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

**Комп'ютерно-інформаційних систем і програмної інженерії**
(повна назва факультету )
**Програмної інженерії**
(повна назва кафедри)

# КВАЛІФІКАЦІЙНА РОБОТА

**на здобуття освітнього ступеня**

**бакалавр**
(назва освітнього ступеня)

**на тему:** Розробка та тестування програмного забезпечення застосунку електронної симуляції електрогітарних звукових ефектів з використанням фреймворку Juce

Виконав: студент IV курсу, групи СП-41
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва спеціальності)

_____ Іванков А. Р.
(підпис) (прізвище та ініціали)

Керівник _____ Стоянов Ю.М.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю.М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М.Р.
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль
2026

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

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

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Петрик М.Р.
(підпис) (прізвище та ініціали)
« 6 » квітня 2026 р.

# ЗАВДАННЯ
## НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Іванков Артем Русланович

1. Тема роботи Розробка та тестування програмного забезпечення застосунку електронної симуляції електрогітарних звукових ефектів з використанням фреймворку Juce

Керівник роботи _____
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом по університету від « 06 » квітня 2026 року № _____

2. Термін подання студентом роботи 22.06.2026

3. Вихідні дані до роботи наукові літературні джерела

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Вступ. 1 Аналіз вимог та огляд предметної області.
2. Проєктування та конструювання
3. Тестування.
4. Безпека життєдіяльності, основи охорони праці.
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень, слайдів)
1. Слайди презентації. 2. Огляд аналогів. 3 Діаграми компонентів.
4. Діаграма варіантів використання. 5. Діаграма процесу дискретизації
6. Процес входу та виходу сигналу. 7. Візуалізація синусоїдної, трикутної та квадратної хвилей
8. Лістинг коду.

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Безпека життєдіяльності, основи охорони праці | | | |
| | | | |

7. Дата видачі завдання ___6 квітня_____ 2026 р.

**КАЛЕНДАРНИЙ ПЛАН**

| № з/п | Назва етапів кваліфікаційної роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|---|
| 1. | *Розробка технічного завдання* | *6.04 – 12.04* | |
| 2. | *Робота над першим розділом «Огляд предметної області»* | *16.04 – 26.04* | |
| 3. | *Робота над другим розділом «Проєктування та реалізація»* | *27.04 – 17.05* | |
| 4. | *Робота над третім розділом «Тестування»* | *18.05 – 24.05* | |
| | *Робота над четвертим розділом «Безпека життєдіяльності, основи охорони праці»* | *25.05 – 31.05* | |
| 5. | *Оформлення пояснювальної записки і графічного матеріалу* | *1.06 – 7.06* | |
| 6. | *Перевірка на академічний плагіат, перевірка керівником та консультантами* | *8.06 – 14.06* | |
| 7. | *Попередній захист кваліфікаційної роботи бакалавра* | *15.06 – 21.06* | |
| 8. | *Захист кваліфікаційної роботи бакалавра* | | |
| | | | |
| 9. | | | |
| 10. | | | |
| 11. | | | |
| 12. | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

# АНОТАЦІЯ

Іванков А. Р. Розробка та тестування програмного забезпечення застосунку електронної симуляції електрогітарних звукових ефектів з використанням фреймворку Juce : робота на здобуття кваліфікаційного ступеня бакалавра : спец. 121 - інженерія програмного забезпечення / наук. кер. Ю. М. Стоянов. Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2026. 54 с. // С. - 54, табл. - 2, рис. - 11, бібліогр. – 20, слайд. – 13, додат. - 2 ,

Ключові слова: інженерія програмного забезпечення, Real-time розробка, Digital Signal Processor.

Метою роботи є створення застосунку електронної симуляції електрогітарних звукових ефектів.

У першому розділі проведено аналіз існуючих програмних рішень для обробки гітарного аудіосигналу, розглянуто принципи цифрової обробки звуку, особливості реалізації аудіоефектів та сучасні засоби створення аудіозастосунків.

У другому розділі досліджено вимоги до системи, подано архітектуру підсистеми, обґрунтовано вибір фреймворку.

У третьому розділі описано реалізацію системи, досліджено роботу основних модулів та проведено тестування функціональності й працездатності програмного забезпечення.

У четвертому розділі розглянуто питання охорони праці та безпеки життєдіяльності.

Об’єктом дослідження є процес цифрової обробки аудіосигналів та програмна реалізація електрогітарних звукових ефектів.

Предметом дослідження є методи, алгоритми та програмні засоби створення застосунку обробки звукового сигналу в реальному часі. Методи дослідження включають аналіз предметної області, проєктування програмних систем, моделювання архітектури застосунку, реалізацію алгоритмів цифрової обробки сигналів, а також функціональне тестування програмного забезпечення.

# ABSTRACT

Artem Ivankov. Development and testing of software for an electronic simulation application of electric guitar sound effects using the Juce framework : bachelor thesis : specialty 121 "Software Engineering": Ternopil Ivan Puluj National Technical University, 2026, 54p. // Pages - 54, tables - 2, figures - 11, references - 20, presentation slides – 13, appendices -2.

Keywords: software engineering, Real-time розробка, Digital Signal Processor.

The aim of this work is to create an application for the electronic simulation of electric guitar sound effects.

The first chapter analyzes existing software solutions for processing guitar audio signals, examines the principles of digital audio processing, discusses the specifics of implementing audio effects, and reviews modern tools for developing audio applications.

The second chapter examines system requirements, presents the subsystem architecture, and justifies the choice of framework.

The third chapter describes the system implementation, examines the operation of the main modules, and tests the functionality and performance of the software.

The fourth chapter addresses occupational health and safety issues.

The subject of the study is the process of digital audio signal processing and the software implementation of electric guitar sound effects.

The subject of the research is the methods, algorithms, and software tools for creating a real-time audio signal processing application. The research methods include domain analysis, software system design, application architecture modeling, implementation of digital signal processing algorithms, and functional testing of the software.

# ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

DSP (Digital Signal Processor) – цифровий сигнальний процесор.

AMP (Amplifier) – підсилювач для електрогітари.

UI (User Interface) – інтерфейс користувача.

DAW (Digital Audio Workstation) – цифрова звукова робоча станція

Sample (“проба”, “вибірка” або “семпл”) – числове значення амплітуди аналогового аудіосигналу, отримане в певний момент часу під час процесу дискретизації.

Audio buffer (аудіобуфер) – масив семплів, який передається на обробку.

IDE (Integrated Development Environment) – інтегроване середовище розробки.

Distortion (Дисторшн або спотворення, викривлення) – ефект перевантаження який використовується в музиці.

Фрейм (Фрейм) – набір семплів усіх каналів, що відповідають одному моменту часу.

Осцилятор – фізична або математична система, яка здійснює регулярні, періодичні коливання.

**ЗМІСТ**

# ВСТУП

Сучасні інформаційні технології активно використовуються в музичній індустрії для створення, обробки та відтворення аудіосигналів. Одним із найбільш поширених напрямів цифрової обробки звуку є моделювання музичного обладнання та звукових ефектів за допомогою програмного забезпечення. Використання цифрових аудіопроцесорів дозволяє музикантам отримувати широкий спектр звукових характеристик без необхідності придбання великої кількості фізичних пристроїв, що робить програмні рішення доступними, гнучкими та зручними у використанні.

Важливість програмних рішень для обробки гітарного сигналу обумовлена не лише їх застосуванням у професійній музичній індустрії, а й зростаючою популярністю серед широкого кола музикантів-аматорів. Традиційно для формування необхідного звучання електрогітари використовуються фізичні педалі ефектів та інше спеціалізоване обладнання, яке може бути дорогим, займати значний простір та потребувати додаткових зусиль для транспортування і налаштування. Програмні симулятори дозволяють замінити або доповнити таке обладнання, надаючи користувачам доступ до різноманітних ефектів за допомогою персонального комп’ютера. Це особливо актуально для музикантів, які часто подорожують, виступають у різних місцях або займаються музикою в домашніх умовах і не мають можливості використовувати велику кількість фізичних пристроїв.

Практичне значення роботи полягає у створенні програмного застосунку для обробки гітарного аудіосигналу в режимі реального часу. Розроблене програмне забезпечення може використовуватися як музикантами-початківцями, так і досвідченими виконавцями для отримання різноманітних звукових ефектів без використання великої кількості фізичного обладнання. Крім того, застосунок може слугувати основою для подальшого розширення функціоналу, додавання нових ефектів та моделювання гітарних підсилювачів.

# 1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

Електрогітарні звукові ефекти є важливою складовою сучасної музичної індустрії та широко використовуються для формування унікального звучання музичних інструментів. Традиційно для зміни характеристик аудіосигналу застосовуються спеціалізовані апаратні пристрої, такі як педалі ефектів, процесори обробки звуку та підсилювачі. Проте розвиток цифрових технологій дозволив реалізовувати аналогічні функції програмними засобами, забезпечуючи високу якість обробки сигналу, гнучкість налаштувань та доступність для широкого кола користувачів.

Сучасні програмні рішення для обробки гітарного сигналу надають можливість моделювати роботу аналогових ефектів у режимі реального часу, створювати власні ланцюги обробки та налаштовувати параметри звучання відповідно до потреб музиканта. Для розробки таких систем використовуються спеціалізовані технології цифрової обробки сигналів, сучасні мови програмування та програмні фреймворки, що забезпечують ефективну роботу з аудіоданими.

## 1.1 Огляд аналогів

У ході проведеного огляду аналогів було виявлено кілька систем зі схожим функціоналом.

### 1.1.1 AmpliTube 5

AmpliTube [2] є найпопулярнішим варіантом електронної симуляції звукових ефектів для початківців та професіоналів в сфері. Перша версія цієї програми вийшла ще в 2003 році. За більше ніж 20 років застосунок накопичив велику кількість можливих звукових ефектів та типів підсилювачів. Також має безкоштовну пробну версію з парою доступних ефектів[1].


Рисунок 1.1 – Ілюстрація функціоналу AmpliTube 5 продемонстрована на офіційному сайті застосунку

### 1.1.2 Neural DSP

Neural DSP [3] не є одним застосунком для симуляції ефектів, але є дистриб'ютором таких програм. В їх асортименті є велика кількість різноманітних плагінів з симуляцією продуктів різних брендів. Велика різноманітність продуктів також супроводжується тим що кожен продукт треба купувати окремо[2].

