. Infinite Impulse Response, IIR). СІХ-фільтри завжди стійкі та можуть мати лінійну фазову характеристику, однак для досягнення гострої частотної вибірковості вони часто потребують значно вищого порядку (і, відповідно, більших обчислювальних ресурсів) порівняно з НІХ-фільтрами. НІХ-фільтри, у свою чергу, можуть забезпечити необхідну частотну характеристику при значно меншому порядку, що робить їх привабливими для реалізації у вбудованих системах з обмеженими ресурсами, хоча питання їх стійкості потребує уваги при проектуванні.

Одним з найпоширеніших типів НІХ-фільтрів, що широко використовується для реалізації еквалайзерів, є біквадратичний фільтр. Це

цифровий фільтр другого порядку, назва якого походить від того, що його передавальна функція у z-області є відношенням двох квадратичних поліномів (біквадратичних функцій) відносно z^{-1}:

$$H(z) = Y(z) / X(z) = (b_0 + b_1z^{-1} + b_2z^{-2}) / (1 + a_1z^{-1} + a_2z^{-2}), \qquad (2.1)$$

де X(z) та Y(z) – Z-перетворення вхідного та вихідного сигналів відповідно;
b_0, b_1, b_2 – коефіцієнти чисельника, що визначають нулі передавальної функції;
a_1, a_2 – коефіцієнти знаменника, що визначають полюси передавальної функції (коефіцієнт a_0 зазвичай нормалізується до 1).

Різницеве рівняння, що описує роботу біквадратичного фільтра в часовій області є відповідним до формули:

$$y[n] = b_0x[n] + b_1x[n-1] + b_2x[n-2] - a_1y[n-1] - a_2y[n-2], \qquad (2.2)$$

де y[n] – поточний вихідний відлік;
x[n] – поточний вхідний відлік;
y[n-1], y[n-2] – попередні вихідні відліки;
x[n-1], x[n-2] – попередні вхідні відліки.

Значення коефіцієнтів (b_0, b_1, b_2, a_1, a_2) повністю визначають амплітудно-частотну (АЧХ) та фазо-частотну (ФЧХ) характеристики фільтра. Шляхом відповідного вибору цих коефіцієнтів на основі біквадратичної структури можна реалізувати різноманітні стандартні типи фільтрів, такі як:

– Фільтр нижніх частот (ФНЧ, Low-pass filter, LPF): пропускає частоти нижче певної частоти зрізу та ослаблює частоти вище неї.

– Фільтр верхніх частот (ФВЧ, High-pass filter, HPF): пропускає частоти вище певної частоти зрізу та ослаблює частоти нижче неї.

– Смуговий фільтр (Band-pass filter, BPF): пропускає частоти в межах певної смуги та ослаблює частоти поза нею.

– Режекторний (загороджувальний) фільтр (Band-stop filter, BSF або Notch filter): ослаблює частоти в межах певної смуги та пропускає частоти поза нею.

– Піковий (Peaking) або провалюючий (Notch) фільтр еквалайзера: підсилює або ослаблює вузьку смугу частот навколо центральної частоти.

– Шельфовий фільтр нижніх частот (Low-shelf filter): підсилює або ослаблює усі частоти нижче певної частоти на задану величину.

– Шельфовий фільтр верхніх частот (High-shelf filter): підсилює або ослаблює усі частоти вище певної частоти на задану величину.

– Всепропускаючий фільтр (All-pass filter): не змінює АЧХ, але змінює ФЧХ сигналу.

Для реалізації фільтрів вищих порядків або складних частотних характеристик, таких як у багатосмуговому еквалайзері, окремі біквадратичні секції часто з'єднують каскадно. При каскадному включенні вихід попередньої біквадратичної секції стає входом для наступної. Загальна передавальна функція такого каскадного фільтра є добутком передавальних функцій окремих секцій. Це дозволяє розбити складне завдання проектування фільтра високого порядку на проектування кількох простіших фільтрів другого порядку, що спрощує розрахунок коефіцієнтів та може покращити стійкість фільтра до помилок квантування коефіцієнтів. У контексті даного проекту, 10-смуговий еквалайзер реалізовано саме шляхом каскадного з'єднання десяти незалежно керованих біквадратичних фільтрів.

Вибір біквадратичних НІХ-фільтрів для реалізації еквалайзера в цій роботі зумовлений їх високою обчислювальною ефективністю порівняно з СІХ-фільтрами аналогічної вибірковості, а також наявністю добре розроблених методик розрахунку коефіцієнтів для стандартних типів частотних характеристик, необхідних для еквалізації.

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ КОМП'ЮТЕРИЗОВАНОЇ СИСТЕМИ

3.1 Опис алгоритму програми вбудованої системи

Програмна реалізація комп'ютеризованої системи USB аудіоплеєра з веб-еквалайзером побудована на послідовності логічних кроків, що охоплюють ініціалізацію апаратних та програмних компонентів, обробку зовнішніх подій, керування процесом відтворення аудіофайлів та забезпечення взаємодії між різними функціональними модулями системи. Алгоритми роботи ключових частин програми візуалізовані за допомогою блок-схем.

Робота системи починається з етапу ініціалізації, як показано на рисунку 3.1. На цьому етапі відбувається конфігурація системного тактового генератора (SystemClock_Config), що встановлює необхідні частоти для роботи мікроконтролера STM32F407VGT6 та його периферійних модулів. Далі слідує ініціалізація ключових периферійних пристроїв та системних служб. Це включає налаштування біквадратичних фільтрів еквалайзера, запуск системного таймера SysTick для генерації мілісекундних переривань, конфігурацію портів вводу-виводу загального призначення (GPIO) для світлодіодної індикації та обробки натискання кнопки, ініціалізацію бібліотеки USB Host для взаємодії з флеш-накопичувачами та налаштування модуля USART2 для комунікації з ESP32.

Після завершення всіх підготовчих операцій програма переходить у нескінченний головний цикл. У цьому циклі безперервно викликається функція USBH_Process(), відповідальна за обробку стану USB Host та підключених пристроїв. Вона керує процесами підключення, енумерації та обміну даними з USB-накопичувачем. Паралельно відбувається перевірка

стану: чи USB-пристрій успішно розпізнано (енумеровано). Якщо енумерація пройшла успішно (що сигналізується прапорцем enum_done), цей прапорець скидається, і викликається функція play_directory для початку відтворення аудіофайлів з кореневого каталогу підключеного накопичувача. Якщо USB-пристрій ще не розпізнано або виникла помилка, цикл продовжується з обробки USB-подій.

Рисунок 3.1 – Блок-схема алгоритму роботи основної програми

Функція play_directory, алгоритм якої представлено на рисунку 3.2, відповідає за пошук та послідовне відтворення MP3-файлів у вказаному каталозі. На вході функція отримує шлях до каталогу та кількість файлів, які потрібно пропустити (seek). Спочатку відбувається ініціалізація локальних змінних, необхідних для роботи з файловою системою. Далі здійснюється спроба відкрити вказаний каталог за допомогою функції f_opendir.

Якщо каталог успішно відкрито, програма входить у цикл, де послідовно зчитуються елементи каталогу (f_readdir). Для кожного елемента перевіряється, чи не досягнуто кінця каталогу та чи не виникла помилка при читанні. Якщо все гаразд, ігноруються службові каталоги (такі як "." та ".."). Якщо поточний елемент є файлом, а не підкаталогом, то формується повний шлях до цього файлу та перевіряється його розширення на відповідність формату ".mp3".

У випадку, якщо знайдено MP3-файл, перевіряється, чи потрібно його пропустити відповідно до вхідного параметра seek. Якщо так, лічильник seek зменшується, і відбувається перехід до наступного елемента каталогу. Якщо ж файл пропускати не потрібно, викликається функція play_mp3 для його відтворення, передаючи їй повний шлях до файлу. Після обробки всіх елементів каталогу або у випадку помилки відкриття каталогу, функція повертає результат останньої операції з файловою системою.

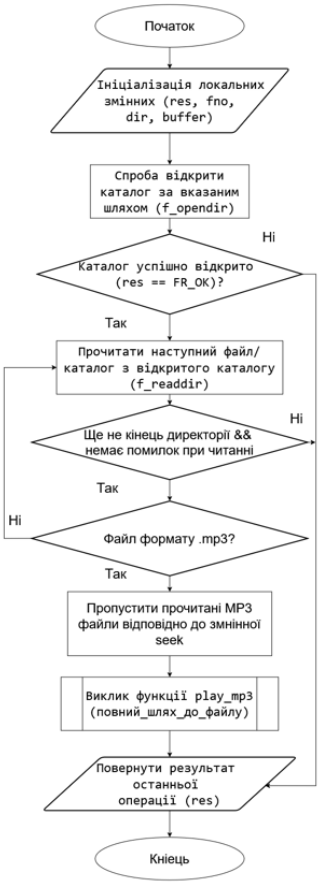

Рисунок 3.2 – Блок-схема алгоритму роботи функції play_directory

Функція play_mp3, алгоритм якої деталізовано на рисунку 3.3, забезпечує безпосереднє відтворення вказаного MP3-файлу. На початку відбувається ініціалізація вказівника читання (read_ptr) та лічильника доступних байт у буфері (bytes_left). Потім здійснюється спроба відкрити переданий MP3-файл для читання.

Якщо файл успішно відкрито, з нього зчитується ID3v2 тег (метадані треку) та заповнюється буфер читання (file_read_buffer) першою порцією даних з файлу. Після цього ініціалізується MP3-декодер, аудіодрайвер та встановлюється початковий рівень гучності. Далі запускається процес відтворення аудіо, вказуючи функцію AudioCallback як функцію зворотного виклику, яка буде постачати дані для відтворення.

Програма входить у нескінченний цикл відтворення та обробки. У цьому циклі постійно відбувається обробка отриманих пакетів команд від ESP32 (зміна гучності, оновлення коефіцієнтів еквалайзера) через виклик processUARTPacket. Також перевіряється, чи залишилося менше половини даних у буфері читання. Якщо так, то буфер дозаповнюється: залишок даних копіюється на початок буфера, і з файлу дочитується наступна порція даних. Після дозаповнення перевіряються умови для зупинки відтворення: досягнення кінця файлу, виникнення помилки читання або натискання користувачем кнопки зупинки. Якщо одна з цих умов виконується, відбувається зупинка відтворення аудіо, очищення ресурсів (реініціалізація аудіо, закриття файлу, очікування відпускання кнопки), і функція завершує свою роботу. Якщо умови зупинки не виконані, або якщо буфер ще достатньо заповнений, цикл продовжується. Якщо файл не вдалося відкрити на самому початку, функція також завершується.

Така організація функції play_mp3 дозволяє ефективно керувати потоком даних, забезпечувати безперервність відтворення завдяки механізму дозаповнення буфера та одночасно реагувати на зовнішні команди керування.

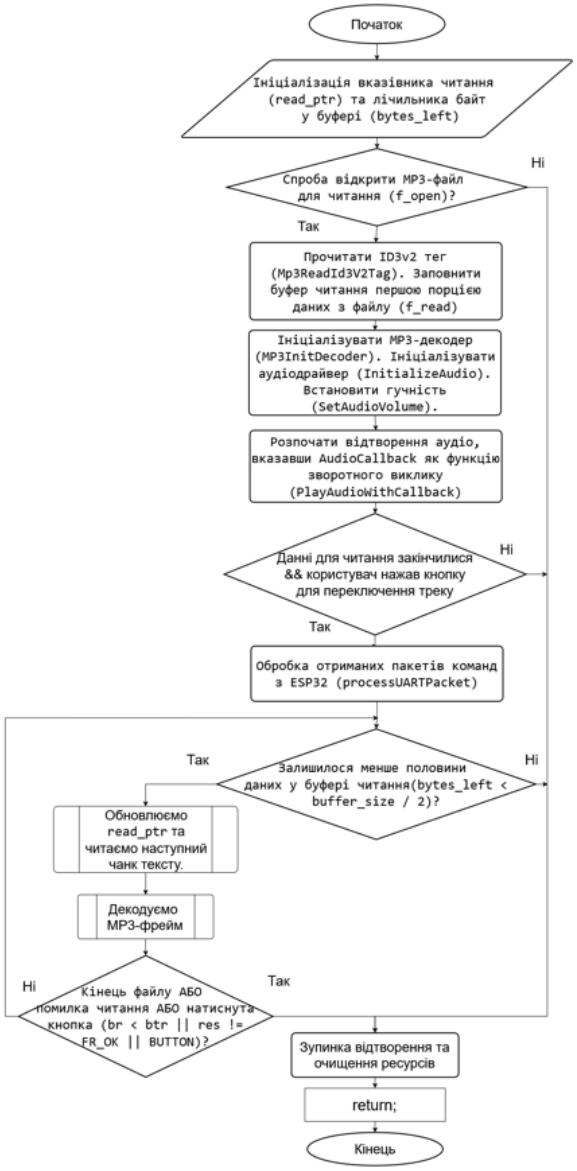

Рисунок 3.3 – Блок-схема алгоритму роботи функції play_mp3

Функція зворотного виклику AudioCallback, алгоритм якої наведено на рисунку 3.4, є серцем процесу безперервного відтворення аудіо. Вона викликається системою, коли необхідно надати наступний блок аудіоданих для передачі на аудіокодек.

На початку роботи функції, залежно від вхідного параметра, обирається один з двох доступних аудіобуферів (реалізація подвійної буферизації), який буде заповнюватися. Потім з основного буфера читання (file_read_buffer), на який вказує read_ptr, отримується відповідний набір аудіозразків та відбувається їх декодування з формату MP3 у PCM-формат за допомогою функції MP3Decode.

Після декодування здійснюється перевірка на можливі помилки або недостатнє заповнення буфера (наприклад, якщо досягнуто кінця файлу і даних для повного фрейму не вистачило). Якщо помилок немає і дані успішно декодовані, до отриманих PCM-семплів застосовуються відповідні біквадратичні фільтри для реалізації еквалайзера. На завершення, підготовлений та оброблений аудіобуфер надається програмі (драйверу аудіо) для передачі на аудіокодек через DMA. Якщо під час декодування виникла помилка або даних було недостатньо, етап фільтрації та передачі буфера може бути пропущений або замінений передачею буфера з тишею.