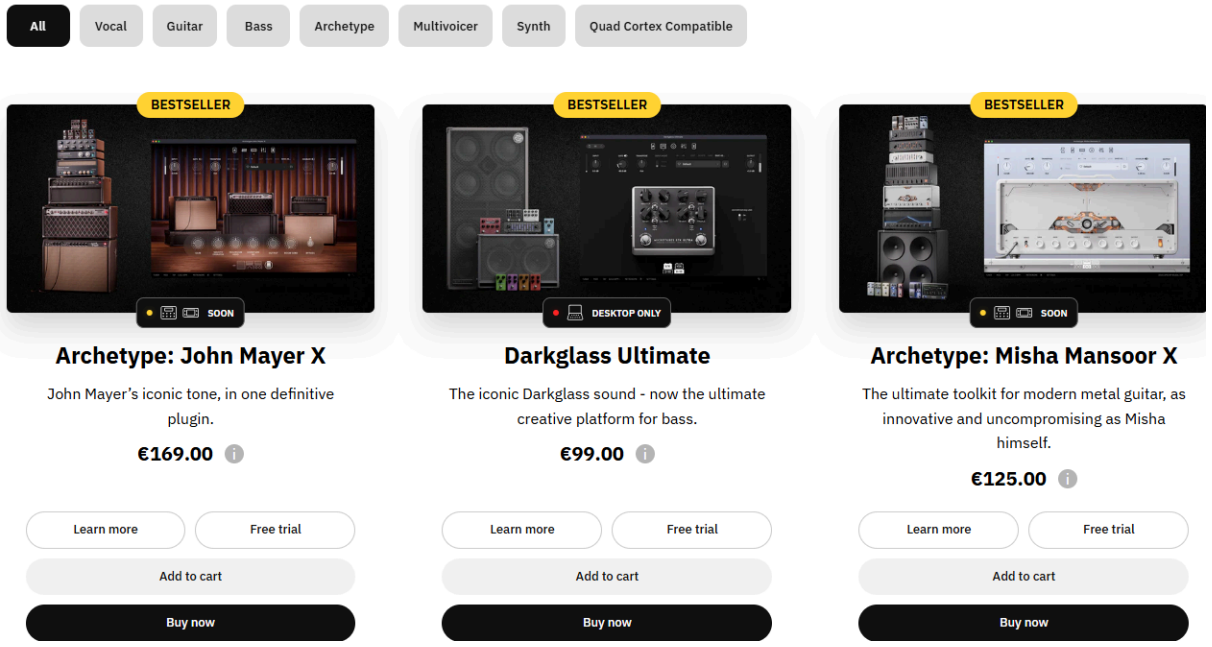


Рисунок 1.2 – Магазин доступних плагінів від Neural DSP

### 1.1.3 Guitar Rig 7

Guitar Rig [4] є окремою програмою схожою в своєму функціоналі на AmpliTube. Перша версія програми була представлена у 2004 році, а сучасна сьома версія містить широкий набір інструментів для обробки аудіосигналу. До складу програми входять різноманітні моделі підсилювачів, кабінетів, мікрофонів та ефектів. Програма може використовуватись як окремий застосунок або як плагін у цифрових аудіостанціях (DAW). Программу можно як купити повністю, так і отримати через підписку. Підписка також надає всі інші сервіси компанії розробника[3].


Рисунок 1.3 – Інтерфейс Guitar Rig 7

### 1.1.4 Порівняння функціоналу AmpliTube 5, Guitar Rig 7 та Neural DSP

Для кращого розуміння функціоналу кожного застосунку було створено таблицю порівнянь характеристик та функції програм які розцінив як важливі для розуміння потрібних функцій та створення вимог до прототипу проєкту дипломної. Порівняння наведені в таблиці 1.1

Таблиця 1.1 - Порівняння характеристик аналогів

| **Характеристика** | **AmpliTube 5** | **Guitar Rig 7** | **Neural DSP** |
|---|---|---|---|
| **1** | **2** | **3** | **4** |
| Формат застосунку | Окремий застосунок, плагін | Окремий застосунок, плагін | Плагін |
| Ціна | €199.99 для базової версії. €99.99 для скороченої версії. | 199.00 US$ для базової версії. 250 $ річна підписка на всі сервіси компанії. Базова версія Guitar Rig 7 включно. | Ціни варіюються. Мінімальний комплект €45. Максимальний €165. |
| Безкоштовна версія | Існує безкоштовна версія. Єдине обмеження є в кількості доступних з початку ефектів. | Існує безкоштовна версія з обмеженою кількістю доступних ефектів. Є демо версія повного застосунку з лімітом в 30 хилин. | Кожен плагін має 14 днів безкоштовного випробування. |
| Використання декількох ефектів одночасно | Так | Так | Так, в межах одного плагіну |
| Вибір порядку ефектів | Так | Так | Так, в межах одного плагіну |
| Збереження налаштувань | Так | Так, в повній версії застосунку | Так |

З цього порівняння ми можемо виділити декілька спільних рис які також буде варто додати в нами розроблюваний застосунок.

## 1.2 Огляд існуючих технологій реалізації

Розробка сучасних аудіозастосунків потребує використання спеціалізованих технологій та програмних засобів, які забезпечують обробку аудіосигналів у режимі реального часу, створення графічного інтерфейсу користувача та підтримку різних операційних систем. Для реалізації подібних систем використовуються як універсальні мови програмування, так і спеціалізовані фреймворки для роботи зі звуком.

Однією з найпоширеніших мов програмування для створення аудіопрограм є мова C++. Її використання обумовлене високою продуктивністю, ефективним управлінням пам’яттю та можливістю роботи з низькорівневими механізмами операційної системи. Завдяки цьому C++ широко застосовується при розробці програм, які виконують складну цифрову обробку сигналів у режимі реального часу.

DSP є складною темою та розробка застосунку з нуля займе багато часу та ресурсів. Тому краще використати вже готові фреймворки. Найпопулярнішим, якщо не єдиним, фреймворком для цифрової обробки звуку є Juce. Причинами обрати juce є:

- Вбудований ввід\вивід аудіо;
- Генерація UI;
- Великі обсяги навчального матеріалу та документації;
- Проєкт з відкритим кодом;

Фреймворк націлений на створення аудіо-плагінів для DAW і підтримує такі формати як:

- VST3 (Virtual Studio Technology, version 3) ;
- AU (Audio Units);
- AAX (Avid Audio eXtension);
- AUv3 (Audio Unit v3);
- LV2;

Кожен формат має свою різну архітектуру. Тому код написаний для одного формату може не співпадати з іншим, і розробнику прийдеться переписувати код з нуля. І так для кожного нового формату. І саме цей процес фреймворк вирішив взяти на себе. Juce може перетворити один код в декілька різних форматів.

Але це не все. Фреймворк також підтримує створення “Standalone” програми. Тобто створення “плагіну” як окремий застосунок. Саме так ми можемо створити окремий застосунок використовуючи цей фреймворк.

Для реалізації алгоритмів цифрової обробки сигналів також широко

використовуються спеціалізовані бібліотеки, зокрема модулі DSP (Digital Signal Processing). Вони забезпечують засоби для роботи з фільтрами, генераторами сигналів, динамічною обробкою звуку та іншими компонентами аудіосистем. Але всі необхідні бібліотеки також йдуть за замовчуванням в цьому фреймворку. Тому ми можемо не витрачати час на пошуки потрібних бібліотек.

Також у процесі створення сучасних аудіозастосунків, та і любих застосунків на c++ загалом, важливу роль відіграють системи автоматизованого складання проєктів. Однією з найбільш популярних технологій є CMake, яка дозволяє автоматизувати процес генерації проєктів для різних середовищ розробки та забезпечує зручне керування залежностями програмного забезпечення.

Таким чином, використання мови програмування C++, фреймворку JUCE, технологій цифрової обробки сигналів та системи складання CMake створює ефективну основу для розробки сучасного застосунку електронної симуляції електрогітарних звукових ефектів.

### 1.3 Висновки до першого розділу

В цьому розділі було проведено огляд предметної області, пов’язаної з електронною симуляцією електрогітарних звукових ефектів. Встановлено, що програмні рішення для обробки гітарного сигналу є важливою складовою сучасної музичної індустрії та дозволяють реалізувати широкий спектр звукових ефектів без використання великої кількості фізичного обладнання.

Під час аналізу існуючих аналогів було розглянуто популярні програмні продукти а також проаналізовано сучасні технології реалізації аудіозастосунків. Визначено, що мова програмування C++ забезпечує необхідний рівень продуктивності для цифрової обробки аудіосигналів у режимі реального часу.

# 2 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ

В даному розділі описано процес проєктування розроблюваних компонентів.

Також наведено програмні засоби та особливості реалізації компонент системи.

## 2.1 Вимоги до системи

В минулому розділі ми дослідили програми-аналоги що допомагає нам визначити вимоги до застосунку.

### 2.1.1 Вимоги до застосунку

Застосунок електронної симуляції створений для полегшення гри на гітарі звичайним користувачам без особливих музикальних потреб. Застосунок націлений на заміну фізичних компонентів які потрібні для гри на електрогітарі. Цим чином полегшити гру на гітарі при частих переїздах.

Функціональні вимоги[17]:

– можливість вибору ефектів;
– можливість вибору одночасно декількох ефектів;
– можливість вибору порядку ефектів;
– зміна налаштувань ефектів;
– можливість зберегти стан програми;

Нефункціональні вимоги:

1. Збереження:

– Автоматичне зберігання: програма має самостійно зберігати стан при виході з програми та встановлювати його при повторному запуску.
– Стан програми має зберігатися в окремому файлі

2. Користувацький інтерфейс:

– Легкий інтерфейс: програма має мати прости та зрозумілий

користувацький інтерфейс.

3. Ефективність:

– Час виконання алгоритму ефектів має не перевищувати довжину аудіо буфера який використовується.

– Застосунок не має мати помітну затримку вводу та виводу сигналу.

4. Безпека користувача та обладнання:

– Програма не має мати помітні звукові дефекти які можуть негативно вплинути на здоров’я користувача.

– Програма не має викликати проблеми з слуховим обладнанням користувача.

### 2.2 Архітектура системи

Архітектура розробленого програмного забезпечення побудована за модульним принципом з розділенням логіки обробки аудіосигналу, графічного інтерфейсу користувача та керування параметрами ефектів. Основою застосунку є фреймворк JUCE, який забезпечує роботу з аудіопристроями, обробку аудіобуферів у реальному часі, створення графічного інтерфейсу та можливість подальшої збірки програми у форматі standalone-застосунку або аудіоплагіна.

Центральним компонентом архітектури є аудіопроцесор, який відповідає за приймання вхідного аудіосигналу, його послідовну обробку та передачу результату на вихідний аудіопристрій. Обробка сигналу виконується у методі processBlock, де аудіодані надходять у вигляді буфера семплів. До цього буфера послідовно застосовуються реалізовані звукові ефекти, зокрема noise gate, tremolo та distortion.

Кожен аудіоефект реалізовано як окремий програмний модуль. Такий підхід дозволяє незалежно розробляти, тестувати та змінювати окремі ефекти без необхідності суттєвої зміни всієї програми. Наприклад, модуль tremolo відповідає за періодичну зміну амплітуди сигналу, distortion — за нелінійне спотворення форми сигналу, а noise gate — за приглушення шумів нижче заданого порогу.

Графічний інтерфейс користувача винесено в окремий рівень програми. Він відповідає за відображення елементів керування, таких як слайдери, кнопки та підписи параметрів. Зміна параметрів користувачем передається до аудіопроцесора, після чого ці значення використовуються під час обробки аудіосигналу в реальному часі.

Таким чином, архітектура програми складається з трьох основних рівнів: рівня графічного інтерфейсу, рівня керування параметрами та рівня цифрової обробки сигналу. Такий підхід забезпечує зручність розробки, розширюваність системи та можливість додавання нових звукових ефектів у майбутньому.

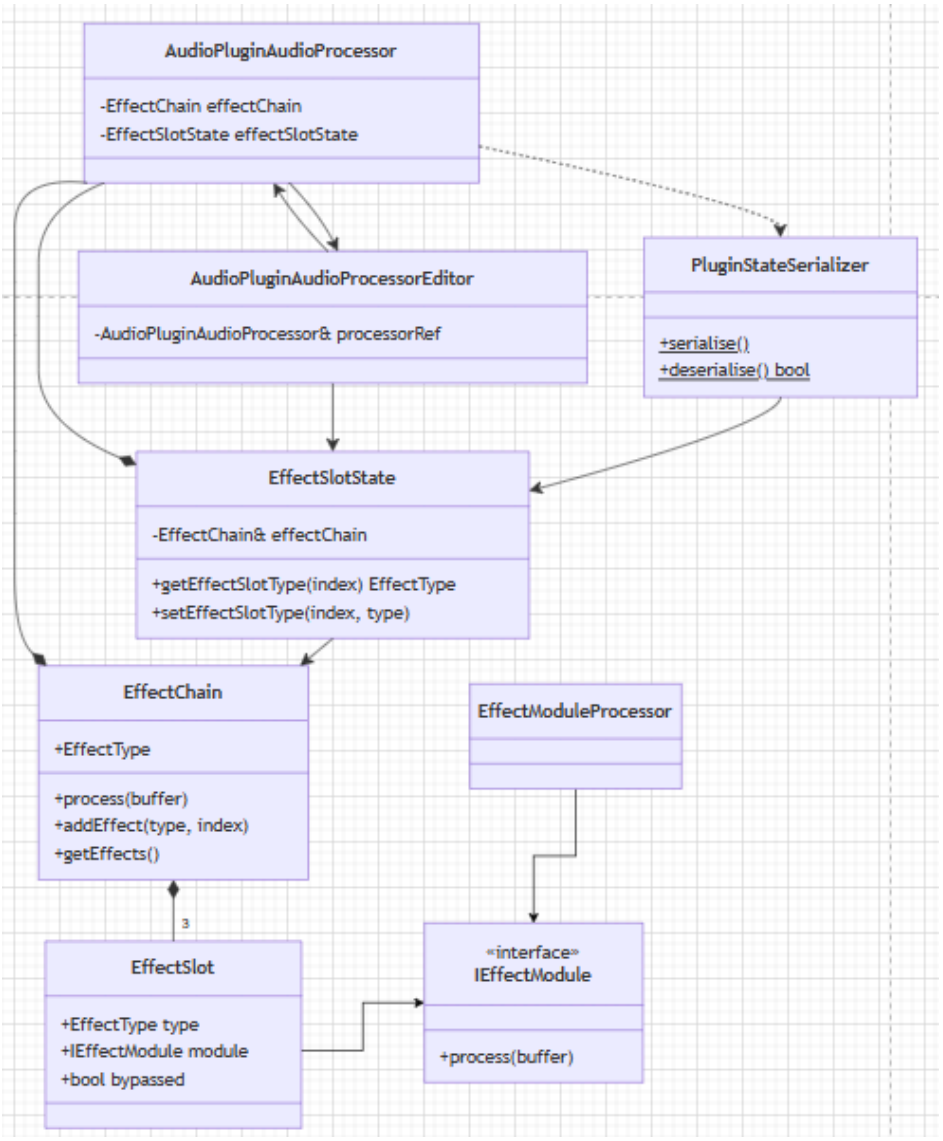


Рисунок 2.1 – Діаграма компонентів основного коду

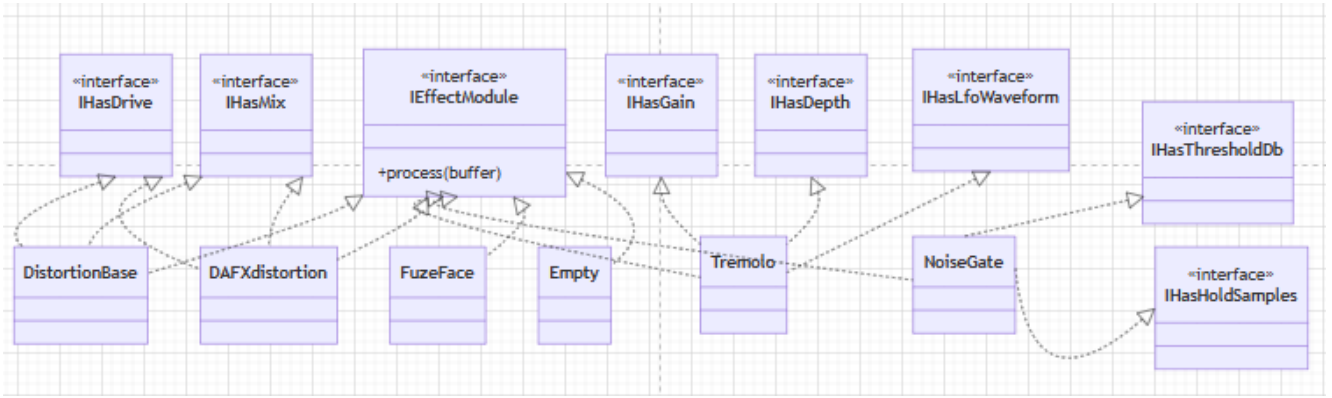


Рисунок 2.2 – Діаграма компонентів класів ефектів

З основних класів зараз я хочу виділити:

PluginProcessor – клас “ядро” всіх процесів. З цього класу починається вся програма. Тут створюються всі потрібні класи. Звідси передається аудіо буфер на обробку та йде підготовка до обробки.

EffectChain – в цьому класі зберігається черга ефектів. Клас передає буфер, отримує результат і передає далі. Саме завдяки цьому класу ми можемо мати більше одного ефекту за раз.

PluginEditor – це клас фреймворку в яку зберігається вся візуальна складова. Завдяки цьому класу користувачі можуть взаємодіяти з програмою.

Файлова структура проєкту побудована для полегшення знаходження потрібних файлів. Через архітектуру параметрів ефектів кількість файлів інтерфейсу засмічує голону папку. Тому було вирішено зробити наступну систему:

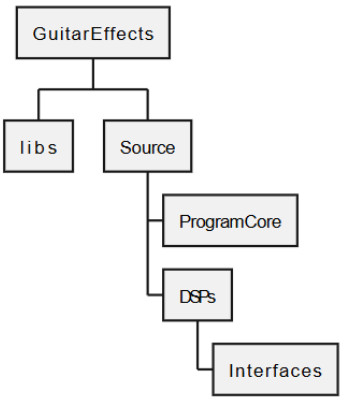


Рисунок 2.3 – Діаграма компонентів класів ефектів

Основна папка проєкту зберігає такі файли пов’язані з github або cmake. В

ній також є файли з використаними бібліотеками, та build файли cmake. Далі йде “Source”. В цій папці знаходиться основна частина програми. В корінні “Source” можно знайти файли створені самим фреймворком. Файли написаного мною коду знаходяться в “ProgramCore”, я вирішив відділити їх. Файли ефектів та їх інтерфейси знаходяться в “DSPs”.

Така система файлів дозволила мені не витрачати час на пошук потрібних файлів в системі.

Далі було створено діаграму використань. Основним актором є користувач, який взаємодіє з графічним інтерфейсом програми. Користувач може відкрити редактор параметрів, вибрати тип ефекту для певного слота ланцюга обробки та налаштувати параметри вибраного ефекту. Після вибору ефекту інтерфейс автоматично відображає доступні елементи керування відповідно до його типу.

Другим актором є хост-система (DAW або standalone-хост), яка забезпечує запуск застосунку, передавання аудіосигналу для обробки, а також збереження та відновлення стану плагіна. Під час роботи хост передає вхідний аудіосигнал до ланцюга ефектів, після чого отримує оброблений сигнал для подальшого відтворення або запису.

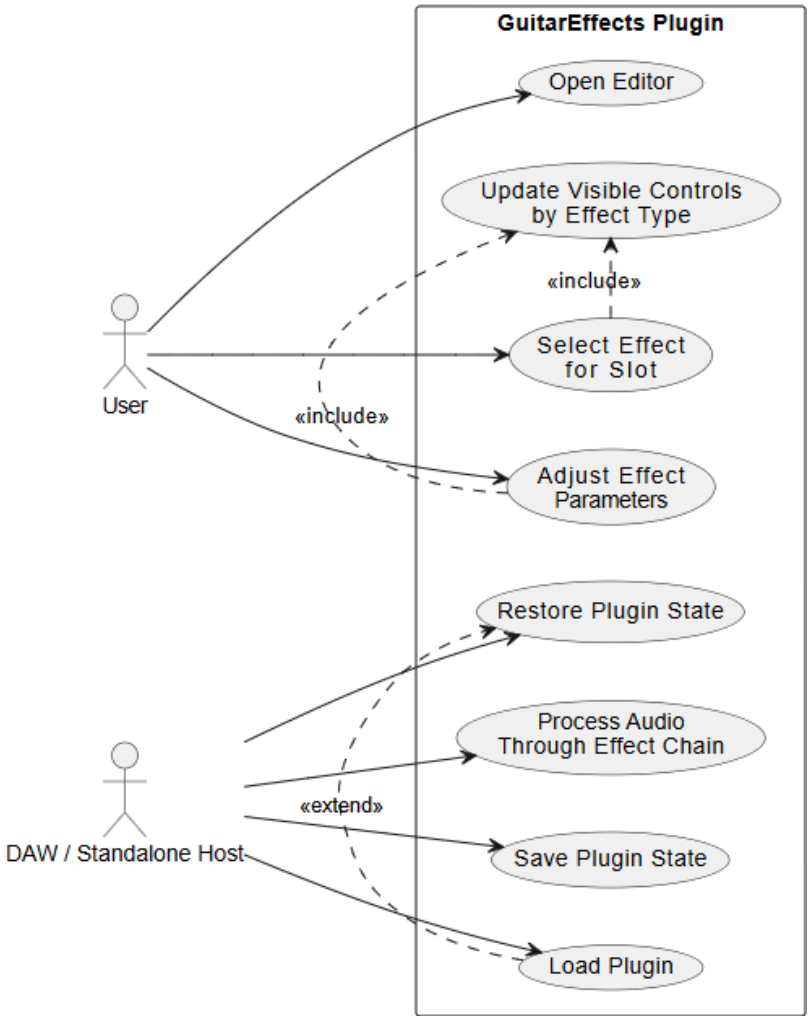


Рисунок 2.4 – Варіанти використання застосунку

## 2.3 Налаштування проєкту

Для початку роботи нам треба збудувати проєкт. Разом з фреймворком juce при завантаженні йде додаткова програма: Projucer. Ця програма націлена на швидку генерацію пустого проєкту з якого можно продожувати роботу. На жаль останні версії програми перестали підтримувати IDE Clion. Я не хотів використовувати підтримуваний visual studio через мій досвід з IDE від Jetbrains. Саме тому я почав шукати альтернативи.

Рішення було знайдено в офіційному github архіві фреймворку[14]. В архіві можно знайти же готові шаблони осноних файлів програми та готовий cmake файл. З цим же можно збудувати Cmake та запустити пустий шаблон проєкту.

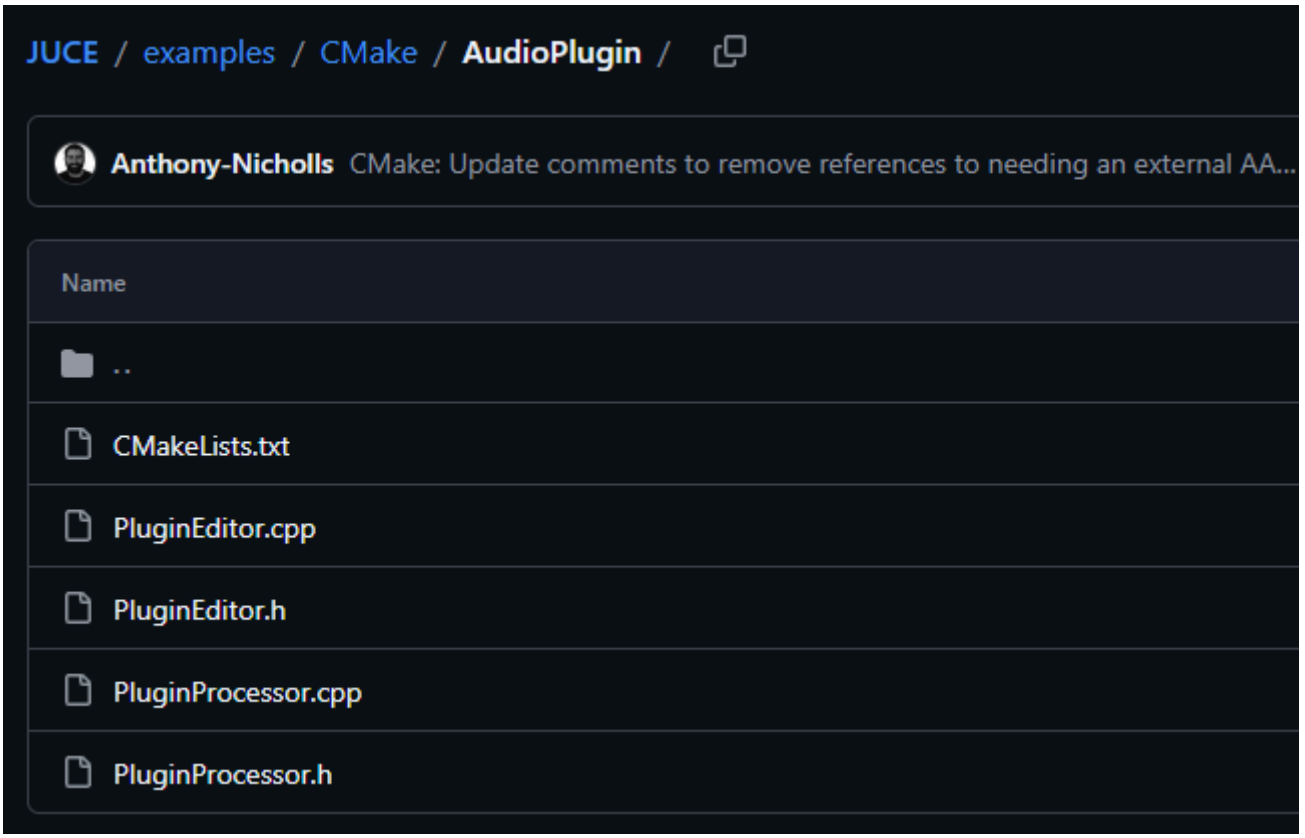


Рисунок 2.5 – Готові шаблони проєкту

Після створення github репозиторія для самого проєкту можно приступати до роботи.