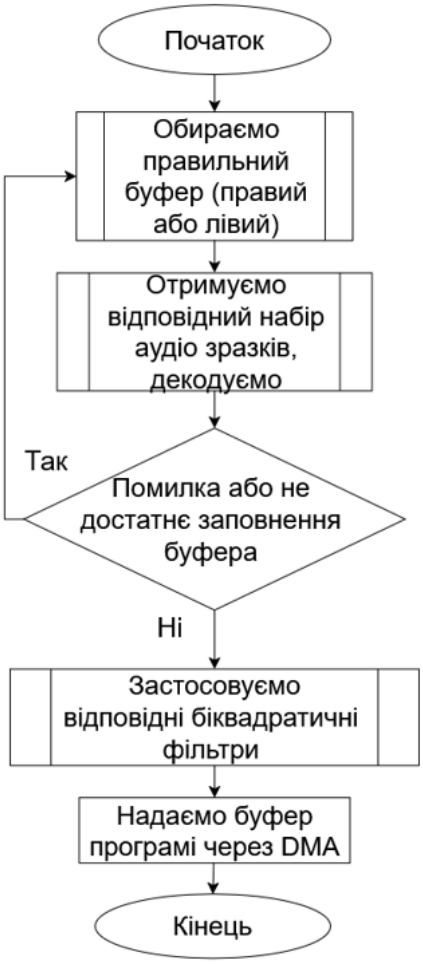

Рисунок 3.4 – Блок-схема алгоритму роботи функції AudioCallback

Така послідовність дій та взаємодія функцій забезпечує стабільне зчитування, декодування, обробку та відтворення аудіофайлів з можливістю динамічного керування параметрами звуку.

3.2 Організація переривань та обробка асинхронних подій на STM32

Програмне забезпечення, розроблене для мікроконтролера STM32, формує основу системи відтворення MP3-файлів з USB-накопичувачів та реалізації функціоналу веб-еквалайзера. Ключову роль у забезпеченні точного таймінгу, обробці асинхронних подій та ефективній роботі з аудіопотоком відіграють механізми переривань.

Налаштування системного таймера SysTick за допомогою функції SysTick_Config(RCC_Clocks.HCLK_Frequency / 1000) забезпечує генерацію переривань з періодом в 1 мілісекунду. Ці переривання використовуються для інкрементування глобальних змінних time_var1 та time_var2 в обробнику TimingDelay_Decrement(). Змінна time_var1 застосовується для реалізації програмних затримок через функцію Delay(), тоді як time_var2 може слугувати для вимірювання довших часових інтервалів або для організації періодичних завдань.

Ініціалізація стеку USB-хоста за допомогою USBH_Init() дозволяє мікроконтролеру STM32 працювати в режимі USB OTG (On-The-Go) та взаємодіяти з підключеними USB флеш-накопичувачами. У головному циклі програми, розташованому у функції main(), періодично викликається функція USBH_Process(). Ця функція відповідає за обробку всіх подій, пов'язаних з USB-інтерфейсом, таких як виявлення підключення або відключення пристрою, процес енумерації (визначення типу пристрою та його характеристик), а також управління передачею даних під час читання файлів.

Для передачі аудіоданих на аудіокодек CS43L22 після їх декодування та обробки активно використовується контролер прямого доступу до пам'яті (DMA). Це дозволяє пересилати блоки аудіосемплів з оперативної пам'яті безпосередньо до периферійного модуля I2S, минаючи центральний процесор. Такий підхід суттєво знижує навантаження на ядро Cortex-M4, звільняючи його для виконання інших обчислювально інтенсивних завдань, таких як

декодування MP3 та застосування фільтрів еквалайзера, і забезпечує плавне, безперервне відтворення аудіо.

Функція зворотного виклику AudioCallback(void *context, int buffer) відіграє центральну роль у конвеєрі обробки аудіо. Вона викликається системою тоді, коли аудіодрайвер потребує наступний блок даних для відтворення (зазвичай, коли DMA завершив передачу попереднього буфера). У цій функції відбувається програмне декодування чергового MP3-фрейму за допомогою бібліотеки mp3dec (функція MP3Decode). Декодовані аудіосемпли (у форматі int16_t) розміщуються в одному з двох аудіобуферів (audio_buffer0 або audio_buffer1), реалізуючи механізм подвійної буферизації.

3.3 Реалізація ключових алгоритмів обробки аудіо та даних на STM32

Програмне забезпечення мікроконтролера STM32, що є ядром аудіоплеєра, використовує механізми переривань для забезпечення точного таймінгу та реакції на асинхронні події. Розглянемо ключові аспекти реалізації та особливості програми з прикладами коду.

Для організації часових затримок та відліку інтервалів використовується системний таймер SysTick. Його ініціалізація відбувається на початку роботи програми. Обробник переривань SysTick, TimingDelay_Decrement, оновлює глобальні змінні, що використовуються для програмних затримок, що можна переглянути в лістингу 3.1.

```
volatile uint32_t time_var1, time_var2;
void TimingDelay_Decrement(void) {
  if (time_var1) {
        time_var1--;
  }
  time_var2++;
}
void Delay(volatile uint32_t nTime) {
  time_var1 = nTime;
  while(time_var1){};
}
```

Лістинг 3.1 – Код функцій для програмних затримок

Підтримка USB On-The-Go для читання файлів з флеш-накопичувачів ініціалізується на старті. У головному циклі програми main() постійно викликається USBH_Process() для обробки подій USB-шини відповідно до лістингу 3.2.

```
     for(;;) {
         USBH_Process(&USB_OTG_Core, &USB_Host);
         if (enum_done >= 2) { // Прапорець enum_done
встановлюється в USR_Callbacks при успішній енумерації
             enum_done = 0;
             play_directory("", 0); // Почати відтворення з
кореневого каталогу
         }
     }
```

Лістинг 3.2 – Код головного циклу програми

Центральною функцією для обробки аудіо є AudioCallback. Вона викликається, коли система готова прийняти наступний блок аудіоданих. У цій функції відбувається декодування MP3 та підготовка даних для DMA. Її частину можна переглянути в лістингу 3.3.

```
     static void AudioCallback(void *context, int buffer) {
       static int16_t audio_buffer0[2304];
       static int16_t audio_buffer1[2304];
       int offset, err;
       int outOfData = 0;
       int16_t *samples;
       if (buffer) {samples = audio_buffer0;
             GPIO_SetBits(GPIOD, GPIO_Pin_13);
             GPIO_ResetBits(GPIOD, GPIO_Pin_14);
       } else {samples = audio_buffer1;
             GPIO_SetBits(GPIOD, GPIO_Pin_14);
             GPIO_ResetBits(GPIOD, GPIO_Pin_13);    }
       offset = MP3FindSyncWord((unsigned char*)read_ptr,
bytes_left);
       bytes_left -= offset; read_ptr += offset;
       err = MP3Decode(hMP3Decoder, (unsigned char**)&read_ptr,
(int*)&bytes_left, samples, 0);
       if (err) {// Обробка помилок декодування (наприклад,
ERR_MP3_INDATA_UNDERFLOW)…}
       } else {MP3GetLastFrameInfo(hMP3Decoder, &mp3FrameInfo);}
       if (!outOfData) {ProvideAudioBuffer(samples,
mp3FrameInfo.outputSamps);}}
```

Лістинг 3.3 – Код функції AudioCallback

Обмін даними з ESP32 для отримання команд зміни гучності відбувається через USART2. Обробник переривань USART2_IRQHandler накопичує байти (див. лістинг 3.4) [14].

```
    void USART2_IRQHandler(void) {
        if (USART_GetITStatus(USART2, USART_IT_RXNE) == SET) {
            uint16_t data = USART_ReceiveData(USART2);
            uint8_t receivedByte = data & 0xFF;
            if (packetIndex < MAX_PACKET_SIZE) {
                uartPacketBuffer[packetIndex++] = receivedByte;
                if (packetIndex >= 3) {
                    if (uartPacketBuffer[0] != PACKET_HEADER1 ||
uartPacketBuffer[1] != PACKET_HEADER2) {
                        packetIndex = 0;
                        memset((void*)uartPacketBuffer, 0,
sizeof(uartPacketBuffer));
                        return;
                    }
                    uint8_t cmdID = uartPacketBuffer[2];
                    if (cmdID == CMD_VOLUME && packetIndex >=
VOL_PACKET_SIZE) {
                        packetComplete = 1;
                    } else if (cmdID == CMD_COEF && packetIndex
>= COEF_PACKET_SIZE) {
                        // Обробка команди коефіцієнтів також
тут, але деталі в розділі про фільтрацію
                        packetComplete = 1;
                    }
                }
            } else {
                packetIndex = 0;
                memset((void*)uartPacketBuffer, 0,
sizeof(uartPacketBuffer));
            }
        }
    }
```

Лістинг 3.4 – Код функції USART2_IRQHandler

У циклі відтворення play_mp3 викликається processUARTPacket для обробки зібраного пакету, функцію можна переглянути у лістингу 3.5. Також, після отримання пакету відбувається його зважування і перевірка на цілісність, що дозволяє налаштувати безпечну та безперервну комунікацію між двома ключовими мікроконтролерами системи.

```
    void processUARTPacket(void) {
        if (!packetComplete) return;
        uint8_t cmdID = uartPacketBuffer[2];
        if (cmdID == CMD_VOLUME && packetIndex ==
VOL_PACKET_SIZE) {
            uint8_t checksum =
calculateChecksum(uartPacketBuffer, VOL_PACKET_SIZE - 1);
            if (checksum == uartPacketBuffer[VOL_PACKET_SIZE -
1]) { // Перевірка контрольної суми
                uint8_t volume_raw = uartPacketBuffer[3]; //
Отримане значення 0-100
                int volume_codec = ((int)volume_raw) + 155; //
Мапування у діапазон кодека (приклад)
                SetAudioVolume(volume_codec); // Встановлення
гучності
            }
        }
        // Обробка CMD_COEF
        packetIndex = 0;
        packetComplete = 0;
        memset((void*)uartPacketBuffer, 0,
sizeof(uartPacketBuffer));
    }
```

Лістинг 3.5 – Код функції processUARTPacket

3.4 Реалізація аудіофільтрації на основі біквадратичних фільтрів та бібліотеки CMSIS-DSP

Однією з ключових функціональних можливостей розробленого аудіоплеєра є 10-смуговий графічний еквалайзер, який дозволяє користувачеві гнучко налаштовувати амплітудно-частотну характеристику аудіосигналу. Реалізація цього функціоналу покладена на мікроконтролер STM32F407VGT6 та здійснюється шляхом застосування каскаду цифрових біквадратичних фільтрів.

Для ефективної та оптимізованої реалізації цифрових фільтрів було обрано бібліотеку CMSIS-DSP (Cortex Microcontroller Software Interface Standard - Digital Signal Processing). Ця бібліотека, розроблена ARM, надає широкий набір функцій для цифрової обробки сигналів, оптимізованих для процесорів Cortex-M, включаючи Cortex-M4, що використовується в STM32F407VGT6.

Перед використанням кожен з десяти біквадратичних фільтрів (що відповідають частотним смугам 32Гц, 64Гц, 125Гц, 250Гц, 500Гц, 1кГц, 2кГц, 4кГц, 8кГц та 16кГц) повинен бути ініціалізований. Ініціалізація відбувається у функції main() один раз при запуску програми. Для кожного фільтра визначається екземпляр структури arm_biquad_casd_df1_inst_f32, що зберігає параметри фільтра, такі як кількість каскадів (секцій), вказівники на масиви коефіцієнтів та стану.

У даному проекті кожен фільтр є однією біквадратичною секцією (1 каскад). Також оголошено масив для зберігання стану фільтра (float filterXHz_state): для однієї біквадратичної секції у формі DF1 потрібно 4 елементи стану (по два для попередніх значень входу та виходу). Використовується масив для коефіцієнтів фільтра (float filterXHz_coefs): містить 5 коефіцієнтів (b0, b1, b2, a1, a2), де a0 приймається рівним 1. Початково ці коефіцієнти встановлюються так, щоб фільтр не змінював сигнал (пропускав його).

Застосування фільтрів до аудіосигналу відбувається всередині функції AudioCallback після того, як черговий MP3-фрейм декодовано і отримано масив аудіосемплів у форматі int16_t.

Декодовані семпли int16_t конвертуються у формат float32_t та записуються у проміжний буфер buf_in. Це необхідно, оскільки функція arm_biquad_cascade_df1_f32 працює з даними типу float32_t. Далі аудіодані з буфера buf_in послідовно пропускаються через кожен з десяти біквадратичних фільтрів. Вихід одного фільтра стає входом для наступного, таким чином реалізується каскадне включення фільтрів, що формують загальну АЧХ еквалайзера [15].

Після проходження через усі фільтри, оброблені дані float32_t з буфера buf_in конвертуються назад у формат int16_t для передачі на ЦАП. При цьому застосовується коефіцієнт масштабування (у даному коді 0.05f), який може слугувати для запобігання перевантаженню (кліпінгу) на виході або для загального регулювання рівня сигналу після фільтрації (див. лістинг 3.6).

```
     if (!outOfData) {for (int i = 0; i <
mp3FrameInfo.outputSamps && i < BLOCK_SIZE_FLOAT; i++)
{buf_in[i] = (float)samples[i];}
         uint32_t samplesToProcess = (mp3FrameInfo.outputSamps <
BLOCK_SIZE_FLOAT) ? mp3FrameInfo.outputSamps : BLOCK_SIZE_FLOAT;
         arm_biquad_cascade_df1_f32(&filter32Hz_settings,
&buf_in[0], &buf_in[0], samplesToProcess);
         arm_biquad_cascade_df1_f32(&filter64Hz_settings,
&buf_in[0], &buf_in[0], samplesToProcess);
         arm_biquad_cascade_df1_f32(&filter125Hz_settings,
&buf_in[0], &buf_in[0], samplesToProcess);
         arm_biquad_cascade_df1_f32(&filter250Hz_settings,
&buf_in[0], &buf_in[0], samplesToProcess);
         arm_biquad_cascade_df1_f32(&filter500Hz_settings,
&buf_in[0], &buf_in[0], samplesToProcess);
         arm_biquad_cascade_df1_f32(&filter1000Hz_settings,
&buf_in[0], &buf_in[0], samplesToProcess);
         arm_biquad_cascade_df1_f32(&filter2000Hz_settings,
&buf_in[0], &buf_in[0], samplesToProcess);
         arm_biquad_cascade_df1_f32(&filter4000Hz_settings,
&buf_in[0], &buf_in[0], samplesToProcess);
         arm_biquad_cascade_df1_f32(&filter8000Hz_settings,
&buf_in[0], &buf_in[0], samplesToProcess);
         arm_biquad_cascade_df1_f32(&filter16000Hz_settings,
&buf_in[0], &buf_in[0], samplesToProcess);
         for (int i = 0; i < samplesToProcess; i++) {
             float processed_sample = buf_in[i] * 0.05f;
             if (processed_sample > 32767.0f) processed_sample =
32767.0f; else if (processed_sample < -32768.0f)
processed_sample = -32768.0f; samples[i] =
(int16_t)processed_sample;} ProvideAudioBuffer(samples,
samplesToProcess);}
```

Лістинг 3.6 – Застосування каскаду біквадратичних фільтрів у AudioCallback

Коефіцієнти кожного з десяти біквадратичних фільтрів можуть динамічно змінюватися користувачем через веб-інтерфейс, розміщений на ESP32. Модуль ESP32 надсилає нові коефіцієнти на STM32 через USART2 у вигляді спеціальних пакетів. Структура пакету для оновлення коефіцієнтів (CMD_COEF) включає: двобайтовий заголовок (PACKET_HEADER1, PACKET_HEADER2), ідентифікатор команди (CMD_COEF), ідентифікатор фільтра (filterID від 0 до 9), п'ять коефіцієнтів фільтра типу float (кожен по 4 байти), однобайтова контрольна сума [17].

Обробник переривань USART2_IRQHandler (див. Лістинг 3.4)

накопичує байти. Коли виявлено потенційний пакет коефіцієнтів (CMD_COEF та достатня кількість байтів COEF_PACKET_SIZE), встановлюється прапорець packetComplete. Функція processUARTPacket (див. Лістинг 3.5) перевіряє цей прапорець. Якщо пакет CMD_COEF отримано та його контрольна сума вірна, з пакету витягується filterID та 5 коефіцієнтів. Ці дані передаються у функцію updateFilterCoefficients, яку можна переглянути в лістингу 3.7.

```
    void updateFilterCoefficients(uint8_t filterID, float
coef[5]) {
    // Вказівники на масиви коефіцієнтів для кожного фільтра
        float* filterCoeffsArray[10] = {
            filter32Hz_coefs,
            filter64Hz_coefs,
            filter125Hz_coefs,
            filter250Hz_coefs,
            filter500Hz_coefs,
            filter1000Hz_coefs,
            filter2000Hz_coefs,
            filter4000Hz_coefs,
            filter8000Hz_coefs,
            filter16000Hz_coefs
    };
        if (filterID < 10) {
            float* target = filterCoeffsArray[filterID];
            for (int i = 0; i < 5; i++) {
                // Invert the sign for feedback coefficients (i
> 2), keep the others as is.
                target[i] = (i > 2) ? -coef[i] : coef[i];
            }
        }
    }
```

Лістинг 3.7 – Код функції updateFilterCoefficients

3.5 Реалізація веб-інтерфейсу та керування еквалайзером на ESP32

Модуль ESP32-Wrover-Dev у комп’ютеризованій системі USB аудіоплеєра виконує ключову роль у забезпеченні користувацького інтерфейсу для керування параметрами звуку. Для цього на модулі було реалізовано Wi-Fi точку доступу, веб-сервер, що надає сторінку з елементами керування еквалайзером та гучністю, а також логіку для розрахунку

коефіцієнтів фільтрів та їх передачі на мікроконтролер STM32. Програмне забезпечення для ESP32 було розроблено в середовищі Arduino IDE з використанням відповідних бібліотек.

На початковому етапі роботи програми відбувається налаштування модуля ESP32 як точки доступу Wi-Fi. Це дозволяє користувачам підключатися до пристрою зі своїх смартфонів, планшетів або комп'ютерів без необхідності підключення до існуючої Wi-Fi мережі (див. лістинг 3.8) [18].

```
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
const char* ssid = "USB_AUDIO_PLAYER";
const char* password = "12345678"; // Пароль для точки
доступу
void setup() {
  // Стартуємо серіальний порт
  Serial.begin(UART_BAUD);
  // Встановлюємо ESP32 як точку доступу
  WiFi.softAP(ssid, password);
  // Виведення IP-адреси точки доступу в монітор порту для
налагодження
  Serial.print("AP IP address: ");
  Serial.println(WiFi.softAPIP());
  Serial.println();
}
```

Лістинг 3.8 – Налаштування точки доступу Wi-Fi

Для надання користувацького інтерфейсу було створено асинхронний веб-сервер на порту 80 за допомогою бібліотеки ESPAsyncWebServer. Ця бібліотека дозволяє ефективно обробляти HTTP-запити без блокування основного циклу програми.

При зверненні до кореневого шляху (/) сервер віддає HTML-сторінку index_html, яка містить елементи керування: один горизонтальний повзунок для загальної гучності та десять вертикальних повзунків для кожної з частотних смуг еквалайзера (32Гц, 64Гц, ..., 16кГц). JavaScript-код на цій сторінці відстежує зміни положення повзунків і при кожній зміні надсилає асинхронні GET-запити на відповідні ендпоінти сервера.

Сервер має два основні ендпоінти для обробки команд від веб-інтерфейсу: /setVolume, який приймає параметр value (0-100), що відповідає бажаному рівню гучності, а також /setEQ, що приймає параметри filter (ID фільтра, 0-9) та gain (підсилення в дБ, від -20 до +20), що відповідають налаштуванням конкретної смуги еквалайзера (див. лістинг 3.9).

```
     void setup() {
       // Ендпоінт для встановлення гучності
       server.on("/setVolume", HTTP_GET, [](AsyncWebServerRequest
*request) {
         if (request->hasParam("value")) {
           int vol = request->getParam("value")->value().toInt();
// Отримання значення гучності
           sendVolumePacket(vol); // Відправка пакету гучності на
STM32
           request->send(200, "text/plain", "Volume OK");
         } else {
           request->send(400, "text/plain", "Bad Request");
         }
       });
       // Ендпоінт для встановлення параметрів смуги еквалайзера
       server.on("/setEQ", HTTP_GET, [](AsyncWebServerRequest
*request) {
           // Розрахунок коефіцієнтів
           calcPeakingCoeffs(fc, Q, gain_dB, FS, coef); // FS -
частота дискретизації (44100.0f)
           // Відправка пакету коефіцієнтів на STM32
           sendCoefPacket((uint8_t)filterID, coef);
     }
```

Лістинг 3.9 – Обробники HTTP-запитів для гучності та еквалайзера

Коли надходить запит на зміну підсилення для певної смуги еквалайзера, ESP32 розраховує 5 коефіцієнтів для відповідного біквадратичного фільтра (див. лістинг 3.10). У наданому коді реалізовані функції для розрахунку коефіцієнтів трьох типів фільтрів: calcPeakingCoeffs (режекторний/піковий), calcLowShelfCoeffs (низькочастотний шельфовий) та calcHighShelfCoeffs (високочастотний шельфовий). Ці функції базуються на формулах з "Audio EQ Cookbook" Роберта Брістоу-Джонсона. Для поточного еквалайзера використовується calcPeakingCoeffs. Розраховані коефіцієнти відповідають коефіцієнтам b0, b1, b2, a1, a2 передавальної функції фільтра.

```
    void calcPeakingCoeffs(float fc, float Q, float gain_dB,
float fs, float coef[5]) {
      float A = pow(10.0f, gain_dB / 40.0f); // Перетворення дБ
в лінійне підсилення амплітуди
      float w0 = 2.0f * PI * fc / fs; // Нормалізована кутова
частота
      float cosw0 = cosf(w0);
      float sinw0 = sinf(w0);
      float alpha = sinw0 / (2.0f * Q); // Проміжний параметр
      // Формули з RBJ Cookbook
      float b0_rbj = 1.0f + alpha * A;
      float b1_rbj = -2.0f * cosw0;
      float b2_rbj = 1.0f - alpha * A;
      float a0_rbj = 1.0f + alpha / A;
      float a1_rbj = -2.0f * cosw0;
      float a2_rbj = 1.0f - alpha / A;
      // Нормалізація коефіцієнтів (ділення всіх на a0_rbj)
      coef[0] = b0_rbj / a0_rbj;  // b0' (в коді STM32 це b0)
      coef[1] = b1_rbj / a0_rbj;  // b1' (в коді STM32 це b1)
      coef[2] = b2_rbj / a0_rbj;  // b2' (в коді STM32 це b2)
      coef[3] = a1_rbj / a0_rbj;  // a1' (в коді STM32 це -a1)
      coef[4] = a2_rbj / a0_rbj;  // a2' (в коді STM32 це -a2)
    }
```

Лістинг 3.10 – Функція розрахунку коефіцієнтів для пікового фільтра

Після отримання команди на зміну гучності або розрахунку нових коефіцієнтів фільтра, ESP32 формує спеціальний пакет даних та відправляє його на STM32 через UART (див. лістинг 3.11 та 3.12).

```
    void sendVolumePacket(int volume) {
      uint8_t packet[VOLUME_PACKET_SIZE];
      packet[0] = HEADER_BYTE1;
      packet[1] = HEADER_BYTE2;
      packet[2] = CMD_VOLUME;
      packet[3] = (uint8_t)volume; // Гучність 0-100
      uint8_t checksum = 0;
      for (int i = 0; i < VOLUME_PACKET_SIZE - 1; i++) {
        checksum += packet[i];
      }
      packet[VOLUME_PACKET_SIZE - 1] = checksum; // Остання
назва індексу тут має бути packet[4]
      Serial.write(packet, VOLUME_PACKET_SIZE);
    }
```

Лістинг 3.11 – Функція відправки пакетів гучності

```
    // Відправка пакету коефіцієнтів
    void sendCoefPacket(uint8_t filterID, float coef[5]) {
      uint8_t packet[COEF_PACKET_SIZE];
      packet[0] = HEADER_BYTE1;
      packet[1] = HEADER_BYTE2;
      packet[2] = CMD_COEF;
      packet[3] = filterID;
      // Запакування 5 коефіцієнтів float (кожен по 4 байти)
      for (int i = 0; i < 5; i++) {
        union { float f; uint8_t b[4]; } data;
        data.f = coef[i];
        memcpy(packet + 4 + i*4, data.b, 4); // Копіювання
байтів float у пакет
      }
      uint8_t checksum = 0;
      for (int i = 0; i < COEF_PACKET_SIZE - 1; i++) {
        checksum += packet[i];
      }
      packet[COEF_PACKET_SIZE - 1] = checksum; // Остання назва
індексу тут має бути packet[24]
      Serial.write(packet, COEF_PACKET_SIZE);
    }
```

Лістинг 3.12 – Функція відправки пакетів з коефіцієнтами

Використання union для перетворення float у масив байтів є стандартним підходом для передачі даних з плаваючою комою через послідовні інтерфейси, де дані передаються побайтово. Контрольна сума додається для перевірки цілісності даних на стороні STM32.

Програмне забезпечення модуля ESP32 забезпечує зручний бездротовий інтерфейс для користувача, розраховує необхідні параметри для цифрової обробки звуку та передає їх на основний оброблювальний модуль STM32, реалізуючи замкнену систему керування аудіосигналом.

3.6 Компіляція, налагодження та результати тестування системи

Завершальним етапом розробки комп’ютеризованої системи USB аудіоплеєра з підтримкою веб-еквалайзера стало збирання (компіляція) програмного коду для обох мікроконтролерних модулів, їх програмування, налагодження взаємодії та комплексне тестування реалізованого функціоналу.

Програмне забезпечення для мікроконтролера STM32F407VGT6 було

розроблено та скомпільовано з використанням інтегрованого середовища розробки STM32CubeIDE. Після успішної компіляції, що включала перевірку синтаксису та зв'язування об'єктних файлів, отримана прошивка була завантажена на мікроконтролер. Програмування плати STM32F407G-DISC1 здійснювалося за допомогою вбудованого програматора/налагоджувача ST-Link, підключеного до персонального комп'ютера через USB-інтерфейс.

Для модуля ESP32-Wrover-Dev розробка та компіляція програмного коду виконувалася в середовищі Arduino IDE з попередньо встановленим пакетом підтримки для плат ESP32 (ESP32 Arduino Core). Процес компіляції також завершився успішно, після чого згенерований бінарний файл було завантажено на модуль ESP32 через USB-інтерфейс за допомогою вбудованих інструментів Arduino IDE.

Після програмування обох мікроконтролерів було проведено поетапне налагодження та тестування системи.

На першому етапі було перевірено коректність ініціалізації периферії STM32, роботу системного таймера SysTick, а також функціонал зчитування файлів з USB-накопичувача з файловою системою FAT32. Тестування показало успішне розпізнавання накопичувача та доступ до файлової системи. Світлодіодна індикація на платі використовувалася для візуального контролю за станами програми.

Було перевірено процес декодування MP3-файлів та виведення аудіосигналу через аудіокодек CS43L22 на підключені навушники. Відтворення звуку було стабільним, без явних спотворень на початковому етапі (без застосування еквалайзера). Функція AudioCallback коректно викликалася, забезпечуючи безперервну подачу даних на DMA.

Окремо було протестовано роботу модуля ESP32. Модуль успішно створював точку доступу Wi-Fi з визначеними SSID (USB_AUDIO_PLAYER) та паролем (12345678). Веб-сервер, розміщений на ESP32, коректно віддавав HTML-сторінку веб-еквалайзера при підключенні клієнтського пристрою (смартфона) до створеної Wi-Fi мережі та переході за IP-адресою модуля.

Наступним кроком було налагодження комунікації між STM32 та ESP32 через послідовний інтерфейс USART2. Перевірялася коректність передачі пакетів даних з командами зміни гучності та коефіцієнтами фільтрів від ESP32 до STM32. Для моніторингу переданих та отриманих даних використовувався монітор послідовного порту в Arduino IDE (для ESP32) та відповідні засоби налагодження в STM32CubeIDE (наприклад, виведення даних через SWV або зупинки на точках переривання для аналізу вмісту буферів). Було підтверджено, що STM32 коректно приймає та розпарсує пакети, перевіряючи заголовки та контрольні суми.

Здійснювалася перевірка реакції системи на дії користувача в веб-інтерфейсі. При зміні положення повзунків гучності та еквалайзера на веб-сторінці, ESP32 коректно розраховував нові параметри (рівень гучності або коефіцієнти фільтрів) та надсилав їх на STM32. На стороні STM32 ці дані приймалися, і було візуально (за допомогою налагоджувальної інформації) та на слух підтверджено, що рівень гучності змінюється відповідно до положення повзунка гучності, зміна положення повзунків еквалайзера призводить до зміни коефіцієнтів відповідних біквадратичних фільтрів та, як наслідок, до помітної зміни тембрального забарвлення відтворюваного аудіосигналу. Кожна з 10 смуг еквалайзера впливала на відповідний діапазон частот.

Система тестувалася на стабільність роботи протягом тривалого часу відтворення аудіо з одночасним активним використанням веб-еквалайзера. Збоїв у роботі, зависань або непередбачуваної поведінки під час тестових сесій виявлено не було [19].

За результатами компіляції, налагодження та тестування було підтверджено працездатність розробленої комп’ютеризованої системи USB аудіоплеєра з підтримкою веб-еквалайзера. Система продемонструвала успішне виконання всіх основних поставлених завдань:

– Зчитування та відтворення MP3-файлів з USB-накопичувача.

– Створення точки доступу Wi-Fi та хостинг веб-сервера з інтерфейсом керування.

– Динамічне керування рівнем гучності через веб-інтерфейс.

– Реалізація 10-смугового графічного еквалайзера з можливістю зміни підсилення кожної смуги через веб-інтерфейс.

– Стабільна взаємодія між мікроконтролером STM32F407VGT6 та модулем ESP32-Wrover-Dev для передачі команд керування.