### 2.4 Теорія DSP

Тепер для розуміння того як працюють ефекти я частково поясню теоретичні принципи DSP. Цифрова обробка сигналів (Digital Signal Processing, DSP) є однією з ключових технологій сучасної аудіоіндустрії. Вона використовується у музичному обладнанні, системах звукозапису, програмних аудіоплагінах, мультимедійних застосунках та телекомунікаційних системах. Основною метою DSP є аналіз, модифікація та покращення сигналів за допомогою математичних методів та алгоритмів, що виконуються цифровими пристроями.

Сигналом називають фізичну величину, яка змінюється в часі та несе певну інформацію. У випадку звуку сигнал являє собою зміну тиску повітря, що поширюється у вигляді хвиль. Коли людина грає на електрогітарі, коливання струн

перетворюються звукознімачами на електричний сигнал. Амплітуда цього сигналу змінюється відповідно до характеристик звукової хвилі та містить інформацію про висоту тону, гучність і тембр звуку.

На початковому етапі такий сигнал є аналоговим, тобто безперервним як у часі, так і за значенням амплітуди. Аналогові сигнали можуть безпосередньо передаватися через кабелі та оброблятися електронними схемами, проте для використання в комп'ютерних системах їх необхідно перетворити у цифрову форму. Цей процес здійснюється за допомогою аналого-цифрового перетворювача (ADC – Analog-to-Digital Converter).

Процес перетворення аналогового сигналу в цифровий складається з двох основних етапів: дискретизації та квантування. Під час дискретизації безперервний сигнал вимірюється через рівні проміжки часу. Кожне окреме вимірювання називається семплом (sample). Частота виконання таких вимірювань називається частотою дискретизації (sample rate) і вимірюється в герцах. У більшості аудіозастосунків використовуються частоти 44100 Гц або 48000 Гц, що означає отримання відповідно 44100 або 48000 семплів за одну секунду.

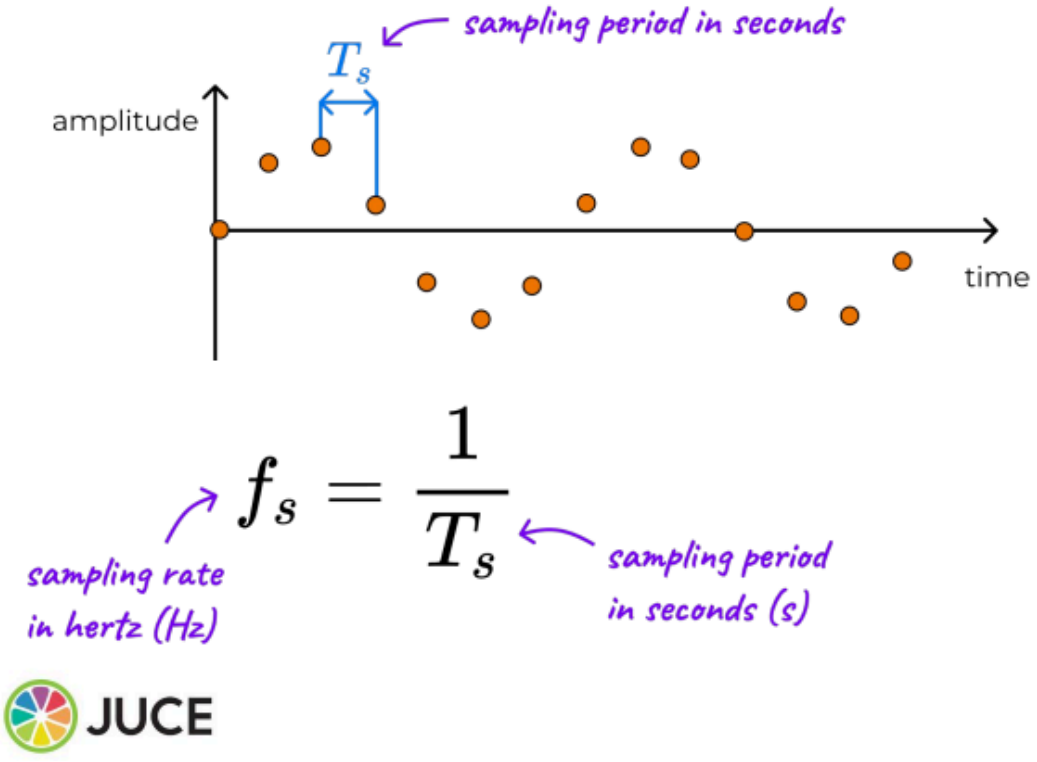


Рисунок 2.5 – Процес дискретизації (семплінгу)

Після дискретизації виконується квантування. На цьому етапі кожному семплу присвоюється певне числове значення, яке може бути представлене цифровою системою. Точність такого представлення визначається бітовою глибиною (bit depth). Наприклад, при використанні 16-бітного формату доступно 65536 можливих рівнів амплітуди сигналу, а при використанні 24-бітного формату їх кількість суттєво збільшується, що дозволяє точніше відтворювати динамічний діапазон звуку.

Отриманий набір семплів утворює цифрове представлення аудіосигналу. Саме з такими даними працюють сучасні аудіозастосунки та фреймворки, зокрема JUCE. Під час обробки аудіо семпли групуються у буфери (audio buffers), які містять певну кількість послідовних відліків сигналу. Обробка виконується над кожним буфером окремо, що забезпечує роботу системи в режимі реального часу.

Цифрова обробка аудіосигналів базується на математичних операціях над послідовністю семплів. До таких операцій належать зміна амплітуди сигналу, фільтрація, модуляція, додавання гармонік та інші методи.

Після завершення цифрової обробки аудіодані передаються до цифро-аналогового перетворювача (DAC – Digital-to-Analog Converter). Він виконує зворотне перетворення цифрового сигналу в аналоговий електричний сигнал, який може бути відтворений через підсилювачі, навушники або акустичні системи. Таким чином забезпечується повний цикл роботи аудіосистеми: від фізичних коливань струни до цифрової обробки сигналу та його подальшого відтворення у вигляді звуку[19].

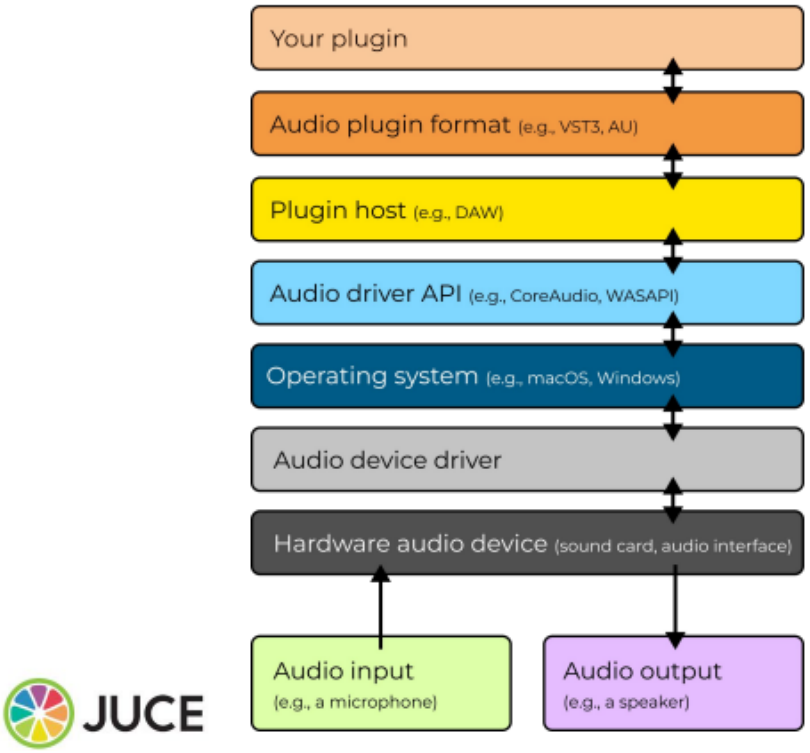


Рисунок 2.6 – Процес входу та виходу сигналу

Завдяки розвитку цифрової обробки сигналів сучасні програмні рішення здатні виконувати складну аудіообробку в реальному часі, забезпечуючи якість звучання, порівнянну з традиційними аналоговими пристроями. Саме тому технології DSP є основою більшості сучасних систем електронної симуляції гітарних підсилювачів та звукових ефектів.

## 2.5 Створення ефекту

Тепер ми можемо створити перший ефект. Вже готовий код дозволяє робити логіку алгоритму в “PluginProcessor”, але краще створити окремий файл для цього. Клас ефекту має два обов'язкових методи:

– prepare: цей метод викликається перед алгоритмом обробки. Його посилає основна частина перед самим аудіобуфером, щоб клас мав час на підготовку.

– process: це вже саме ядро процесора. Сюди ефект отримує аудіобуфер і саме тут знаходиться алгоритм.

– reset: цей метод викликається при видаленні класу.

Клас може мати більше методів. Але це залежить від складності ефекту. Далі я на прикладах покажу як вони працюють.

### 2.5.1 Distortion, викривлення

Дисторшн є доволі простим ефектом. Крім же описаних вище методів нам треба лише додати математичну формулу спотворення. Як вже було сказано Звукова хвиля в електронному форматі ділиться на безліч точок, кожна з яких має значення від -1 до 1. Саме взаємодіючи з цими значеннями ми впливаємо на звук[5].

Лістинг 2.1 - Код математичної формули викривлення

```
float fuzzExp(float clean, float gain)
    {
        float q = clean * gain;

        float sign = (q >= 0.f) ? 1.f : -1.f;
        float z = sign * (1.0f - std::exp(-std::abs(q)));

        return z;
    }
```

Тепер нам треба лише пройтись по аудіобуферу та змінити значення нашою формулою. Тепер варто пояснити структуру аудіобуфера. З розвитком технологій одного каналу звуку було замало, тому тепер програми мають працювати з двома, а то і більшою кількістю потоків. В кожному каналі одна точка семпла відповідає точці в тому самому місці в іншому каналі. Тому нам треба пройтися по кожному каналу та по кожному семплу в буфері. Порядок того, чи обрати один канал, обробити всі його семпли і йти до наступного каналу, чи обробляти обидва семпли за раз має велику різницю.