Незначні труднощі, що виникали на етапі налагодження комунікації між модулями, були пов'язані з початковим узгодженням формату пакетів та швидкості передачі даних по USART, що було успішно вирішено шляхом ретельної перевірки програмного коду та налаштувань периферії на обох мікроконтролерах. Впровадження механізму контрольних сум для пакетів даних підвищило надійність комунікаційного каналу.

РОЗДІЛ 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Характеристика життєдіяльності людини у системі „людина - машина – середовище існування”

Взаємодія людини з технікою й середовищем стає дедалі інтенсивнішою і складнішою, формуючи багатовимірну систему, де відбувається постійна передача інформації й енергії між її компонентами. Людина, машина (техніка) і навколишнє середовище розглядаються в ергономіці як складові й невід’ємні елементи єдиної системи "людина – техніка – середовище".

У рамках системного підходу «людина – машина – середовище» ергономіка досліджує трудову діяльність людини у складі системи "людина – техніка – середовище" й приділяє особливу увагу функціональній структурі системи "людина – машина". Такий підхід дозволяє не лише описати механіку взаємодії, але й виявити потенційні вузькі місця, де можуть виникати фізіологічні чи когнітивні навантаження.

Людина є одним з елементів зазначеної системи, в якій під терміном “людина” розуміється не лише одна істота, індивід, а й група людей, колектив, мешканці населеного пункту, регіону, країни, суспільство [20].

У центрі «людина – машина – середовище» системи – операційна особа, яка сприймає інформацію через органи чуття, обробляє її в центральній нервовій системі й формує реакцію через дії. Саме цей процес охоплює фізіологічні, психологічні, когнітивні та соціальні аспекти життєдіяльності, що становлять основу безпеки й ефективності взаємодії. Інтерфейси й технічні засоби повинні бути ретельно спроектовані відповідно до фізичних можливостей людини – антропометричних, сенсорних, моторних, а також

відповідно до її здатності обробляти складну інформацію під час роботи чи побуту .

Принципи ергономіки, що охоплюють дизайн та модифікацію систем взаємодії, спрямовані на підвищення безпеки, комфорту і продуктивності. Ергономічні стандарти, закладені в ISO 9241 та інші нормативи, передбачають відповідність між можливостями людини та вимогами машинного інтерфейсу. Це гарантує, що структура, форма, розташування та інтуїтивність елементів управління відповідають потребам користувачів, зменшуючи ймовірність помилок чи травм .

Комплексні підходи до оцінки ризиків у таких системах дедалі частіше спираються на математичні моделі. Зокрема, застосування марковських процесів дозволяє моделювати динаміку небезпечних подій, враховуючи випадковість взаємодій людини з машиною і середовищем, що дає змогу передбачити критичні стани й упровадити превентивні заходи.

Механізм безпечної взаємодії включає врахування фізіологічних параметрів – зір, слух, моторика, втомлюваність, стрес. Сприймання звукових сигналів, зорових індикаторів, тактильного зворотного зв’язку повинно бути адекватно налаштоване: якщо індикатор надто яскравий або звук перевищує 80–85 дБА, це може викликати дискомфорт, втому або навіть слухову травму [21].

Під час розробки USB аудіо плеєра особливої уваги потребує ергономіка сприймання звукових сигналів. Пристрій відтворює аудіо в межах безпечного діапазону гучності, уникає високочастотних ультразвукових складових, а також забезпечує інтуїтивно зрозумілий інтерфейс управління. Надмірно гучні сигнали або погано продумана навігація спричиняють не лише дискомфорт, а й зростання когнітивного навантаження, що суперечить принципам безпечної взаємодії «людина – техніка – середовище».

Центральна нервова система відповідає не лише за миттєву реакцію, але й за адаптацію – здатність пристосовуватися до змін навантаження, фільтрувати інформаційний шум і зберігати пильність навіть в умовах

монотонної праці чи побуту. Зниження адаптаційних ресурсів, викликане дефіцитом сну, емоційним перевантаженням або шкідливими чинниками середовища, збільшує ризик помилок, нещасних випадків або професійних захворювань.

Крім фізіології, значущі психологічні аспекти: мотивація, увага, втома, рівень комфорту, взаємодія з іншими людьми. У високотехнологічному контексті, коли взаємодія з машинами відбувається через інтерфейси, кнопки, екрани, голосові помічники, виникає потреба враховувати когнітивну навантаженість і рівень довіри до системи. Помилки часто зумовлені не поганою технікою, а невідповідністю дизайну очікуванням користувача.

Необґрунтовано складний інтерфейс або незрозумілі інструкції можуть стати джерелом паніки у стресових умовах, особливо якщо система не передбачає чітких повідомлень про стан процесу або не підтримує оперативної допомоги. Сучасні стандарти HF/E закликають вбудовувати кольорові коди, звукові сигнали, контекстну інструкцію, все для підтримки когнітивної безпеки.

Безпеку не забезпечує лише технічний дизайн – важливими є організаційні рішення. Вони включають навчання користувачів, моніторинг параметрів середовища (шум, освітлення, температура), періодичні огляди систем, оцінку ризиків і впровадження протоколів реагування у випадку надзвичайних ситуацій. Адекватна організаційна структура здатна зменшувати ймовірність виникнення аварій і підвищувати здатність системи до самовідновлення.

При цьому принцип системності «людина–машина–середовище» передбачає, що ризики мають розглядатися в комплексі: змінюється середовище – змінюється робота пристрою, людина може досі нормально реагувати – але умови можуть зумовити зрив взаємодії.

Система «людина–машина–середовище» – це динамічна модель, у якій безпека виходить за межі технічних характеристик і розширюється до врахування когнітивної, фізіологічної та емоційної взаємодії. Безпечна

система – це така, що не лише мінімізує ризики, а й динамічно підтримує людину, спираючись на її потреби і стан у реальному часі.

4.2 Заходи щодо боротьби з шкідливою дією ультразвуку на організм людини

Ультразвук — це звукові хвилі високої частоти, що лежать за межами сприйняття людським слухом, але можуть викликати несприятливі фізіологічні ефекти навіть при непомітності для свідомості. У промислових умовах ультразвук застосовується в очищенні, дефектоскопії, зварюванні та інших технологічних процесах. Проте саме у таких середовищах проявляється і його потенційна небезпека: тривала дія ультразвуку може спричиняти головний біль, втому, порушення сну, розлади вестибулярного апарату, серцево судинної, нервової та ендокринної систем. Один із ключових профілактичних підходів — зниження інтенсивності джерела впливу, що називають першою лінією оборони проти ультразвукових ризиків [22].

Вплив високочастотних сигналів важливий параметр який врахований в системі USB аудіоплеєра. Під час дизайну плати і компонентів важливо забезпечити належне екранування, фільтрування та технічні бар’єри, щоб уникнути витоків ультразвукових завад. Це не лише запобігає появі шуму на виході, але й забезпечує безпечну експлуатацію пристрою — як для слуху, так і вібраційно-фізіологічно чутливих користувачів.

Для захисту організму від повітряного ультразвуку й шуму необхідне впровадження технічних бар’єрів. Застосовують звукоізолюючі кожухи, екрани, кабіни зі спеціальних звукопоглинальних матеріалів, обкладених гумою чи мастикою. Крім того, доцільним є розміщення устаткування в окремих звукоізольованих приміщеннях або зонування робочого простору. У критичних випадках використовують дистанційне управління — через пульт або сенсорні системи — щоб уникнути контакту людини з джерелом ультразвуку [23].

Контактний шлях передачі ультразвуку (через руки чи тіло) потребує окремих заходів. Найефективнішими вважаються рукавички зі спеціальним покриттям (гумові або двошарові) та інструменти з віброізолюючими ручками. У стандартах ДСН 3.3.6.037 99 передбачено граничні рівні, які не повинні перевищувати 0,1 Вт/см² по інтенсивності або $1{,}6\times10^{-2}$ м/с у віброшвидкості. Якщо ж неможливо уникнути контакту з джерелом, роботу слід виконувати в режимі повного припинення ультразвукової генерації або з використанням спеціальних інструментів.

Медичний контроль — ще один ключовий елемент безпеки. Працівники, які працюють з ультразвуковим обладнанням, мають проходити періодичні медичні огляди не рідше одного разу на рік, включно з обстеженням нервової, серцево-судинної, вестибулярної систем — згідно з наказом МОЗ №246 (2007). Своєчасне виявлення відхилень на ранніх стадіях дозволяє своєчасно скоригувати умови праці або перейти на інші обов’язки, що мають менше ультразвукове навантаження.

Застосувати фізіотерапевтичні заходи для відновлення працівників: масаж, УФ опромінення, водні процедури, позитивно впливає на тонус та регенерацію нервової системи. У сукупності ці методи — технічні, організаційні, санітарно-господарські та медичні — утворюють ефективну систему боротьби з ультразвуковим навантаженням [24].

Комплексний підхід до захисту від шкідливої дії ультразвуку — це поєднання технічних засобів (звукоізоляція, дистанційне керування), індивідуальних заходів (рукавички, навушники), організаційних обмежень (режим роботи, мінімальні вікові вимоги) та медичного контролю. Регулярне проведення інструментальної перевірки рівнів ультразвуку забезпечує дотримання встановлених меж захисту, а своєчасне обслуговування та оновлення обладнання попереджає виникнення небезпечних перевищень. Головна мета — знизити інтенсивність та тривалість впливу ультразвуку на організм, — що дозволяє безпечно застосовувати ультразвук в технічних і промислових умовах, зберігаючи здоров’я працівників.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено та протестовано комп’ютеризовану систему USB аудіоплеєра з веб-еквалайзером, також обґрунтовано вибір двокомпонентної архітектури: мікроконтролер STM32F407VGT6 для обробки та відтворення аудіо, та модуль ESP32-Wrover-Dev для реалізації мережевої взаємодії та веб-інтерфейсу, що дозволило ефективно розподілити обчислювальні завдання.

Програмне забезпечення для STM32F407VGT6, розроблене в STM32CubeIDE, забезпечує зчитування та декодування MP3-файлів з USB-накопичувача, застосування десятисмугового еквалайзера на базі біквадратичних фільтрів з використанням бібліотеки CMSIS-DSP, та виведення аудіосигналу через шину I2S з використанням DMA. Програмне забезпечення для ESP32-Wrover-Dev, створене в Arduino IDE, реалізує точку доступу Wi-Fi, асинхронний веб-сервер з інтерфейсом керування гучністю та еквалайзером, а також розрахунок коефіцієнтів фільтрів.

Для обміну даними між STM32 та ESP32 було реалізовано передачу даних на базі USART, що використовує пакети з заголовками, ідентифікаторами команд, даними та контрольною сумою для передачі налаштувань гучності та коефіцієнтів фільтрів.

Комплексне тестування підтвердило працездатність системи, включаючи відтворення MP3, керування гучністю та динамічну зміну АЧХ через веб-еквалайзер. Взаємодія компонентів стабільна, якість звуку та реакція системи відповідають проектним вимогам.

У результаті було створено завершений програмно-апаратний комплекс, що демонструє ефективне поєднання спеціалізованого мікроконтролера для аудіообробки та модуля з Wi-Fi для веб-інтерфейсу. Розроблений прототип має практичне значення для подальших розробок у сфері аудіотехніки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Жаровський Р.О., Луцик Н.С., Осухівська Г.М., Паламар А.М., Тиш Є.В. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для здобувачів першого (бакалаврського) рівня вищої освіти за спеціальністю 123 «Комп'ютерна інженерія» усіх форм навчання. Тернопіль: ТНТУ, 2024. 39 с.
2. Харченко О., Яцишин В. Розробка та керування вимогами до програмного забезпечення на основі моделі якості. Вісник ТДТУ. Тернопіль, 2009. Т. 14. №1. С. 201-207.
3. Kharchenko A., Bodnarchuk I., Yatcysyn V. The Method for Comparative Evaluation of Software Architecture with Accounting of Trade-offs. American Journal of Information Systems. 2014. Vol. 2, No. 1. P. 20-25.
4. STM32F407VGT6, STMicroelectronics Datasheet. URL: https://www.snapeda.com/parts/STM32F407VGT6/STMicroelectronics/datasheet/ (дата звернення: 02.03.2025).
5. ESP32-WROVER Datasheet. URL: https://cdn-shop.adafruit.com/product-files/3384/esp32-wrover_datasheet_en.pdf (дата звернення: 10.03.2025).
6. CS43L22 Datasheet. URL: https://datasheet.octopart.com/CS43L22-CNZ-Cirrus-Logic-datasheet-5397077.pdf (дата звернення: 12.03.2025).
7. Луцків А., Лупенко С., Пасічник В. Паралельні та розподільнені обчислення. Навчальний посібник. Львів: Видавництво «Магнолія 2006», 2024. 566 с.
8. STM32CubeIDE, Integrated Development Environment for STM32. URL: https://www.st.com/en/development-tools/stm32cubeide.html (дата звернення: 14.03.2025).
9. Arduino IDE 2.3.6. URL: https://www.arduino.cc/en/software/ (дата звернення: 16.03.2025).

10. Паламар М.І., Стрембіцький М.О., Паламар А.М. Проектування комп'ютеризованих вимірювальних систем і комплексів. Навчальний посібник. Тернопіль: ТНТУ. 2019. 150 с.

11. What is the I2S Communication Protocol? URL: https://www.digikey.com/en/maker/tutorials/2023/what-is-the-i2s-communication-protocol (дата звернення: 10.04.2025).

12. IT Master, I2C інтерфейс. URL: https://itmaster.biz.ua/directory/standarts/i2c.html (дата звернення: 20.04.2025).

13. Voloskyi V., Leshchyshyn Y., Romanyshyn N., Palamar A., Tarasenko L. Method and algorithm for efficient cell balancing in the lithium-ion battery control system. CEUR Workshop Proceedings, The 1st International Workshop on Bioinformatics and Applied Information Technologies (BAIT 2024), Zboriv, Ukraine, October 02-04, 2024. Vol. 3842. P. 258-267.

14. 12-UART-Library (STM32F4) URL: https://mikrocontroller.bplaced.net/wordpress/?page_id=223 (дата звернення: 10.05.2025).

15. Biquad Cascade IIR Filters Using a Direct Form II Transposed Structure. URL: https://arm-software.github.io/CMSIS_5/DSP/html/group__BiquadCascadeDF2T.html (дата звернення: 14.05.2025).

16. Max Cookbook, Biquad filter. URL: https://music.arts.uci.edu/dobrian/maxcookbook/keywords/biquad (дата звернення: 16.05.2025).

17. Biquad calculator. URL: https://www.earlevel.com/main/2021/09/02/biquad-calculator-v3/ (дата звернення: 19.05.2025).

18. Ліщина В., Жаровський Р. Методи підвищення пропускної здатності в мережах LTE. Матеріали X науково-технічної конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні

моделі системи та технології» (7-8 грудня 2022 року). Тернопіль: ТНТУ. 2022. С. 86.

19. Слюз І., Жаровський Р. Критерії ефективності тестування комп'ютерної інформаційної системи. Матеріали XI Міжнародна науково-технічна конференція молодих учених та студентів «Актуальні задачі сучасних технологій» (7-8 грудня 2022 року). Тернопіль: ТНТУ. 2022. С. 174

20. Перша допомога при кровотечах та ушкодженнях м'яких тканин. URL: https://dspace.uzhnu.edu.ua/jspui/bitstream/lib/22288/1/Безпека життєдіяльності (конспект лекцій).pdf (дата звернення: 28.05.2025).

21. Санітарні норми виробничого шуму, ультразвуку та інфразвуку ДСН 3.3.6.037-99. URL: https://zakon.rada.gov.ua/rada/show/va037282-99#Text (29.05.2025)

22. Медична енциклопедія. URL: https://medical-enc.com.ua/ultrazvuk.htm (дата звернення: 29.05.2025).

23. Шкідливий вплив ультразвуку на здоров'я працюючих та його профілактика. URL: https://oppb.com.ua/news/shkidlyvyy-vplyv-ultrazvuku-na-zdorovya-pracyuyuchyh-ta-yogo-profilaktyka (дата звернення: 30.05.2025).

24. Охорона праці. Якою може бути дія ультразвуку на організм людини? URL: https://ukrtextbook.com/oxorona-praci-moskalova-v-m/oxorona-praci-moskalova-v-m-yakoyu-mozhe-buti-diya-ultrazvuku-na-organizm-lyudini.html (дата звернення: 01.06.2025).

Додаток А

Технічне завдання

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії

Кафедра комп'ютерних систем та мереж

"Затверджую"

Завідувач кафедри КС

_____ Осухівська Г.М.

"___" _____2025 р

Комп'ютеризована система USB аудіоплеєра з підтримкою веб-еквалайзера

ТЕХНІЧНЕ ЗАВДАННЯ

на _8_ листках

Вид робіт: Кваліфікаційна робота

На здобуття освітнього ступеня «Бакалавр»

Спеціальність 123 «Комп'ютерна інженерія»

«УЗГОДЖЕНО»	«ВИКОНАВЕЦЬ»
Керівник кваліфікаційної роботи	Студент групи СІ-42
_____ канд. фіз-мат. наук Варавін А.В.	_____ Пуляк М.Я.
«____» _____ 2025 р.	«____» _____ 2025 р.

Тернопіль 2025

1 Загальні відомості

1.1 Повна назва та її умовне позначення

Повна назва теми кваліфікаційної роботи: «Комп’ютеризована система USB аудіоплеєра з підтримкою веб-еквалайзера».

Умовне позначення кваліфікаційної роботи: КС КРБ 123.232.00.00

1.2 Виконавець

Студент групи СІ-42, факультету комп’ютерно-інформаційних систем і програмної інженерії, кафедри комп’ютерних систем та мереж, Тернопільського національного технічного університету імені Івана Пулюя, Пуляк М.Я.

1.3 Підстава для виконання роботи

Підставою для виконання кваліфікаційної роботи є наказ по університету (№4/7-53 від 27.01.2025 р.)

1.4 Планові терміни початку та завершення роботи

Плановий термін початку виконання кваліфікаційної роботи – 27.01.2025 р.

Плановий термін завершення виконання кваліфікаційної роботи – 26.06.2025 р.

1.5 Порядок оформлення та пред’явлення результатів роботи

Порядок оформлення пояснювальної записки та графічного матеріалу здійснюється у відповідності до чинних норм та правил ISO, ЄСКД, ЄСПД та ДСТУ.

Пред’явлення проміжних результатів роботи з виконання кваліфікаційної роботи здійснюється у відповідності до графіку, затвердженого керівником роботи. Попередній захист кваліфікаційної роботи відбувається при готовності роботи – наявності пояснювальної записки та графічного матеріалу.

Пред’явлення результатів кваліфікаційної роботи відбувається шляхом захисту на відповідному засіданні ЕК, ілюстрацією основних досягнень за допомогою графічного матеріалу.

2 Призначення і цілі створення системи

2.1 Призначення системи

Розроблювана система призначена для високоякісного відтворення аудіофайлів з USB-накопичувачів з можливістю гнучкого налаштування користувачем амплітудно-частотної характеристики (АЧХ) звукового сигналу в реальному часі. Керування параметрами еквалайзера та загальною гучністю здійснюється через веб-інтерфейс, доступний за допомогою Wi-Fi з будь-якого пристрою з веб-браузером.

2.2 Мета створення системи

Мета роботи полягає у створенні надійного програмно-апаратного комплексу на базі мікроконтролера STM32F407VGT6 для обробки аудіо та модуля ESP32-Wrover-Dev для реалізації веб-сервера та користувацького інтерфейсу. Система повинна забезпечувати відтворення MP3-файлів, застосування цифрових біквадратичних фільтрів для еквалізації та передачу команд керування від веб-інтерфейсу на основний оброблювальний модуль.

2.3 Характеристика об'єкту

Об’єктом розробки є вбудована комп’ютеризована система, що поєднує функціонал аудіоплеєра та багатосмугового графічного еквалайзера. Система призначена для використання в якості автономного пристрою для відтворення та налаштування звуку, або як модуль для інтеграції в більші мультимедійні системи.

3 Вимоги до системи

3.1 Вимоги до системи в цілому

3.1.1 Вимоги до структури та функціонування системи

Система повинна мати двокомпонентну модульну структуру: мікроконтролер STM32F407VGT6 відповідає за зчитування аудіофайлів з USB, їх декодування, застосування ефектів еквалайзера на основі біквадратичних фільтрів та виведення аудіосигналу на аудіокодек CS43L22. Модуль ESP32-Wrover-Dev реалізує точку доступу Wi-Fi, веб-сервер з інтерфейсом еквалайзера, розраховує коефіцієнти фільтрів та передає команди керування на STM32.

3.1.2 Вимоги до способів та засобів зв’язку між компонентами системи

Модуль ESP32 передає команди керування (зміна гучності, коефіцієнти фільтрів) на мікроконтролер STM32 через послідовний інтерфейс USART (UART) у вигляді спеціально сформованих пакетів даних з контрольною сумою. STM32 взаємодіє з аудіокодеком CS43L22 через шину I2S для передачі аудіоданих та шину I2C для конфігурування. Зчитування з USB-накопичувача відбувається через USB OTG інтерфейс.

3.1.3 Вимоги до режимів функціонування системи

У нормальному режимі система очікує підключення USB-накопичувача, після чого починає відтворення аудіофайлів. Веб-інтерфейс еквалайзера доступний постійно після ініціалізації ESP32. Зміни налаштувань еквалайзера або гучності через веб-інтерфейс повинні застосовуватися до аудіосигналу в реальному часі. Передбачено можливість зупинки/продовження відтворення за допомогою фізичної кнопки.

3.1.4 Вимоги по діагностуванню системи

Базове діагностування може включати світлодіодну індикацію стану роботи ключових модулів (наприклад, активність USB, процес відтворення, отримання команд). Веб-інтерфейс може відображати поточні значення налаштувань еквалайзера та гучності. Можливе виведення налагоджувальної інформації через послідовний порт на етапі розробки.

3.1.5 Перспективи розвитку, проектування системи

Перспективи розвитку системи можуть включати підтримку інших аудіоформатів (наприклад, FLAC, WAV), розширення кількості смуг еквалайзера, додавання можливості збереження користувацьких пресетів еквалайзера, інтеграцію з іншими джерелами аудіо (наприклад, Bluetooth-аудіо), а також розробку більш досконалого графічного інтерфейсу.

3.2 Показники призначення

Система розрахована на відтворення MP3-файлів з USB-накопичувачів. Веб-еквалайзер повинен забезпечувати керування щонайменше 10 частотними смугами. Затримка застосування змін, внесених через веб-інтерфейс, до аудіосигналу не повинна перевищувати 0.5 секунди. Система повинна забезпечувати стабільне відтворення аудіо з частотою дискретизації 44.1 кГц.

3.2.1 Вимоги до надійності

Система повинна забезпечувати стабільне відтворення аудіо без збоїв та зависань протягом тривалого часу. Програмне забезпечення має бути стійким до можливих помилок при читанні файлів або передачі даних. Передбачено відновлення роботи після перепідключення USB-накопичувача.

3.3 Вимоги до безпеки

Пристрій повинен бути спроектований з урахуванням електробезпеки. Використання стандартних модулів розробки (STM32 Discovery, ESP32-Wrover-Dev) та зовнішнього живлення (USB, батареї) передбачає дотримання правил безпечної експлуатації цих компонентів. Корпус пристрою (якщо передбачений) повинен запобігати випадковому дотику до струмопровідних частин.

3.3.1 Вимоги до експлуатації, технічного обслуговування, ремонту і зберігання компонентів системи

Компоненти системи повинні експлуатуватися в умовах, що відповідають їх технічним специфікаціям (температура, вологість). Технічне обслуговування може включати оновлення прошивки. Ремонт передбачає заміну несправних модулів. Зберігання компонентів повинно здійснюватися в сухих умовах.

3.4 Вимоги до захисту інформації від несанкціонованого доступу

Для Wi-Fi точки доступу, створюваної ESP32, повинен бути встановлений пароль для запобігання несанкціонованого підключення та керування пристроєм. Передача команд між ESP32 та STM32 відбувається по дротовому з'єднанню USART, що обмежує можливість зовнішнього перехоплення в межах пристрою.

3.4.1 Вимоги по збереженню інформації при аваріях

Поточні налаштування еквалайзера та гучності зберігаються в оперативній пам'яті ESP32 та STM32 і не зберігаються при вимкненні живлення. Система не передбачає зберігання критично важливих даних, втрата яких може призвести до значних наслідків.

3.4.2 Вимоги по стандартизації і уніфікації

Використовуються стандартні інтерфейси (USB, I2S, I2C, USART, Wi-Fi) та протоколи (HTTP, TCP/IP). Програмне забезпечення розробляється з використанням поширених середовищ розробки (STM32CubeIDE, Arduino IDE) та стандартних бібліотек (CMSIS-DSP, FatFs, ESPAsyncWebServer).

3.4.3 Вимоги до функцій (завдань), що виконуються системою:

Система повинна забезпечувати: зчитування та декодування MP3-файлів; відтворення аудіо через аудіокодек; 10-смугову еквалізацію звуку; керування гучністю; веб-інтерфейс для налаштування еквалайзера та гучності; передачу команд керування між ESP32 та STM32. Затримка відгуку веб-інтерфейсу на дії користувача не повинна бути значною.

4 Вимоги до документації

Документація повинна відповідати вимогам ЄСКД та ДСТУ

Комплект документації повинен складатись з:

– пояснювальної записки;
– графічного матеріалу:
  а) Структурна схема.
  б) Блок схема роботи комп’ютеризованої системи.
  в) Схема електрична принципова.
  г) Схема електрична монтажна (з’єднань).

*Примітка: У комплект документації можуть вноситися зміни та доповнення в процесі розробки.

5 Стадії та етапи проектування

Таблиця 1 – Стадії та етапи виконання кваліфікаційної роботи бакалавра

№ етапу	Назва етапу виконання кваліфікаційної роботи	Термін виконання
1	Розробка технічного завдання	27.01.2025р. – 02.02.2025р.
2	Аналіз технічного завдання usb аудіоплеєра	12.02.2025р. – 25.03.2025р.
3	Проєктування системи usb аудіоплеєра	06.04.2025р. – 03.05.2025р.
4	Програмна реалізація та тестування комп'ютеризованої системи	08.05.2025р. – 26.05.2025р.
5	Безпека життєдіяльності, основи охорони праці	27.05.2025р. – 02.06.2025р.
6	Оформлення пояснювальної записки і графічного матеріалу	03.06.2025р. – 08.06.2025р.
7	Перевірка на академічний плагіат, перевірка керівником та консультантами	9.06.2025р. – 15.06.2025р.
8	Попередній захист кваліфікаційної роботи бакалавра	16.06.2025р. – 22.06.2025р.
9	Захист кваліфікаційної роботи бакалавра	26.06.2025р.

6 Додаткові умови виконання кваліфікаційної роботи

Під час виконання кваліфікаційної роботи у дане технічне завдання можуть вноситися зміни та доповнення.