Лістинг 2.2 - Варіант 1: Спочатку канал, потім семпл (фрейм)

```
for (int channel = 0; channel < numChannels; ++channel)
{
    auto* data = buffer.getWritePointer(channel);

    for (int sample = 0; sample < numSamples; ++sample)
    {
        data[sample] = process(data[sample]);
    }
}
```

Тут, в лістингу 2.2, спочатку повністю обробляється перший канал, потім другий і так далі.

Перевага такого підходу полягає в тому, що дані кожного каналу в JUCE зберігаються безперервно в пам'яті. Процесор послідовно читає елементи масиву, що добре використовує кеш пам'яті та забезпечує високу продуктивність. Саме тому цей спосіб найчастіше застосовується для ефектів, які працюють незалежно з кожним каналом, наприклад distortion, noise gate або фільтрів.

Лістинг 2.3 - Варіант 2: Спочатку семпл (фрейм), потім канал

```
for (int sample = 0; sample < numSamples; ++sample)
{
    for (int channel = 0; channel < numChannels; ++channel)
    {
        auto value = buffer.getSample(channel, sample);
        buffer.setSample(channel, sample, process(value));
    }
}
```

Тут алгоритм спочатку обробляє певний момент часу для всіх каналів, а потім переходить до наступного семплу.

Такий підхід зручний тоді, коли необхідно одночасно аналізувати декілька

каналів. Наприклад, при реалізації стереоефектів, де результат залежить від значень лівого та правого каналів, або під час обчислення загальної гучності сигналу для всього аудіофрейму.

У випадку ефекту викривлення ми можемо використати обробку спочатку каналу а потім фрейму. Але для такого ефекту це різниці не має. Ефект викривлення є доволі легким, але і різноманітним. Основна ідея це “викривлення” аудіосигналу формулою яка робить значення семплу більшою за 1 або меншою за -1. В такому випадку значення “обрізається” і починає дорівнювати 1 або -1. Це і дає нам ефект викривлення. Але підняття значень поза межі дозволенного можно безліччю способів. Тому не дивно побачити два ефекти які називаються викривленням але звучать по різному. Навіть в моєму проєкті є два різних ефекти викривлення.

### 2.5.2 Тремоло

Наступний ефект створення якого я хочу розглянути це тремоло. Особливість цього ефекту в тому, що нам треба використати одну бібліотеку з фреймворку. Для симуляції ефекту підняття та опускання гучності сигналу ми можемо використати осцилятор. Бібліотеки фреймворку мають в собі можливість створення потрібного осцилятора. В цьому випадку використання варіанту проходження спочатку по канал не є доречною. модифікатор має бути однаковий для кожного фрейму. Тому якби ми використовували обхід каналу то нам треба було б зберігати дані в окремому масиві може бути небезпечно при real-time коді.

В осциляторі зазвичай використовується три основних видів хвиль:

– Sine wave (синусоїдальна)
– Triangle wave (трикутна)
– Square wave (прямокутна, квадратна)

При створенні генератора конструктор потребує формулу хвилі. В своєму ефекті я маю дві формули, одну для синусоїдної та трикутної хвилі.

Синусоїдальна хвиля (Sine) є найпоширенішою формою сигналу для

низькочастотної модуляції. Вона характеризується плавною та безперервною зміною значення без різких переходів або кутів. Завдяки цьому параметр, який модулюється, змінюється максимально природно, що забезпечує м'яке та музичне звучання ефекту Tremolo.

Трикутна хвиля (Triangle) також забезпечує плавну модуляцію, проте зміна її значення відбувається лінійно. На відміну від синусоїдальної хвилі, швидкість зміни параметра залишається сталою протягом більшої частини періоду. У результаті ефект набуває більш вираженого та ритмічного характеру звучання[5,18]. Тому трикутна хвиля часто використовується у випадках, коли необхідно отримати більш помітну та регулярну модуляцію без різких стрибків значення.

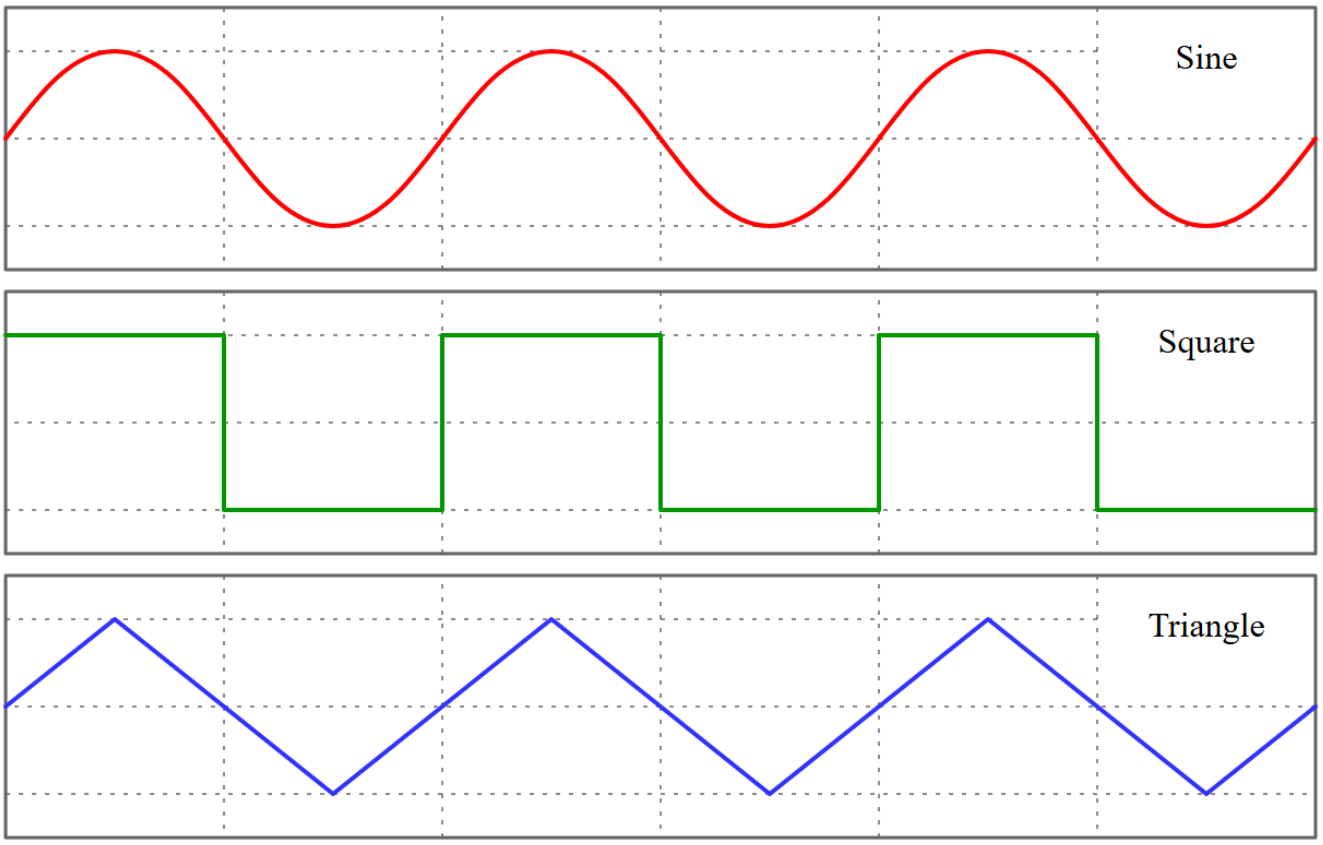


Рисунок 2.7 – Візуалізація синусоїдної, трикутної та квадратної хвилі

Лістинг 2.4 - Код осциллятора

```
static float triangle(float phase) {
    const auto fl = phase / juce::MathConstants<float>::twoPi;
```

```
    return 4.f * std::abs(fl-std::floor(fl+0.5f))-1.f;
  }
std::array<juce::dsp::Oscillator<float>, 2u> lfos{
           juce::dsp::Oscillator<float>{[](auto phase) { return
std::sin(phase);}},
    juce::dsp::Oscillator<float>{triangle}
  };
```

Тепер після створення осцилятора ми можемо почати розробляти алгоритм. Осцилятор має поступові коливання від 0 до 1. Тому помноживши це значення на значення семплу ми можемо керувати рівнем звуку. На основі значень LFO та Depth обчислюється коефіцієнт модуляції modulationValue. При нульовому значенні Depth амплітуда сигналу залишається незмінною, а зі збільшенням Depth вплив LFO на гучність стає більш вираженим.

Лістинг 2.5 - Код алгоритму тремоло

```
void process(juce::AudioBuffer<float>& buffer) override {
    updateLfos();
           for (const auto frameIndex : std::views::iota(0,
buffer.getNumSamples())) {

      const auto lfoValue = getNextLfoValue();  // [0, 1]
      const auto depth = modulationDepth.getNextValue();

          const auto modulationValue = (1.0f - depth) + depth *
lfoValue;
      const auto gain = gainSmoothed.getNextValue()
      for (const auto channelIndex :
           std::views::iota(0, buffer.getNumChannels())) {
            const auto inputSample = buffer.getSample(channelIndex,
frameIndex);

          const auto outputSample = inputSample * modulationValue *
gain;                              // OUTPUT SAMPLE
```

```
            buffer.setSample(channelIndex, frameIndex, outputSample);
                }
        }
    }
```

### 2.5.3 Особливості обробки аудіо в режимі реального часу

На відміну від багатьох інших програмних систем, застосунки цифрової обробки звуку працюють у режимі реального часу. Це означає, що кожен аудіобуфер повинен бути оброблений за строго обмежений проміжок часу. Якщо обробка не буде завершена до моменту, коли аудіосистема потребуватиме наступний буфер, користувач почує характерні звукові артефакти: потріскування, клацання, випадіння звуку або повне переривання відтворення.

З цієї причини код, який виконується в аудіопотоці, повинен відповідати вимогам Real-Time Safe програмування. Основним принципом є забезпечення гарантованого часу виконання алгоритму без непередбачуваних затримок.

Під час обробки аудіосигналу не рекомендується виконувати динамічне виділення пам'яті за допомогою операторів new, delete, а також контейнерів стандартної бібліотеки, які можуть автоматично змінювати свій розмір, наприклад std::vector::push_back(). Виділення пам'яті може призводити до звернення до операційної системи та викликати затримки невизначеної тривалості. З цієї причини необхідні буфери та структури даних зазвичай створюються під час ініціалізації ефекту в методі prepareToPlay(), а не під час виконання processBlock().

Також небажаним є використання файлових операцій, таких як читання або запис даних на диск. Швидкість роботи накопичувача не є гарантованою, а тому подібні операції можуть заблокувати виконання аудіопотоку на десятки або навіть сотні мілісекунд. Аналогічні обмеження стосуються мережевих запитів, доступу до баз даних та інших зовнішніх ресурсів.

Особливу увагу необхідно приділяти використанню механізмів синхронізації потоків. Виклики std::mutex, std::lock_guard, CriticalSection та інших блокуючих

примітивів можуть призвести до ситуації, коли аудіопотік буде змушений очікувати звільнення ресурсу іншим потоком. Навіть короткочасне блокування здатне викликати пропуски аудіобуферів та появу звукових артефактів. Саме тому в аудіопрограмуванні широко використовуються атомарні змінні (std::atomic) та lock-free структури даних.