Додаток Б

Лістинг коду програми для STM32

```
#define ARM_MATH_CM4

#include "main.h"
#include "core_cm4.h"
#include "stm32f4xx_conf.h"
#include "mp3dec.h"
#include "Audio.h"
#include <string.h>
#include <stdlib.h>
#include "stm32f4xx.h"          // Core header for STM32F4
#include "stm32f4xx_usart.h"    // USART functions and definitions
#include "stm32f4xx_gpio.h"     // GPIO configuration
#include "stm32f4xx_rcc.h"      // Clock configuration
#include "misc.h"               // NVIC configuration
#include "stm32_ub_uart.h"      // UART lib
#include "arm_math.h"
#include "arm_stereo.h"

// Macros
#define f_tell(fp)       ((fp)->fptr)
#define BUTTON           (GPIOA->IDR & GPIO_Pin_0)

// Variables
volatile uint32_t        time_var1, time_var2;
USB_OTG_CORE_HANDLE      USB_OTG_Core;
USBH_HOST                USB_Host;
RCC_ClocksTypeDef        RCC_Clocks;
volatile int             enum_done = 0;

// MP3 Variables
#define FILE_READ_BUFFER_SIZE 8192
MP3FrameInfo             mp3FrameInfo;
HMP3Decoder                   hMP3Decoder;
FIL                           file;
char                     file_read_buffer[FILE_READ_BUFFER_SIZE];
volatile int             bytes_left;
char                     *read_ptr;

// Variables for UART command reception
UART_RXSTATUS_t  check;
UART_RX_t UART_RX[UART_ANZ];
volatile uint16_t uartCmdIndex = 0;
volatile uint8_t uartCmdComplete = 0;

// Define packet sizes and header values
#define VOL_PACKET_SIZE    5
#define COEF_PACKET_SIZE   25
```

```
#define PACKET_HEADER1     0xAA
#define PACKET_HEADER2     0x55

// Command IDs
#define CMD_VOLUME 0x01
#define CMD_COEF   0x02

// Global buffer for incoming packet
#define MAX_PACKET_SIZE  25  // Maximum packet size we expect
volatile uint8_t uartPacketBuffer[MAX_PACKET_SIZE * 3];
volatile uint8_t packetIndex = 0;
volatile uint8_t packetComplete = 0;

// Function prototypes
void processUARTPacket(void);
void updateFilterCoefficients(uint8_t filterID, float coef[5]);

// Biquad filter
#define BLOCK_SIZE_FLOAT 2304

arm_biquad_casd_df1_inst_f32 filter32Hz_settings;
arm_biquad_casd_df1_inst_f32 filter64Hz_settings;
arm_biquad_casd_df1_inst_f32 filter125Hz_settings;
arm_biquad_casd_df1_inst_f32 filter250Hz_settings;
arm_biquad_casd_df1_inst_f32 filter500Hz_settings;
arm_biquad_casd_df1_inst_f32 filter1000Hz_settings;
arm_biquad_casd_df1_inst_f32 filter2000Hz_settings;
arm_biquad_casd_df1_inst_f32 filter4000Hz_settings;
arm_biquad_casd_df1_inst_f32 filter8000Hz_settings;
arm_biquad_casd_df1_inst_f32 filter16000Hz_settings;

float filter32Hz_state [4];
float filter64Hz_state [4];
float filter125Hz_state [4];
float filter250Hz_state [4];
float filter500Hz_state [4];
float filter1000Hz_state [4];
float filter2000Hz_state [4];
float filter4000Hz_state [4];
float filter8000Hz_state [4];
float filter16000Hz_state [4];

float filter32Hz_coefs [5] = {
          1.0f,
          0.0f,
          0.0f,
          0.0f,
          0.0f
};
float filter64Hz_coefs [5] = {
          1.0f,
          0.0f,
          0.0f,
```

```
        0.0f,
        0.0f
};
float filter125Hz_coefs [5] = {
        1.0f,
        0.0f,
        0.0f,
        0.0f,
        0.0f
};
float filter250Hz_coefs [5] = {
        1.0f,
        0.0f,
        0.0f,
        0.0f,
        0.0f
};
float filter500Hz_coefs [5] = {
        1.0f,
        0.0f,
        0.0f,
        0.0f,
        0.0f
};
float filter1000Hz_coefs [5] = {
        1.0f,
        0.0f,
        0.0f,
        0.0f,
        0.0f
};
float filter2000Hz_coefs [5] = {
        1.0f,
        0.0f,
        0.0f,
        0.0f,
        0.0f
};
float filter4000Hz_coefs [5] = {
        1.0f,
        0.0f,
        0.0f,
        0.0f,
        0.0f
};
float filter8000Hz_coefs [5] = {
        1.0f,
        0.0f,
        0.0f,
        0.0f,
        0.0f
};
float filter16000Hz_coefs [5] = {
```

```
          1.0f,
          0.0f,
          0.0f,
          0.0f,
          0.0f
};

float buf_in [BLOCK_SIZE_FLOAT]; // 2304

// Private function prototypes
static void AudioCallback(void *context,int buffer);
static uint32_t Mp3ReadId3V2Tag(FIL* pInFile, char* pszArtist,
         uint32_t   unArtistSize,   char*   pszTitle,   uint32_t
unTitleSize);
static void play_mp3(char* filename);
static FRESULT play_directory (const char* path, unsigned char
seek);

void USART2_IRQHandler(void) {
    if (USART_GetITStatus(USART2, USART_IT_RXNE) == SET) {
        // Read received data (only lower 8 bits are valid)
        uint16_t data = USART_ReceiveData(USART2);
        uint8_t receivedByte = data & 0xFF;

        // Accumulate byte in the packet buffer if there's room
        if (packetIndex < MAX_PACKET_SIZE) {
            uartPacketBuffer[packetIndex++] = receivedByte;

            // If we detect a complete packet based on the expected
size,
            // you might choose to check the command ID to decide
the expected length.
            // For simplicity, let's say we check header first:
            if (packetIndex >= 3) {
                // Check header to determine packet type
                if  (uartPacketBuffer[0]  !=  PACKET_HEADER1  ||
uartPacketBuffer[1] != PACKET_HEADER2) {
                    // Invalid header, reset buffer
                    packetIndex = 0;
                    memset((void*)uartPacketBuffer,           0,
MAX_PACKET_SIZE);
                    return;
                }
                // Now, use command ID (byte 2) to decide expected
packet size
                if   (uartPacketBuffer[2]   ==   CMD_VOLUME   &&
packetIndex >= VOL_PACKET_SIZE) {
                    packetComplete = 1;
                } else if (uartPacketBuffer[2] == CMD_COEF &&
packetIndex >= COEF_PACKET_SIZE) {
                    packetComplete = 1;
                }
            }
```

```
        } else {
            // Buffer overflow: reset
            packetIndex = 0;
            memset((void*)uartPacketBuffer, 0, MAX_PACKET_SIZE);
        }
    }
}

// Function to compute checksum: simple sum modulo 256
uint8_t calculateChecksum(const uint8_t *data, uint8_t len) {
    uint16_t sum = 0;
    for (uint8_t i = 0; i < len; i++) {
        sum += data[i];
    }
    return (uint8_t)(sum & 0xFF);
}

// Process a complete packet
void processUARTPacket(void) {
    if (!packetComplete)
        return;

    // Determine packet type based on command ID (byte 2)
    uint8_t cmdID = uartPacketBuffer[2];
    if (cmdID == CMD_VOLUME && packetIndex == VOL_PACKET_SIZE) {
        // Verify checksum for volume packet
        uint8_t checksum = calculateChecksum(uartPacketBuffer,
VOL_PACKET_SIZE - 1);
        if (checksum == uartPacketBuffer[VOL_PACKET_SIZE - 1]) {
            uint8_t volume = uartPacketBuffer[3];
            int volume16_t = ((int)volume) + 155;
            SetAudioVolume(volume16_t);
        }
    }
    else if (cmdID == CMD_COEF && packetIndex == COEF_PACKET_SIZE)
{
        // Verify checksum for coefficient packet
        uint8_t checksum = calculateChecksum(uartPacketBuffer,
COEF_PACKET_SIZE - 1);
        if (checksum == uartPacketBuffer[COEF_PACKET_SIZE - 1]) {
            uint8_t filterID = uartPacketBuffer[3];
            float coef[5];
            for (int i = 0; i < 5; i++) {
                union {
                    float f;
                    uint8_t bytes[4];
                } converter;
                memcpy(converter.bytes, &uartPacketBuffer[4 + i *
4], 4);
                coef[i] = converter.f;
            }
            // Call a function to update the corresponding filter
with these coefficients
```

```
            updateFilterCoefficients(filterID, coef);
        }
    }

    // Reset buffer after processing
    packetIndex = 0;
    packetComplete = 0;
    memset((void*)uartPacketBuffer, 0, MAX_PACKET_SIZE);
}

// Example function: update filter coefficients for a given filter
ID
void updateFilterCoefficients(uint8_t filterID, float coef[5]) {
    // Array of pointers to the coefficient arrays for each
frequency band.
    float* filterCoeffsArray[10] = {
        filter32Hz_coefs,
        filter64Hz_coefs,
        filter125Hz_coefs,
          filter250Hz_coefs,
        filter500Hz_coefs,
        filter1000Hz_coefs,
        filter2000Hz_coefs,
          filter4000Hz_coefs,
          filter8000Hz_coefs,
          filter16000Hz_coefs
    };

    // Ensure filterID is within range.
    if (filterID < 10) {
        float* target = filterCoeffsArray[filterID];
        for (int i = 0; i < 5; i++) {
            // Invert the sign for feedback coefficients (i > 2),
keep the others as is.
            target[i] = (i > 2) ? -coef[i] : coef[i];
        }
    }
}

void SystemClock_Config(void)
{
    // Step 1: Reset RCC settings to default
    RCC_DeInit();

    // Step 2: Enable High-Speed External (HSE) oscillator
    RCC_HSEConfig(RCC_HSE_ON);

    // Wait for HSE to stabilize
    if (RCC_WaitForHSEStartUp() == SUCCESS)
    {
        // Step 3: Configure the PLL
        RCC_PLLConfig(RCC_PLLSource_HSE, 8, 336, 2, 7);
```

```
        // Step 4: Enable the PLL
        RCC_PLLCmd(ENABLE);

        // Wait for PLL to lock
        while (RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);

        // Step 5: Set system clock source to PLL
        RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);

        // Step 6: Configure the AHB, APB1, and APB2 prescalers
        RCC_HCLKConfig(RCC_SYSCLK_Div1);      // AHB = 168 MHz
        RCC_PCLK1Config(RCC_HCLK_Div4);        // APB1 = 42 MHz
        RCC_PCLK2Config(RCC_HCLK_Div2);        // APB2 = 84 MHz

        // Step 7: Enable the Flash prefetch buffer and set latency
        FLASH_SetLatency(FLASH_Latency_5);
        FLASH_PrefetchBufferCmd(ENABLE);
    }
}

/*
 * Main function. Called when startup code is done with
 * copying memory and setting up clocks.
 */
int main(void) {
     SystemClock_Config();

     // Intitialize the filters settings
     arm_biquad_cascade_df1_init_f32(&filter32Hz_settings,     1,
&filter32Hz_coefs[0], &filter32Hz_state[0]);
     arm_biquad_cascade_df1_init_f32(&filter64Hz_settings,     1,
&filter64Hz_coefs[0], &filter64Hz_state[0]);
     arm_biquad_cascade_df1_init_f32(&filter125Hz_settings,    1,
&filter125Hz_coefs[0], &filter125Hz_state[0]);
     arm_biquad_cascade_df1_init_f32(&filter250Hz_settings,    1,
&filter250Hz_coefs[0], &filter250Hz_state[0]);
     arm_biquad_cascade_df1_init_f32(&filter500Hz_settings,    1,
&filter500Hz_coefs[0], &filter500Hz_state[0]);
     arm_biquad_cascade_df1_init_f32(&filter1000Hz_settings,   1,
&filter1000Hz_coefs[0], &filter1000Hz_state[0]);
     arm_biquad_cascade_df1_init_f32(&filter2000Hz_settings,   1,
&filter2000Hz_coefs[0], &filter2000Hz_state[0]);
     arm_biquad_cascade_df1_init_f32(&filter4000Hz_settings,   1,
&filter4000Hz_coefs[0], &filter4000Hz_state[0]);
     arm_biquad_cascade_df1_init_f32(&filter8000Hz_settings,   1,
&filter8000Hz_coefs[0], &filter8000Hz_state[0]);
     arm_biquad_cascade_df1_init_f32(&filter16000Hz_settings,  1,
&filter16000Hz_coefs[0], &filter16000Hz_state[0]);

     GPIO_InitTypeDef  GPIO_InitStructure;
     // SysTick interrupt each 1ms
     RCC_GetClocksFreq(&RCC_Clocks);
     SysTick_Config(RCC_Clocks.HCLK_Frequency / 1000);
```

```
     // GPIOD Peripheral clock enable
     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);

     // Configure PD12, PD13, PD14 and PD15 in output pushpull
mode
     GPIO_InitStructure.GPIO_Pin  =  GPIO_Pin_12  |  GPIO_Pin_13|
GPIO_Pin_14| GPIO_Pin_15;
     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
     GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
     GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
     GPIO_Init(GPIOD, &GPIO_InitStructure);

     // Initialize USB Host Library
     USBH_Init(&USB_OTG_Core,    USB_OTG_FS_CORE_ID,    &USB_Host,
&USBH_MSC_cb, &USR_Callbacks);

     //  Configure  USART2  using  standard  peripheral  library
functions
     UB_Uart_Init();

     for(;;) {
          USBH_Process(&USB_OTG_Core, &USB_Host);

          if (enum_done >= 2) {
               enum_done = 0;
               play_directory("", 0);
          }
     }
}

const char *get_filename_ext(const char *filename) {
    const char *dot = strrchr(filename, '.');
    if(!dot || dot == filename) return "";
    return dot + 1;
}

static FRESULT play_directory (const char* path, unsigned char
seek) {
     FRESULT res;
     FILINFO fno;
     DIR dir;
     char *fn; /* This function is assuming non-Unicode cfg. */
     char buffer[200];
#if _USE_LFN
     static char lfn[_MAX_LFN + 1];
     fno.lfname = lfn;
     fno.lfsize = sizeof(lfn);
#endif


     res = f_opendir(&dir, path); /* Open the directory */
     if (res == FR_OK) {
```

```
          for (;;) {
               res = f_readdir(&dir, &fno); /* Read a directory
item */
               if (res != FR_OK || fno.fname[0] == 0) break; /*
Break on error or end of dir */
               if (fno.fname[0] == '.') continue; /* Ignore dot
entry */
#if _USE_LFN
               fn = *fno.lfname ? fno.lfname : fno.fname;
#else
               fn = fno.fname;
#endif
               if (fno.fattrib & AM_DIR) { /* It is a directory */

               } else { /* It is a file. */
                    sprintf(buffer, "%s/%s", path, fn);

                    // Check if it is an mp3 file
                    if (strcmp("mp3", get_filename_ext(buffer)) ==
0) {

                         // Skip "seek" number of mp3 files...
                         if (seek) {
                              seek--;
                              continue;
                         }

                         play_mp3(buffer);
                    }
               }
          }
     }

     return res;
}

static void play_mp3(char* filename) {
     unsigned int br, btr;
     FRESULT res;

     bytes_left = FILE_READ_BUFFER_SIZE;
     read_ptr = file_read_buffer;

     if (FR_OK == f_open(&file, filename, FA_OPEN_EXISTING |
FA_READ)) {

          // Read ID3v2 Tag
          char szArtist[120];
          char szTitle[120];
          Mp3ReadId3V2Tag(&file,   szArtist,   sizeof(szArtist),
szTitle, sizeof(szTitle));

          // Fill buffer
```

```
          f_read(&file, file_read_buffer, FILE_READ_BUFFER_SIZE,
&br);

          // Play mp3
          hMP3Decoder = MP3InitDecoder();
          Delay(200);
          InitializeAudio(Audio44100HzSettings);

          SetAudioVolume(0xAF);
          PlayAudioWithCallback(AudioCallback, 0);
          SetAudioVolume(205);

          for(;;) {
               /*
                *
                * If past half of buffer, refill...
                *
                * When bytes_left changes, the audio callback has
just been executed. This
                * means that there should be enough time to copy
the end of the buffer
                * to the beginning and update the pointer before
the next audio callback.
                * Getting audio callbacks while the next part of
the file is read from the
                * file system should not cause problems.
                */

               // Process received UART commands asynchronously
               processUARTPacket();

               if (bytes_left < (FILE_READ_BUFFER_SIZE / 2)) {
                    // Copy rest of data to beginning of read
buffer
                    memcpy(file_read_buffer,            read_ptr,
bytes_left);

                    // Update read pointer for audio sampling
                    read_ptr = file_read_buffer;

                    // Read next part of file
                    btr = FILE_READ_BUFFER_SIZE - bytes_left;
                    res   =   f_read(&file,   file_read_buffer   +
bytes_left, btr, &br);

                    // Update the bytes left variable
                    bytes_left = FILE_READ_BUFFER_SIZE;

                    // Out of data or error or user button... Stop
playback!
                    if (br < btr || res != FR_OK || BUTTON) {
                         StopAudio();
```

```
                        // Re-initialize and set volume to avoid
noise
                        InitializeAudio(Audio44100HzSettings);
                        SetAudioVolume(0);

                        // Close currently open file
                        f_close(&file);

                        // Wait for user button release
                        while(BUTTON){};

                        // Return to previous function
                        return;
                    }
                }
            }
        }
}

/*
 * Called by the audio driver when it is time to provide data to
 * one of the audio buffers (while the other buffer is sent to the
 * CODEC using DMA). One mp3 frame is decoded at a time and
 * provided to the audio driver.
 */
static void AudioCallback(void *context, int buffer) {
    static int16_t audio_buffer0[2304];
    static int16_t audio_buffer1[2304];
// 4096
    int offset, err;
    int outOfData = 0;

    int16_t *samples;
    if (buffer) {
        samples = audio_buffer0;
        GPIO_SetBits(GPIOD, GPIO_Pin_13);
        GPIO_ResetBits(GPIOD, GPIO_Pin_14);
    } else {
        samples = audio_buffer1;
        GPIO_SetBits(GPIOD, GPIO_Pin_14);
        GPIO_ResetBits(GPIOD, GPIO_Pin_13);
    }

    offset    =    MP3FindSyncWord((unsigned    char*)read_ptr,
bytes_left);
    bytes_left -= offset;
    read_ptr += offset;

    err  =  MP3Decode(hMP3Decoder,  (unsigned  char**)&read_ptr,
(int*)&bytes_left, samples, 0);

    if (err) {
        /* error occurred */
```

```
          switch (err) {
          case ERR_MP3_INDATA_UNDERFLOW:
               outOfData = 1;
               break;
          case ERR_MP3_MAINDATA_UNDERFLOW:
               /* do nothing - next call to decode will provide
more mainData */
               break;
          case ERR_MP3_FREE_BITRATE_SYNC:
          default:
               outOfData = 1;
               break;
          }
     } else {
          // no error
          MP3GetLastFrameInfo(hMP3Decoder, &mp3FrameInfo);

          // Duplicate data in case of mono to maintain playback
speed
          if (mp3FrameInfo.nChans == 1) {
               for(int i = mp3FrameInfo.outputSamps;i >= 0;i--)
     {
                    samples[2 * i]=samples[i];
                    samples[2 * i + 1]=samples[i];
               }
               mp3FrameInfo.outputSamps *= 2;
          }
     }
     if (!outOfData) {
          // Converting all samples to float
          for (int i = 0; i < BLOCK_SIZE_FLOAT; i++) {
                    buf_in[i] = (float)samples[i];
          }

          arm_biquad_cascade_df1_f32(&filter32Hz_settings,
&buf_in[0], &buf_in[0], BLOCK_SIZE_FLOAT);

          arm_biquad_cascade_df1_f32(&filter64Hz_settings,
&buf_in[0], &buf_in[0], BLOCK_SIZE_FLOAT);

          arm_biquad_cascade_df1_f32(&filter125Hz_settings,
&buf_in[0], &buf_in[0], BLOCK_SIZE_FLOAT);

          arm_biquad_cascade_df1_f32(&filter250Hz_settings,
&buf_in[0], &buf_in[0], BLOCK_SIZE_FLOAT);

          arm_biquad_cascade_df1_f32(&filter500Hz_settings,
&buf_in[0], &buf_in[0], BLOCK_SIZE_FLOAT);

          arm_biquad_cascade_df1_f32(&filter1000Hz_settings,
&buf_in[0], &buf_in[0], BLOCK_SIZE_FLOAT);
```

```
          arm_biquad_cascade_df1_f32(&filter2000Hz_settings,
&buf_in[0], &buf_in[0], BLOCK_SIZE_FLOAT);

          arm_biquad_cascade_df1_f32(&filter4000Hz_settings,
&buf_in[0], &buf_in[0], BLOCK_SIZE_FLOAT);

          arm_biquad_cascade_df1_f32(&filter8000Hz_settings,
&buf_in[0], &buf_in[0], BLOCK_SIZE_FLOAT);

          arm_biquad_cascade_df1_f32(&filter16000Hz_settings,
&buf_in[0], &buf_in[0], BLOCK_SIZE_FLOAT);

          // Converting samples back to int
          for (int i = 0; i < BLOCK_SIZE_FLOAT; i++) {
               samples[i] = (int)(buf_in[i] * 0.05);
          }

          // Send buffer to codec for playing
          ProvideAudioBuffer(samples, mp3FrameInfo.outputSamps);
     }
}

/*
 * Called by the SysTick interrupt
 */
void TimingDelay_Decrement(void) {
     if (time_var1) {
          time_var1--;
     }
     time_var2++;
}

/*
 * Delay a number of systick cycles (1ms)
 */
void Delay(volatile uint32_t nTime) {
     time_var1 = nTime;
     while(time_var1){};
}

/*
 * Dummy function to avoid compiler error
 */
void _init() {

}

/*
 * Taken from
 * http://www.mikrocontroller.net/topic/252319
 */
static uint32_t Mp3ReadId3V2Text(FIL* pInFile, uint32_t unDataLen,
char* pszBuffer, uint32_t unBufferSize)
```

```
{
     UINT unRead = 0;
     BYTE byEncoding = 0;
     if((f_read(pInFile, &byEncoding, 1, &unRead) == FR_OK) &&
(unRead == 1))
     {
          unDataLen--;
          if(unDataLen <= (unBufferSize - 1))
          {
               if((f_read(pInFile, pszBuffer, unDataLen, &unRead)
== FR_OK) ||
                         (unRead == unDataLen))
               {
                    if(byEncoding == 0)
                    {
                         // ISO-8859-1 multibyte
                         // just add a terminating zero
                         pszBuffer[unDataLen] = 0;
                    }
                    else if(byEncoding == 1)
                    {
                         // UTF16LE unicode
                         uint32_t r = 0;
                         uint32_t w = 0;
                         if((unDataLen > 2) && (pszBuffer[0] ==
0xFF) && (pszBuffer[1] == 0xFE))
                         {
                              // ignore BOM, assume LE
                              r = 2;
                         }
                         for(; r < unDataLen; r += 2, w += 1)
                         {
                              // should be acceptable for 7 bit
ascii
                              pszBuffer[w] = pszBuffer[r];
                         }
                         pszBuffer[w] = 0;
                    }
               }
               else
               {
                    return 1;
               }
          }
          else
          {
               // we won't read a partial text
               if(f_lseek(pInFile, f_tell(pInFile) + unDataLen) !=
FR_OK)
               {
                    return 1;
               }
          }
```

```
	}
	else
	{
		return 1;
	}
	return 0;
}

/*
 * Taken from
 * http://www.mikrocontroller.net/topic/252319
 */
static uint32_t Mp3ReadId3V2Tag(FIL* pInFile, char* pszArtist,
uint32_t unArtistSize, char* pszTitle, uint32_t unTitleSize)
{
	pszArtist[0] = 0;
	pszTitle[0] = 0;

	BYTE id3hd[10];
	UINT unRead = 0;
	if((f_read(pInFile, id3hd, 10, &unRead) != FR_OK) || (unRead
!= 10))
	{
		return 1;
	}
	else
	{
		uint32_t unSkip = 0;
		if((unRead == 10) &&
				(id3hd[0] == 'I') &&
				(id3hd[1] == 'D') &&
				(id3hd[2] == '3'))
		{
			unSkip += 10;
			unSkip = ((id3hd[6] & 0x7f) << 21) | ((id3hd[7] &
0x7f) << 14) | ((id3hd[8] & 0x7f) << 7) | (id3hd[9] & 0x7f);

			// try to get some information from the tag
			// skip the extended header, if present
			uint8_t unVersion = id3hd[3];
			if(id3hd[5] & 0x40)
			{
				BYTE exhd[4];
				f_read(pInFile, exhd, 4, &unRead);
				size_t unExHdrSkip = ((exhd[0] & 0x7f) << 21)
| ((exhd[1] & 0x7f) << 14) | ((exhd[2] & 0x7f) << 7) | (exhd[3] &
0x7f);
				unExHdrSkip -= 4;
				if(f_lseek(pInFile,	f_tell(pInFile)	+
unExHdrSkip) != FR_OK)
				{
					return 1;
				}
```

```
                }
                uint32_t nFramesToRead = 2;
                while(nFramesToRead > 0)
                {
                     char frhd[10];
                     if((f_read(pInFile,  frhd,  10,  &unRead)  !=
FR_OK) || (unRead != 10))
                     {
                          return 1;
                     }
                     if((frhd[0] == 0) || (strncmp(frhd, "3DI", 3)
== 0))
                     {
                          break;
                     }
                     char szFrameId[5] = {0, 0, 0, 0, 0};
                     memcpy(szFrameId, frhd, 4);
                     uint32_t unFrameSize = 0;
                     uint32_t i = 0;
                     for(; i < 4; i++)
                     {
                          if(unVersion == 3)
                          {
                               // ID3v2.3
                               unFrameSize <<= 8;
                               unFrameSize += frhd[i + 4];
                          }
                          if(unVersion == 4)
                          {
                               // ID3v2.4
                               unFrameSize <<= 7;
                               unFrameSize += frhd[i + 4] & 0x7F;
                          }
                     }

                     if(strcmp(szFrameId, "TPE1") == 0)
                     {
                          // artist
                          if(Mp3ReadId3V2Text(pInFile,
unFrameSize, pszArtist, unArtistSize) != 0)
                          {
                               break;
                          }
                          nFramesToRead--;
                     }
                     else if(strcmp(szFrameId, "TIT2") == 0)
                     {
                          // title
                          if(Mp3ReadId3V2Text(pInFile,
unFrameSize, pszTitle, unTitleSize) != 0)
                          {
                               break;
                          }
```

```
                    nFramesToRead--;
                }
                else
                {
                    if(f_lseek(pInFile,   f_tell(pInFile)   +
unFrameSize) != FR_OK)
                    {
                        return 1;
                    }
                }
            }
        }
        if(f_lseek(pInFile, unSkip) != FR_OK)
        {
            return 1;
        }
    }

    return 0;
}
```

Додаток В

Лістинг коду програми для ESP32

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

// WiFi credentials for the ESP32 access point
const char* ssid = "USB_AUDIO_PLAYER";
const char* password = "12345678";

// Create a web server on port 80
AsyncWebServer server(80);

// UART baud rate 38400
#define UART_BAUD 38400

// Sample rate and Q factor used for coefficient calculation
#define FS 44100.0f
#define Q_FACTOR 1.4f // 0.703f

// Command identifiers for packets
#define CMD_VOLUME 0x01
#define CMD_COEF   0x02

// Packet headers
#define HEADER_BYTE1 0xAA
#define HEADER_BYTE2 0x55

// Packet sizes
#define VOLUME_PACKET_SIZE 5    // Header (2) + CMD (1) + Volume
(1) + Checksum (1)
#define COEF_PACKET_SIZE   25   // Header (2) + CMD (1) + FilterID
(1) + 5 floats (20) + Checksum (1)

// Frequency bands (in Hz) corresponding to each EQ slider (filter
IDs 0-9)
const float eqFrequencies[10] = {32.0f, 64.0f, 125.0f, 250.0f,
500.0f, 1000.0f, 2000.0f, 4000.0f, 8000.0f, 16000.0f};

// Global volume value (0 to 255)
volatile int volumeValue = 200;

// ---------- Functions for Packet Sending ----------

// Function to send a volume packet over UART
void sendVolumePacket(int volume) {
  uint8_t packet[VOLUME_PACKET_SIZE];
  packet[0] = HEADER_BYTE1;
  packet[1] = HEADER_BYTE2;
```

```
  packet[2] = CMD_VOLUME;
  packet[3] = (uint8_t)volume;

  // Calculate checksum: sum of header, CMD and volume modulo 256
  uint8_t checksum = 0;
  for (int i = 0; i < VOLUME_PACKET_SIZE - 1; i++) {
    checksum += packet[i];
  }
  packet[4] = checksum;

  Serial.write(packet, VOLUME_PACKET_SIZE);
}

// Function to send a coefficient packet over UART
void sendCoefPacket(uint8_t filterID, float coef[5]) {
  uint8_t packet[COEF_PACKET_SIZE];
  packet[0] = HEADER_BYTE1;
  packet[1] = HEADER_BYTE2;
  packet[2] = CMD_COEF;
  packet[3] = filterID;

  // Pack the 5 float coefficients (each 4 bytes) into packet
(little-endian)
  for (int i = 0; i < 5; i++) {
    union {
      float f;
      uint8_t b[4];
    } data;
    data.f = coef[i];
    memcpy(packet + 4 + i*4, data.b, 4);
  }

  // Calculate checksum: sum of first 24 bytes modulo 256
  uint8_t checksum = 0;
  for (int i = 0; i < COEF_PACKET_SIZE - 1; i++) {
    checksum += packet[i];
  }
  packet[COEF_PACKET_SIZE - 1] = checksum;

  Serial.write(packet, COEF_PACKET_SIZE);
}

// ---------- Function to Calculate Peaking EQ Coefficients -----
-----
// This example uses a standard peaking EQ formula (RBJ Cookbook).
// It calculates 5 coefficients: [a0', a1', a2', b1', b2'] to be
used in a biquad filter.
// The filter difference equation is assumed to be:
// y[n] = a0'*x[n] + a1'*x[n-1] + a2'*x[n-2] - b1'*y[n-1] -
b2'*y[n-2]
void calcPeakingCoeffs(float fc, float Q, float gain_dB, float fs,
float coef[5]) {
  float A = pow(10.0f, gain_dB / 40.0f);
```

```
  float w0 = 2.0f * PI * fc / fs;
  float cosw0 = cosf(w0);
  float sinw0 = sinf(w0);
  float alpha = sinw0 / (2.0f * Q);

  // Standard peaking EQ formulas (RBJ Cookbook):
  float b0 = 1.0f + alpha * A;
  float b1 = -2.0f * cosw0;
  float b2 = 1.0f - alpha * A;
  float a0 = 1.0f + alpha / A;
  float a1 = -2.0f * cosw0;
  float a2 = 1.0f - alpha / A;

  // Normalize the coefficients (dividing all by a0)
  coef[0] = b0 / a0;  // a0'
  coef[1] = b1 / a0;  // a1'
  coef[2] = b2 / a0;  // a2'
  coef[3] = a1 / a0;  // b1' (feedback, note the minus sign is
applied in filter processing)
  coef[4] = a2 / a0;  // b2'
}

// Calculate Low-Shelf filter coefficients
// fc: cutoff frequency (Hz)
// S: shelf slope (usually 1.0)
// gain_dB: boost or cut in decibels (positive = boost, negative
= cut)
// fs: sampling rate (Hz)
// coef: output array of 5 coefficients [a0', a1', a2', b1', b2']
void calcLowShelfCoeffs(float fc, float S, float gain_dB, float
fs, float coef[5]) {
    float A = powf(10.0f, gain_dB / 40.0f);
    float w0 = 2.0f * M_PI * fc / fs;
    float cosw0 = cosf(w0);
    float sinw0 = sinf(w0);
    // Calculate alpha using the shelf slope S
    float alpha = sinw0 / 2.0f * sqrtf((A + 1.0f/A) * (1.0f/S -
1.0f) + 2.0f);

    float b0 =    A * ((A + 1.0f) - (A - 1.0f) * cosw0 + 2.0f *
sqrtf(A) * alpha);
    float b1 =  2.0f * A * ((A - 1.0f) - (A + 1.0f) * cosw0);
    float b2 =    A * ((A + 1.0f) - (A - 1.0f) * cosw0 - 2.0f *
sqrtf(A) * alpha);
    float a0 =        (A + 1.0f) + (A - 1.0f) * cosw0 + 2.0f *
sqrtf(A) * alpha;
    float a1 =   -2.0f * ((A - 1.0f) + (A + 1.0f) * cosw0);
    float a2 =        (A + 1.0f) + (A - 1.0f) * cosw0 - 2.0f *
sqrtf(A) * alpha;