Теоретично використання декількох потоків може прискорити виконання складних алгоритмів цифрової обробки сигналів. Однак на практиці такий підхід не завжди є доцільним для аудіоефектів реального часу. Створення нових потоків, перемикання контексту між ними та синхронізація результатів також потребують процесорного часу. Крім того, операційна система не гарантує точного моменту виконання кожного потоку, що може призводити до нестабільних затримок. Наприклад, якщо для обробки одного аудіобуфера використовується декілька потоків, аудіопотік буде змушений очікувати завершення роботи найповільнішого з них. У випадку перевантаження системи або зміни планування потоків операційною системою це може призвести до перевищення допустимого часу обробки буфера та виникнення звукових дефектів. Саме тому більшість сучасних аудіоефектів, зокрема distortion, tremolo, noise gate та інші ефекти гітарної обробки, виконують основну обробку сигналу в одному високопріоритетному аудіопотоці. Такий підхід забезпечує передбачуваний час виконання алгоритму та стабільну роботу системи навіть за низьких значень затримки (latency).

Таким чином, під час розробки аудіозастосунків необхідно враховувати специфіку роботи в режимі реального часу та уникати операцій, які можуть викликати непередбачувані затримки. Дотримання принципів Real-Time Safe програмування є однією з основних вимог до створення якісного програмного забезпечення для цифрової обробки звуку.

### 2.6 Створення параметрів

У розробленому застосунку параметри аудіоефектів реалізовано за модульним принципом. Для кожного типу параметра створюється окремий

інтерфейс, наприклад для параметрів Drive, Mix, Gain, Depth, LFO Waveform, Threshold або Hold. Якщо певний ефект підтримує конкретний параметр, він наслідує відповідний інтерфейс. Таким чином, наявність параметра визначається не назвою ефекту, а його можливостями.

Такий підхід дозволяє уникнути жорсткої прив’язки графічного інтерфейсу до конкретних ефектів. Наприклад, якщо декілька різних ефектів мають параметр Mix, немає потреби окремо створювати цей параметр для кожного ефекту. Достатньо один раз реалізувати інтерфейс відповідного параметра, після чого всі ефекти, які його наслідують, автоматично можуть використовувати однакову логіку зчитування, запису та відображення цього параметра.

У графічному інтерфейсі для кожного слота ефектів заздалегідь створюються елементи керування: слайдери, підписи та випадаючі списки. Після вибору ефекту система перевіряє, які параметри підтримує поточний ефект у конкретному слоті. Якщо ефект має відповідний параметр, елемент керування стає видимим і користувач може змінювати його значення. Якщо ефект не підтримує цей параметр, відповідний слайдер або список приховується.

Наприклад, ефекти спотворення можуть мати параметри Drive та Mix, ефект Tremolo може використовувати Gain, Depth та тип хвилі LFO, а Noise Gate — Threshold та Hold. Завдяки інтерфейсному підходу ці параметри не дублюються для кожного класу ефекту, а додаються до ефекту через наслідування потрібних інтерфейсів.

Лістинг 2.6 – Приклад інтерфейсу параметра

```
class IHasGain
{
public:
    virtual ~IHasGain() = default;

    virtual void setGain(float newDrive) = 0;
    virtual float getGain() const = 0;
};
```

Зміна значення параметра в інтерфейсі користувача передається до стану слота ефекту. Для цього в обробниках подій слайдерів викликаються відповідні методи встановлення значень, наприклад setDrive, setMix, setGain, setDepth, setThresholdDb або setHoldSamples. Після цього оновлені значення можуть використовуватися під час обробки аудіосигналу.

Також варто відмітити що пряма зміна деяких параметрів під час обробки може викликати аудіо дефекти. Це відбувається через різку зміну в параметрах та стрибок в звуковій хвилі яке може бути некоректно відторене комплектуючим користувача. Для цього ми “пом’якшуємо” зміну налаштувань. juce::SmoothedValue надає можливість не змінювати параметр одразу, а змінювати за деякий інтервал часу. Це має зменшити можливість появи аудіо дефектів.

Перевагою такого рішення є розширюваність архітектури. У разі додавання нового ефекту достатньо визначити, які параметри він підтримує, і реалізувати відповідні інтерфейси. Після цього графічний інтерфейс зможе автоматично відобразити потрібні елементи керування для цього ефекту. Це спрощує підтримку коду, зменшує дублювання та робить систему ефектів більш гнучкою.

### 2.7 Зберігання

У розробленому застосунку реалізовано механізм серіалізації та десеріалізації стану ланцюга ефектів. Його призначення полягає у збереженні вибраних ефектів, їх порядку в слотах та значень параметрів, щоб після повторного відкриття проєкту або застосунку користувач міг продовжити роботу з попередніми налаштуваннями. Для цього створено окремий клас PluginStateSerializer, який відповідає за перетворення поточного стану ефектів у JSON-структуру та відновлення стану з JSON-рядка. Такий підхід дозволяє відокремити логіку збереження від основної логіки аудіопроцесора, що покращує структуру коду та спрощує його підтримку. JSON був обраний через мою знайомість з цією мовою розмітки а також її читабельність для людини, що

полегшує ручне тестування застосунку.

Під час серіалізації для кожного слота ефекту створюється окремий об'єкт. У ньому зберігається тип ефекту та набір його параметрів. Тип ефекту перетворюється у текстовий токен, наприклад tremolo, noise_gate або distortion_base. Це робить збережений стан зрозумілим і незалежним від внутрішніх числових значень переліку EffectType.

Окремо формується об'єкт параметрів params. До нього додаються лише ті параметри, які підтримуються поточним ефектом. Наприклад, якщо ефект має параметр Drive, у JSON додається поле drive; якщо ефект підтримує Mix, додається поле mix; для Tremolo можуть зберігатися gain, depth та форма хвилі LFO, а для Noise Gate — thresholdDb і holdSamples. Такий підхід пов'язаний із модульною системою параметрів, де наявність параметра визначається можливостями конкретного ефекту. У кореневому об'єкті стану також зберігається номер версії формату. Це дозволяє в майбутньому змінювати структуру збережених даних і при цьому контролювати сумісність старих або нових версій стану. Якщо під час завантаження буде виявлено версію, яка є новішою за підтримувану програмою, десеріалізація не виконується.

Відновлення стану виконується у зворотному порядку. Спочатку JSON-рядок розбирається за допомогою засобів JUCE. Потім перевіряється наявність кореневого об'єкта, масиву слотів та підтримуваної версії. Перед завантаженням нового стану всі слоти скидаються у порожній стан. Після цього для кожного слота зчитується тип ефекту, встановлюється відповідний ефект у ланцюгу, а потім застосовуються його параметри. Важливо, що параметри застосовуються лише у тому випадку, якщо поточний ефект справді їх підтримує. Наприклад, значення drive буде встановлено тільки для ефекту, який має параметр Drive. Це захищає програму від некоректного стану та дозволяє безпечно працювати з різними типами ефектів.

Фреймворк JUCE вже має стандартний механізм збереження та відновлення стану аудіоплагіна. Для цього в класі AudioProcessor використовуються методи getStateInformation() та setStateInformation(). Перший метод викликається хостом

або standalone-застосунком тоді, коли необхідно зберегти стан плагіна. Другий метод викликається під час відновлення стану, наприклад при повторному відкритті проєкту в DAW або завантаженні збережених налаштувань. Таким чином, якщо виклик PluginStateSerializer::serialise() розмістити в методі getStateInformation(), а виклик PluginStateSerializer::deserialise() — у методі setStateInformation(), JUCE автоматично інтегрує цей механізм у життєвий цикл плагіна. Користувачу не потрібно вручну реалізовувати окрему систему збереження: хост або standalone-обгортка самостійно викликає відповідні методи у потрібний момент.

### 2.8 Слоти ефектів

Тепер варто пояснити як саме ми можемо обробляти аудіобуфер декількома ефектами за раз. У розробленому застосунку для цього було використано клас EffectChain. Його основне призначення полягає у зберіганні ланцюга аудіоефектів та послідовному застосуванні кожного ефекту до вхідного аудіобуфера. Кожен елемент ланцюга представлений структурою EffectSlot, яка містить тип ефекту, вказівник на об'єкт ефекту та прапорець bypass. Тип ефекту задається за допомогою переліку EffectType, де визначені доступні варіанти ефектів: distortionBase, tremolo, Empty та інші. Наявність типу Empty дозволяє використовувати порожній слот, який не змінює сигнал, але зберігає сталу структуру ланцюга.

Empty ефект є класом ефекту. Має однакові інтерфейси але алгоритм є “пустим”. Хоч алгоритм існує він не змінює дані аудіобуферу і повертає оригінальний звук. Цей “ефект” існує для полегшення розробки. Це дозволяє нам ставитись до “пустого” слоту  вмасивві як до звичайного ефекту, наприклад видалення іншого ефекту тепер є простою заміною одного ефекту на іншого.

Лістинг 2.7 – Алгоритм “пустого” ефекту

```
void process(juce::AudioBuffer<float>& buffer) override
```

```
    {
                for (const auto channelIndex : std::views::iota(0,
buffer.getNumChannels()))
        {
            auto* data = buffer.getWritePointer(channelIndex);

                    for (const auto frameIndex : std::views::iota(0,
buffer.getNumSamples()))
            {
                data[frameIndex] = data[frameIndex] ;
            }
        }
    }
```

Важливою особливістю реалізації є використання інтерфейсу IEffectModule. Усі ефекти, незалежно від їх конкретної реалізації, наслідують цей інтерфейс і мають спільний набір основних методів, таких як prepare, process та reset. Завдяки цьому клас EffectChain не залежить від внутрішньої логіки конкретного ефекту. Він працює з усіма ефектами однаково — через спільний інтерфейс. Це дозволяє зберігати різні типи ефектів в одному контейнері std::vector<EffectSlot>. Оскільки ефекти є об'єктами різних класів, для їх зберігання використовується std::unique_ptr<IEffectModule>. Такий підхід забезпечує поліморфізм: під час виконання програми викликається метод process саме того ефекту, який фактично знаходиться у слоті. Під час створення об'єкта EffectChain у конструкторі одразу додаються три порожні слоти. Це формує початковий ланцюг ефектів, який надалі може бути змінений користувачем через інтерфейс програми. Користувач може замінити порожній слот на конкретний ефект, наприклад Tremolo, Noise Gate або Distortion.

Важливо відмітити що багато функцій std::vector не є безпечними для використання в реальному часі. Саме тому всі дії які можуть викликати помилки при обмеженому часі виконуються до обробки аудіо. З цих причин також не можно змінити кількість слотів під час обробки. Але використання векторів

дозволяють це зробити в інший можливий час і загалом залишають можливість розширити максимальну кількість ефектів.

Метод prepare викликається перед початком обробки аудіо. Він зберігає поточні параметри аудіосистеми: частоту дискретизації, розмір аудіобуфера та кількість каналів. Після цього для кожного ефекту в ланцюгу викликається метод підготовки. Це необхідно для того, щоб кожен модуль ефекту міг налаштувати свої внутрішні змінні відповідно до поточних параметрів аудіопотоку.

Основна обробка аудіосигналу виконується в методі process. У цьому методі клас проходить по всіх слотах ланцюга ефектів. Якщо ефект не перебуває у стані bypass, для нього викликається метод process, у який передається аудіобуфер. Оскільки буфер передається за посиланням, кожен ефект змінює його безпосередньо. Після завершення роботи одного ефекту змінений буфер передається до наступного ефекту. Таким чином утворюється послідовний ланцюг обробки сигналу. Наприклад, якщо в ланцюгу знаходяться Noise Gate, Tremolo та Distortion, то спочатку буфер буде оброблений модулем Noise Gate, потім результат потрапить у Tremolo, а після цього — у Distortion. Порядок ефектів має важливе значення, оскільки кожен наступний ефект працює вже з результатом попередньої обробки.

Метод createEffect відповідає за створення конкретного об'єкта ефекту відповідно до вибраного типу. Для цього використовується конструкція switch, яка створює потрібний клас ефекту та повертає його як std::unique_ptr<IEffectModule>. У цьому ж методі для окремих ефектів задаються початкові значення параметрів, наприклад drive, mix, depth, gain, thresholdDb або holdSamples. Також тут можно визначити значення параметрів за замочуванням. Методи addEffect, removeEffect, moveEffect та setBypassed забезпечують керування ланцюгом ефектів. За їх допомогою можна додавати нові ефекти, замінювати ефект у певному слоті, видаляти ефекти, змінювати їх порядок або тимчасово вимикати обробку певного модуля без його видалення з ланцюга.

## 2.9 Висновки до другого розділу

У даному розділі було виконано проєктування та реалізацію програмного застосунку. На основі аналізу аналогів було сформовано функціональні та нефункціональні вимоги до застосунку, які визначили основні напрямки його розробки.

Було спроєктовано архітектуру системи з розділенням графічного інтерфейсу користувача, логіки керування параметрами та модуля цифрової обробки сигналів. Такий підхід дозволив забезпечити модульність, розширюваність та зручність подальшого супроводу програмного забезпечення. Для реалізації проєкту було обрано мову програмування C++ та фреймворк JUCE, який забезпечує роботу з аудіопристроями, обробку аудіосигналів у режимі реального часу та підтримку різних форматів аудіоплагінів.

У ході реалізації було розглянуто базові принципи цифрової обробки сигналів, особливості структури аудіобуферів та вимоги до програмування систем реального часу. Окрему увагу приділено питанням продуктивності та стабільності роботи аудіоефектів, а також обмеженням, які виникають під час виконання коду в аудіопотоці.

Було реалізовано набір аудіоефектів, систему їх послідовного об'єднання в ланцюг обробки, механізм керування параметрами на основі інтерфейсів та систему збереження стану застосунку за допомогою серіалізації. Запропоновані рішення дозволяють легко додавати нові ефекти та параметри без суттєвих змін у вже існуючому коді.

Таким чином, у результаті виконання другого розділу було створено програмну основу застосунку електронної симуляції електрогітарних звукових ефектів, яка забезпечує обробку аудіосигналу в режимі реального часу, налаштування параметрів ефектів та збереження конфігурації користувача. Отримані результати створюють основу для проведення тестування та оцінки працездатності розробленого програмного забезпечення у наступному розділі.

# 3 ТЕСТУВАННЯ

Для якісного проведення процесу тестування були розроблені та проведені функціональні тести [26] для кожного з програмних компонентів системи.

## 3.1 Тестування застосунку

Результати тестування представлені у таблиці 3.1.

Таблиця 3.1 - Функціональне тестування застосунку

| № | Назва тесту | Кроки | Очікуваний результат | Тест пройдено? |
|---|---|---|---|---|
| **1** | **2** | **3** | **4** | **5** |
| 1 | Запуск програми | 1. Відкриваємо exe файл програми | Програма успішно запускається | так |
| 2 | Додавання ефекту в слот | 1. Додаємо ефект через меню | Ефект з’являється в списку ефектів та вихідне аудіо змінюється | так |
| 3 | Додавання декілька ефектів | 1. Додаємо ефект через меню<br>2. Додаємо другий ефект через меню в другий слот | Ефекти з’являється в списку ефектів та вихідне аудіо змінюється поєднуючи ефекти | так |
| 4 | Додавання декілька ефектів в іншому порядку | 1. Додаємо перший ефект в слот два<br>2. Додаємо другий ефект через меню в слот один | Ефекти з’являється в списку ефектів та вихідне аудіо змінюється поєднуючи ефекти. Звучання має бути відмінним від тесту №3 | так |

Продовження таблиці 3.1

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| 5 | Перевірка ефекта тремоло | 1. Додаємо ефект тремоло | Отримане аудіо має виразний ефект тремоло | так |
| 6 | Перевірка ефекту noise gate | 1. Додаємо ефект noise gate | Вивід аудіо відсутній при звуках замалої гучності. Відсутні шуми | так |
| 7 | Перевірка змін параметрів | 1. Додаємо ефект тремоло<br>2. Переміщуємо повзунок параметра Depth | Аудіо помітно змінюється | так |
| 8 | Створення файлу збереження | 1. Додаємо довільну кількість ефектіів в довільному порядку<br>2. Відкриваємо меню застосунку<br>3. Натискаємо Save State | Програма має створити JSON файл в обраному місці | так |
| 9 | Збереження та завантаження стану при закритті та запуску програми | 1. Відкриваємо застосунок<br>2. Додаємо довільну кількість ефектіів в довільному порядку<br>3. Закриваємо застосунок<br>4. Відкриваємо застосунок | Ефекти та параметри мають мати ті самі характеристики що і при закритті | так |
| 10 | Різька зміна параметрів | 1. Обираємо distortion<br>2. Різко міняємо налаштування gain | Аудіо не має мати ніяких зайвих звуків | так |
| 11 | Вибір одного ефекту декілька разів | 1. Додаємо один ефект в слот один та дав | Ефекти з’являється в списку ефектів та вихідне аудіо має підсилений ефект двох однакових ефектів | так |

Продовження таблиці 3.1

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| 12 | Додавання максимальної кількості ефектів | 1. Додаємо ефекти в слот один, два та три | Програма обробляє буфер вчасно.Аудіо виводиться без виразних дефектів | так |
| 13 | Зміна пристрою вводу | 1. Заходимо в меню застосунку<br>2. Обираємо інший пристрій вводу | Пристрій успішно змінюється | так |

### 3.2 Висновки до третього розділу

У процесі тестування було перевірено основні функціональні можливості розробленого застосунку електронної симуляції електрогітарних звукових ефектів. Проведені тести охоплювали запуск програми, додавання та комбінування ефектів, зміну їх параметрів, перевірку роботи окремих ефектів, а також механізми збереження та відновлення стану застосунку. Крім того, було перевірено коректність роботи системи при різних сценаріях використання, включаючи додавання максимальної кількості ефектів та зміну пристроїв аудіовводу.

Отримані результати підтверджують працездатність та стабільність програмного забезпечення. Розроблений застосунок успішно виконує поставлені завдання з обробки гітарного аудіосигналу та може використовуватися як основа для подальшого розширення функціональності, зокрема шляхом додавання нових звукових ефектів, моделей підсилювачів і вдосконалення користувацького інтерфейсу.

# 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

Безпека життєдіяльності та охорона праці є важливою складовою професійної діяльності розробника програмного забезпечення.

## 4.1 Надзвичайні ситуації метеорологічного характеру

Надзвичайна ситуація – це порушення нормальних умов життєдіяльності населення на окремій території чи об’єкті, спричинене аварією, катастрофою, стихійним лихом або іншою небезпечною подією, що може призвести до людських жертв, матеріальних збитків та погіршення стану навколишнього середовища [8].

До надзвичайних ситуацій природного характеру належать геологічні, гідрологічні, метеорологічні, біологічні та інші небезпечні явища. Особливе місце серед них займають надзвичайні ситуації метеорологічного характеру, оскільки вони виникають досить часто та можуть охоплювати значні території [8, 9].

Надзвичайні ситуації метеорологічного характеру – це небезпечні атмосферні явища, які за інтенсивністю, тривалістю або масштабами поширення становлять загрозу життю та здоров’ю людей, завдають матеріальних збитків та порушують функціонування об’єктів господарювання [8, 9].

До основних небезпечних метеорологічних явищ належать:

– сильний вітер;

– шквали;

– буревії;

– урагани;

– смерчі;

– сильні дощі та зливи;

– сильні снігопади;

– хуртовини;

– град;

– ожеледь та ожеледиця;

– сильні морози;

– аномальна спека [9].

Сильний вітер є одним із найпоширеніших небезпечних явищ. Його дія призводить до пошкодження будівель, ліній електропередач, транспортних комунікацій, падіння дерев та рекламних конструкцій. При швидкості вітру понад 25 м/с значно зростає ризик руйнування покрівель і легких споруд.

Буревій являє собою вітер великої сили та тривалості, який може досягати швидкості понад 30 м/с. Наслідками буревіїв є руйнування інженерних споруд, порушення енергопостачання та транспортного сполучення.

Особливо небезпечним атмосферним явищем є смерч. Смерч утворюється внаслідок інтенсивного обертання повітряних мас і характеризується великою руйнівною силою. Під час проходження смерчу можуть руйнуватися будинки, пошкоджуватися транспортні засоби та інженерні мережі.

Значної шкоди завдають сильні дощі та зливи. Вони можуть спричиняти підтоплення територій, руйнування дорожнього покриття, порушення роботи систем водовідведення та електропостачання. Особливо небезпечними є тривалі опади, що призводять до паводків та підтоплення населених пунктів.

У зимовий період значну небезпеку становлять сильні снігопади та хуртовини. Внаслідок інтенсивних опадів ускладнюється рух транспорту, порушується робота комунальних служб та виникають перебої у функціонуванні об’єктів критичної інфраструктури.

Ожеледь та ожеледиця призводять до збільшення кількості дорожньо-транспортних пригод, травмування населення та обривів повітряних ліній електропередач. Значні відкладення льоду на проводах і конструкціях можуть викликати їх руйнування.

Град також належить до небезпечних атмосферних явищ. Великі градини можуть пошкоджувати покрівлі будівель, транспортні засоби, лінії електропередач та сільськогосподарські культури.

Основними наслідками надзвичайних ситуацій метеорологічного характеру є:

– загибель і травмування людей;

– руйнування житлових та виробничих будівель;

– пошкодження інженерних мереж;

– порушення транспортного сполучення;

– перебої у роботі систем електро-, тепло- та водопостачання;

– значні економічні збитки [8, 9].

Для зменшення негативних наслідків надзвичайних ситуацій метеорологічного характеру необхідно здійснювати постійний моніторинг погодних умов, своєчасно інформувати населення про можливу небезпеку та забезпечувати виконання заходів цивільного захисту [8].

Основними заходами захисту населення є:

– отримання інформації від органів цивільного захисту;

– обмеження перебування на відкритій місцевості під час небезпечних погодних явищ;

– укриття в капітальних будівлях;

– закріплення предметів, які можуть бути знесені вітром;

– створення резервів води, продуктів харчування та засобів автономного освітлення;

– дотримання правил безпеки під час ліквідації наслідків стихійних явищ.

Таким чином, надзвичайні ситуації метеорологічного характеру є одними з найпоширеніших природних небезпек. Їх своєчасне прогнозування та виконання профілактичних заходів дозволяє значно зменшити ризик для життя людей і матеріальних цінностей [8, 9].

### 4.2 Вимоги ергономіки до організації робочого місця оператора ПК

Під час розробки програмного забезпечення значна частина робочого часу проводиться за персональним комп’ютером. Тому важливого значення набуває

правильна організація робочого місця, яка забезпечує комфортні умови праці, зменшує втому працівника та сприяє підвищенню продуктивності праці [10, 11].

Ергономіка – це наука, яка вивчає взаємодію людини з технічними засобами та виробничим середовищем з метою створення безпечних і комфортних умов праці [11].

Під час виконання робіт, пов’язаних із програмуванням, тестуванням та налагодженням програмного забезпечення застосунку електронної симуляції електрогітарних звукових ефектів, оператор постійно взаємодіє з комп’ютером, монітором, клавіатурою, мишею та аудіообладнанням. Саме тому робоче місце повинно відповідати вимогам ергономіки.

Робочий стіл повинен забезпечувати достатню площу для розміщення обладнання та документації. Рекомендована висота робочої поверхні становить 680–800 мм. Конструкція столу повинна дозволяти вільне розташування ніг працівника [10, 11].

Робоче крісло має бути регульованим за висотою, кутом нахилу спинки та відстанню до робочого столу. Спинка крісла повинна підтримувати природне положення хребта та зменшувати статичне навантаження на м’язи спини [10, 11].