    // Normalize coefficients by a0
    coef[0] = b0 / a0;  // a0'
    coef[1] = b1 / a0;  // a1'
```

```
    coef[2] = b2 / a0;  // a2'
    coef[3] = a1 / a0;  // b1' (note: used with a minus in the
difference eq.)
    coef[4] = a2 / a0;  // b2'
}

// Calculate High-Shelf filter coefficients
// fc: cutoff frequency (Hz)
// S: shelf slope (usually 1.0)
// gain_dB: boost or cut in decibels (positive = boost, negative
= cut)
// fs: sampling rate (Hz)
// coef: output array of 5 coefficients [a0', a1', a2', b1', b2']
void calcHighShelfCoeffs(float fc, float S, float gain_dB, float
fs, float coef[5]) {
    float A = powf(10.0f, gain_dB / 40.0f);
    float w0 = 2.0f * M_PI * fc / fs;
    float cosw0 = cosf(w0);
    float sinw0 = sinf(w0);
    float alpha = sinw0 / 2.0f * sqrtf((A + 1.0f/A) * (1.0f/S -
1.0f) + 2.0f);

    float b0 =    A * ((A + 1.0f) + (A - 1.0f) * cosw0 + 2.0f *
sqrtf(A) * alpha);
    float b1 = -2.0f * A * ((A - 1.0f) + (A + 1.0f) * cosw0);
    float b2 =    A * ((A + 1.0f) + (A - 1.0f) * cosw0 - 2.0f *
sqrtf(A) * alpha);
    float a0 =        (A + 1.0f) - (A - 1.0f) * cosw0 + 2.0f *
sqrtf(A) * alpha;
    float a1 =    2.0f * ((A - 1.0f) - (A + 1.0f) * cosw0);
    float a2 =        (A + 1.0f) - (A - 1.0f) * cosw0 - 2.0f *
sqrtf(A) * alpha;

    // Normalize coefficients by a0
    coef[0] = b0 / a0;  // a0'
    coef[1] = b1 / a0;  // a1'
    coef[2] = b2 / a0;  // a2'
    coef[3] = a1 / a0;  // b1'
    coef[4] = a2 / a0;  // b2'
}

// ---------- Web Server HTML Page ----------

const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML>
<html>
<head>
  <title>ESP32 Audio Control</title>
   <meta  name="viewport"  content="width=device-width,  initial-
scale=1">
  <style>
    body {
      background-color: #000;
```

```
      color: #ffffff;
      font-family: Arial, sans-serif;
      text-align: center;
      margin: 0;
      padding: 0px;
    }
    h2 {
      color: #FFD700; /* Gold/Yellow */
    }
    /* Horizontal slider for volume */
    .volume-container {
      margin-bottom: 30px;
    }
    .volume-slider {
      width: 80%;
    }
    /* Vertical EQ sliders */
    .eq-container {
      display: flex;
      justify-content: center;
      align-items: flex-end;
      height: 300px;
      border: 2px solid #FFD700;
      border-radius: 10px;
      background-color: #222;
    }
    .eq-slider {
      -webkit-appearance: none;  /* Remove default styling */
      width: 200px; /* This is the length of the slider track */
      height: 15px; /* Thickness of the slider track */
      transform: rotate(-90deg);
      margin: 100px -25px;
      background: #FFD700; /* Yellow track */
      border-radius: 5px;
      outline: none;
    }
    .eq-slider::-webkit-slider-thumb {
      appearance: none;
      width: 25px;
      height: 25px;
      background: #ffffff; /* White thumb */
      cursor: pointer;
      border-radius: 50%;
    }
    .eq-label {
      margin-top: 5px;
      font-size: 14px;
    }
     /* Mobile-friendly fallback: on smaller screens, display
horizontal sliders */
    @media (max-width: 600px) {
      .eq-container {
        flex-direction: column;
```

```
        height: auto;
        align-items: center;
      }
      .eq-band {
        width: 100%;
        margin: 5px 0;
      }
      .slider-wrapper {
        width: 100%;
        height: auto;
        position: static;
      }
      .eq-slider {
        position: static;
        transform: none;
        width: 200px;
        margin: 10px auto;
      }
    }
  </style>
</head>
<body>
  <h2>USB Audio Player Control</h2>
  <div class="volume-container">
    <div>
      <span>Volume</span><br>
          <input  type="range"  min="0"  max="100"  value="50"
class="volume-slider" id="volumeSlider">
      <output id="volOut">50</output>
    </div>
  </div>
  <div class="eq-container">
    <!-- Create 10 vertical sliders for EQ bands -->
    <div>
      <input type="range" min="-20" max="20" value="0" class="eq-
slider" id="eq0">
          <div  class="eq-label">32  Hz<br><span  id="eqOut0">0
dB</span></div>
    </div>
    <div>
      <input type="range" min="-20" max="20" value="0" class="eq-
slider" id="eq1">
          <div  class="eq-label">64  Hz<br><span  id="eqOut1">0
dB</span></div>
    </div>
    <div>
      <input type="range" min="-20" max="20" value="0" class="eq-
slider" id="eq2">
         <div  class="eq-label">125  Hz<br><span  id="eqOut2">0
dB</span></div>
    </div>
    <div>
```

```
      <input type="range" min="-20" max="20" value="0" class="eq-
slider" id="eq3">
          <div  class="eq-label">250  Hz<br><span  id="eqOut3">0
dB</span></div>
    </div>
    <div>
      <input type="range" min="-20" max="20" value="0" class="eq-
slider" id="eq4">
          <div  class="eq-label">500  Hz<br><span  id="eqOut4">0
dB</span></div>
    </div>
    <div>
      <input type="range" min="-20" max="20" value="0" class="eq-
slider" id="eq5">
          <div  class="eq-label">1000  Hz<br><span  id="eqOut5">0
dB</span></div>
    </div>
    <div>
      <input type="range" min="-20" max="20" value="0" class="eq-
slider" id="eq6">
          <div  class="eq-label">2000  Hz<br><span  id="eqOut6">0
dB</span></div>
    </div>
    <div>
      <input type="range" min="-20" max="20" value="0" class="eq-
slider" id="eq7">
          <div  class="eq-label">4000  Hz<br><span  id="eqOut7">0
dB</span></div>
    </div>
    <div>
      <input type="range" min="-20" max="20" value="0" class="eq-
slider" id="eq8">
          <div  class="eq-label">8000  Hz<br><span  id="eqOut8">0
dB</span></div>
    </div>
    <div>
      <input type="range" min="-20" max="20" value="0" class="eq-
slider" id="eq9">
         <div  class="eq-label">16000  Hz<br><span  id="eqOut9">0
dB</span></div>
    </div>
  </div>
  <script>
    // Volume slider event
    var volumeSlider = document.getElementById("volumeSlider");
    var volOut = document.getElementById("volOut");
    volumeSlider.oninput = function() {
      volOut.innerHTML = this.value;
      var xhr = new XMLHttpRequest();
      xhr.open("GET", "/setVolume?value=" + this.value, true);
      xhr.send();
    };
```

```
    // EQ slider event handler
    function updateEQ(filterIndex, sliderId, outputId) {
      var slider = document.getElementById(sliderId);
      var output = document.getElementById(outputId);
      slider.oninput = function() {
        output.innerHTML = this.value + " dB";
        // Send command to set EQ parameters for this filter
        var xhr = new XMLHttpRequest();
        xhr.open("GET", "/setEQ?filter=" + filterIndex + "&gain="
+ this.value, true);
        xhr.send();
      };
    }
    updateEQ(0, "eq0", "eqOut0");
    updateEQ(1, "eq1", "eqOut1");
    updateEQ(2, "eq2", "eqOut2");
    updateEQ(3, "eq3", "eqOut3");
    updateEQ(4, "eq4", "eqOut4");
    updateEQ(5, "eq5", "eqOut5");
    updateEQ(6, "eq6", "eqOut6");
    updateEQ(7, "eq7", "eqOut7");
    updateEQ(8, "eq8", "eqOut8");
    updateEQ(9, "eq9", "eqOut9");
  </script>
</body>
</html>
)rawliteral";

// ---------- ESP32 Code for Handling Requests ----------

void notFound(AsyncWebServerRequest *request) {
  request->send(404, "text/plain", "Not found");
}

void setup() {
  // Start Serial for UART communication with STM32
  Serial.begin(UART_BAUD);

  // Set ESP32 as an access point
  WiFi.softAP(ssid, password);
  Serial.print("AP IP address: ");
  Serial.println(WiFi.softAPIP());
  Serial.println();

  // Serve the HTML page
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send_P(200, "text/html", index_html);
  });

  // Endpoint to set volume (sends a volume packet)
    server.on("/setVolume",   HTTP_GET,   [](AsyncWebServerRequest
*request) {
    if (request->hasParam("value")) {
```

```
      int vol = request->getParam("value")->value().toInt();
      sendVolumePacket(vol);
      request->send(200, "text/plain", "Volume OK");
    } else {
      request->send(400, "text/plain", "Bad Request");
    }
  });

  // Endpoint to set EQ for a specific filter band (sends a
coefficient packet)
    server.on("/setEQ",    HTTP_GET,    [](AsyncWebServerRequest
*request) {
    if (request->hasParam("filter") && request->hasParam("gain"))
{
     int filterID = request->getParam("filter")->value().toInt();
              float    gain_dB    =    request->getParam("gain")-
>value().toFloat();

      // For our example, use a fixed Q factor.
      float Q = Q_FACTOR;
      // Get the center frequency from our predefined table.
      float fc = eqFrequencies[filterID];

     // Calculate filter coefficients based on peaking EQ formula.
      float coef[5];

      //calcLowShelfCoeffs(fc, 1, gain_dB, FS, coef);
      //calcHighShelfCoeffs(fc, 1, gain_dB, FS, coef);
      calcPeakingCoeffs(fc, Q, gain_dB, FS, coef);

      // Send the coefficient packet for this filter band.
      sendCoefPacket((uint8_t)filterID, coef);
      sendCoefPacket((uint8_t)filterID, coef);
      sendCoefPacket((uint8_t)filterID, coef);

      request->send(200, "text/plain", "EQ OK");
    } else {
      request->send(400, "text/plain", "Bad Request");
    }
  });

  server.onNotFound(notFound);
  server.begin();
}

void loop() {
  // The async server handles requests in the background.
  // No blocking code here.
}
```

Додаток Г

Перелік елементів до схеми електричної принципової

Поз. позначення	*Найменування*	*Кіл.*	*Примітка*
A1	Блок MCU	1	
	Конденсатори		
C21, C22	CC0603JRNPO9BN6R8 YAGEO	2	
C13,C15,C16, C17,C18,C20, C23,C29,C30, C31,C32,C33, C34,C35	CC0603KRX7R9BB104 YAGEO	14	
C19	CC0603KRX7R9BB105 YAGEO	1	
C36	CC0805KRX7R9BB225 YAGEO	1	
C22	CC0805MKX5R7BB106 YAGEO	1	
C14	CC0603JRNPO9BN221 YAGEO	1	
L1	Дросель BMA1608-601	1	
	Резистори		
R37	RC0603JR-070RL YAGEO	1	
R20, R33	RC0603JR-074K7L YAGEO	2	
R34, R38	RC0603JR-0710KL YAGEO	2	
R21, R22	RC0603FR-07100KL YAGEO	2	
R23, R24	RC0603FR-07340KL YAGEO	2	
U1	Мікроконтролер STM32F407VGT6	1	
	Кварцові резонатори		
X2	MC106-C-080-32.768	1	
X3	MC106-C-080-8M	1	
SB18, SB19	Джампери MJ-0-6	2	

					КСКП 123.232-42 81 01 ПЕ		
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розробив		*Пуляк М.Я.*			*Комп'ютеризована система USB аудіоплеєра з підтримкою веб-еквалайзера Перелік елементів*	*Літ.*	*Арк.*	*Акрушів*
Перевірив		*Варавін А.В.*				*н*	*1*	*3*
Рецензент		*Бойко І.В.*				*ТНТУ, каф. КС, гр. СІ-42*		
Н. Контр.		*Тиш Є.В.*						
Зав. каф.		*Осухівська Г.М.*						

Поз. позначення	*Найменування*	*Кіл.*	*Примітка*
A2	Блок CS43L22	1	
	Конденсатори		
C46	CC0603JRNPO9BN560 YAGEO	1	
C55	CC0603KRX7R9BB223 YAGEO	1	
C40,C41,C43, C44,C45	CC0603KRX7R9BB104 YAGEO	5	
C42,C47,C50, C51,C53,C57	CC0603KRX7R9BB105 YAGEO	6	
C59	CC0805MKX5R7BB106 YAGEO	1	
C48	CC0603JRNPO9BN221 YAGEO	1	
C49,C52,C54, C56	SMD 0603/0805	4	
CN4	З'єднувач ST-225-02 CUI Devices	1	
	Резистори		
R65	RC0603JR-070RL YAGEO	1	
R61	RC0603FR-0751RL YAGEO	1	
R43	RC0603JR-0710KL YAGEO	1	
R48, R50	RC0603FR-07100KL YAGEO	2	
R47, R49	RC0603FR-07340KL YAGEO	2	
R46,R51-R55	RC0603JR-074K7L YAGEO	6	
U4	CS43L22-CNZ	1	
A3	Блок Micro USB	1	
C49	Конденсатор CC0805MKX5R6BB475 YAGEO	1	
CN5	Роз'єм Molex 475900001	1	
	Світлодіоди		
LD6	LTST-C190KRKT Lite-On	1	
LD7	KT-SMD0603-G	1	

Поз. позначення	*Найменування*	*Кіл.*	*Примітка*
	Резистори		
R46, R60	RC0603JR-070RL YAGEO	2	
R58	RC0603JR-0722RL YAGEO	1	
R51, R56	RC0603JR-074K7L YAGEO	2	
R45	RC0603JR-0710KL YAGEO	1	
R57	RC0603FR-07340KL YAGEO	1	
T1	Транзистор S9013	1	
U2	Перемикач живлення STMPS2141STR	1	
U3	Фільтр EMIF02-USB03F2	1	
A4	Блок ESP32	1	
U5	ESP32-WROVER-DEV	1	