Монітор необхідно розташовувати таким чином, щоб верхня межа екрана знаходилася на рівні очей користувача або трохи нижче. Відстань від очей до екрана повинна становити 50–70 см. Таке розташування сприяє зменшенню навантаження на органи зору.

Клавіатура повинна розміщуватися на відстані 10–30 см від краю столу. Під час роботи передпліччя мають розташовуватися майже горизонтально, а кут у ліктьових суглобах повинен становити приблизно 90° [10, 11].

Важливим чинником комфортної роботи є освітлення. Недостатня освітленість або наявність відблисків на екрані призводять до швидкої втоми очей та зниження продуктивності праці [10, 11]. Перевагу слід надавати природному освітленню, а штучне освітлення повинно забезпечувати рівномірне освітлення приміщення.

Для забезпечення комфортних умов праці необхідно підтримувати оптимальні параметри мікроклімату. Температура повітря повинна становити 22–24 °С у холодний період року та 23–25 °С у теплий період. Відносна вологість повітря повинна знаходитися в межах 40–60 %.

Під час роботи за комп’ютером важливу роль відіграє режим праці та відпочинку. Тривале безперервне виконання робіт за монітором призводить до перевтоми та зниження концентрації уваги. Для запобігання негативним наслідкам рекомендується робити короткі перерви тривалістю 10–15 хвилин через кожні 1–2 години роботи [10, 11]. Особливістю розробки застосунку електронної симуляції електрогітарних звукових ефектів є необхідність використання додаткового аудіообладнання: навушників, аудіоінтерфейсу, акустичних систем та електрогітари. Під час роботи з таким обладнанням необхідно дотримуватися вимог електробезпеки, використовувати справні кабелі та уникати перевищення допустимого рівня гучності в навушниках [10, 12].

Раціональна організація робочого місця програміста сприяє зниженню втоми, підвищенню продуктивності праці та збереженню здоров’я працівника. Дотримання ергономічних вимог дозволяє забезпечити безпечні та комфортні умови праці під час розробки сучасного програмного забезпечення [10–12].

### 4.3. Висновки до четвертого розділу

У даному розділі було розглянуто питання безпеки життєдіяльності та охорони праці під час розробки програмного забезпечення застосунку електронної симуляції електрогітарних звукових ефектів. Проведено аналіз надзвичайних ситуацій метеорологічного характеру, їх основних причин виникнення, можливих наслідків та заходів щодо захисту населення. Встановлено, що своєчасне прогнозування небезпечних погодних явищ і виконання заходів цивільного захисту дозволяє суттєво знизити ризики для життя людей та матеріальних цінностей.

# ВИСНОВКИ

У ході виконання дипломної роботи було розроблено та протестовано програмне забезпечення застосунку електронної симуляції електрогітарних звукових ефектів з використанням фреймворку JUCE. Під час виконання роботи було проведено аналіз предметної області, розглянуто існуючі програмні рішення для обробки гітарного аудіосигналу, визначено основні функціональні та нефункціональні вимоги до системи, а також обрано технології для її реалізації.

На етапі проєктування було розроблено архітектуру застосунку, яка забезпечує розділення логіки цифрової обробки сигналу, графічного інтерфейсу користувача та механізмів збереження стану програми. Реалізована модульна структура дозволяє незалежно розробляти та розширювати окремі компоненти системи, що спрощує подальший розвиток програмного забезпечення.

У результаті реалізації було створено застосунок, який забезпечує обробку аудіосигналу в режимі реального часу та підтримує використання декількох звукових ефектів одночасно. До складу системи увійшли ефекти Tremolo, Distortion та Noise Gate з можливістю налаштування їх параметрів через графічний інтерфейс. Також реалізовано механізм збереження та відновлення конфігурації застосунку між сеансами роботи.

Проведене тестування підтвердило коректність роботи реалізованого функціоналу. Усі розроблені функції успішно пройшли перевірку, а застосунок продемонстрував стабільну роботу при різних сценаріях використання. Отримані результати свідчать про досягнення поставленої мети роботи та виконання визначених завдань.

Розроблений застосунок може використовуватися як основа для подальшого розвитку системи електронної симуляції гітарного обладнання. Перспективними напрямами вдосконалення є додавання нових звукових ефектів, реалізація симуляції гітарних підсилювачів і кабінетів, підтримка більшої кількості форматів плагінів та покращення користувацького інтерфейсу.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. AmpliTube. URL: https://www.ikmultimedia.com/products/amplitube5cs/ (дата звернення: 15.06.2026).

2. Дистриб'ютор плагінів Neural DSP. URL: https://neuraldsp.com/plugins (дата звернення: 15.06.2026).

3. Дистриб’ютор Guitar Rig 7. URL: https://www.native-instruments.com/en/products/komplete/guitar/guitar-rig-7-player/ (дата звернення: 15.06.2026).

4. Офіційний сайт JUCE. URL: https://juce.com/ (дата звернення: 30.04.2026).

5. Official JUCE Audio Plugin Development Course. URL: https://www.wolfsoundacademy.com/products/official-juce-audio-plugin-development-course (дата звернення: 30.04.2026).

6. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для здобувачів спеціальності 121 – Інженерія програмного забезпечення, всіх форм навчання / укладачі Михалик Д.М., Цуприк Г.Б., Бревус В.М. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2024. 45 с.

7. Udo Zölzer DAFX: Digital Audio Effects, John Wiley & Sons 2002, 553 pp.

8. Кодекс цивільного захисту України : Закон України від 02.10.2012 № 5403-VI. Поточна редакція. База даних «Законодавство України» / Верховна Рада України. URL: https://zakon.rada.gov.ua/laws/show/5403-17 (дата звернення: 07.06.2026).

9. ДСТУ 3891:2013. Безпека у надзвичайних ситуаціях. Терміни та визначення понять. Київ : ДП «УкрНДНЦ». URL: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=57361 (дата звернення: 07.06.2026).

10. НПАОП 0.00-7.15-18. Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями. Затверджено наказом Міністерства соціальної політики України від 14.02.2018 № 207. URL: https://zakon.rada.gov.ua/laws/show/z0508-18 (дата звернення: 07.06.2026).

11. Ткачук К. Н., Халімовський М. О., Зацарний В. В. та ін. Основи охорони праці : підручник 2-ге видання. Київ : Основа, 2014. 448 с.

12. Закон України «Про охорону праці» : Закон України від 14.10.1992 № 2694-XII. Поточна редакція. База даних «Законодавство України» / Верховна Рада України. URL: https://zakon.rada.gov.ua/laws/show/2694-12(дата звернення: 07.06.2026).

13. Желібо Є.П., Зацарний В.В. Безпека життєдіяльності. Підручник. – К.: Каравела, 2009.

14. Офіційний Github фреймворку JUCE. URL: https://github.com/juce-framework/JUCE (дата звернення: 15.06.2026).

15. Офіційна документація фреймворку JUCE. URL: https://docs.juce.com/master/index.html (дата звернення: 15.06.2026).

16. Офіційна сайт IDE Clion. URL: https://www.jetbrains.com/clion/ (дата звернення: 15.06.2026).

17. Functional and Non Functional Requirements. URL: https://www.geeksforgeeks.org/software-engineering/functional-vs-non-functional-requirements/ (дата звернення: 15.06.2026).

18. The Fourier Series. URL: https://lpsa.swarthmore.edu/Fourier/Series/WhyFS.html#Triangle_Wave (дата звернення: 15.06.2026).

19. Digital Signal Processing (DSP). URL: https://www.geeksforgeeks.org/electronics-engineering/what-is-digital-signal-processing/ (дата звернення: 15.06.2026).

20. Digital signal processor fundamentals and system design, CERN, Geneva, Switzerland 2008, 63 pp.

# ДОДАТКИ

# ДОДАТОК А

Тези конференції

*IX Міжнародна студентська науково - технічна конференція*
*"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"*

УДК 004.41
Іванков А. – ст. гр. СП-41
*Тернопільський національний технічний університет імені Івана Пулюя*

## ОПТИМІЗАЦІЯ ОБРОБКИ АУДІОСИГНАЛІВ У РЕАЛЬНОМУ ЧАСІ

Науковий керівник: PhD Стоянов Ю. М.

Ivankov A.
*Ternopil Ivan Puluj National Technical University*

## OPTIMIZATION OF REAL-TIME AUDIO SIGNAL PROCESSING

Supervisor: PhD Stoyanov Y. M.

Ключові слова: обробка аудіосигналів, реальний час, гітарні ефекти, цифрова обробка сигналів, оптимізація.

Keywords: audio signal processing, real-time systems, guitar effects, digital signal processing, optimization.

**Вступ**. Сучасні програмні рішення для обробки аудіосигналів широко використовуються у музичній індустрії, зокрема для емуляції електрогітарних підсилювачів та ефектів. Завдяки розвитку цифрової обробки сигналів (DSP) та обчислювальних ресурсів, програмні симулятори здатні забезпечувати якість звучання, близьку до аналогових пристроїв. Однак, обробка аудіо сигналів у реальному часі накладає жорсткі обмеження на продуктивність і затримки, що створює потребу в ефективних методах оптимізації[1].

Особливістю таких систем є необхідність обробки безперервного потоку аудіоданих із гарантованим часом відгуку. Будь-які затримки або перевищення часу обробки можуть призводити до появи шумів, спотворень або втрати аудіосигналу, що є критичним для музичних застосувань.

Мета роботи – дослідження та аналіз методів оптимізації обробки аудіосигналів у реальному часі для застосунків симуляції гітарних ефектів

**Основна частина**. Обробка аудіосигналів у реальному часу передбачає виконання обчислень із мінімальною затримкою (latency), що є критичним для інтерактивних аудіозастосунків. Основними вимогами є стабільність роботи, відсутність артефактів та ефективне використання ресурсів процесора[2].

Ключові аспекти оптимізації включають:

1. Буферизація аудіоданих. Обробка сигналу здійснюється блоками фіксованого розміру (audio buffers). Зменшення розміру буфера знижує затримку, але збільшує навантаження на процесор. Оптимальний вибір розміру буфера є компромісом між продуктивністю та якістю обробки.
2. Ефективна реалізація DSP-алгоритмів. Наприклад гітарні ефекти, такі як спотворення, затримка, реверберація, базуються на різних математичних моделях. Наприклад, ефект затримки реалізується через використання циклічних буферів, а реверберація – через згортку сигналу з імпульсною характеристикою. Оптимізація таких алгоритмів включає уникнення надлишкових обчислень та використання апроксимацій[3].

3. Мінімізація використання пам'яті та копіювання даних. Часті операції копіювання можуть суттєво впливати на продуктивність. використання посилань та in-place обробки дозволяє зменшити накладні витрати.

4. Паралелізація обчислень. Сучасні процесори мають декілька ядер, що дозволяють розділити обчислення між потоками. Проте у випадку аудіобробки важливо уникати болокувань і синхронізацій, які можуть викликати затримку або переривання звуку.

5. Використання можливостей фреймворку JUCE. Цей фреймворк надає оптимізаційні класи для роботи з аудіопотоками, буферами та плагінами (VST,AU). Правильне використання Audio Processor та AudioBuffer дозволяє забезпечити ефективну обробку сигналу у реальному часі.

Особливу увагу приділено нелінійним ефектам, таким як спотворення, які потребують обчислень складання функцій. Для оптимізації застосовуються табличні значення або спрощені математичні моделі.

Крім того, важливим аспектом уникнення звукових збоїв, які виникають при перевищенні часу обробки аудіобуфера. Це вимагає суворого дотримання обмежень реального часу та оптимізації кодного етапу обробки сигналу.

**Результати**. У результатах дослідження було визначено основні підходи до оптимізації обробки аудіосигналів у реальному часі. Їх застосування дозволяє:

- зменшення затримку обробку аудіосигналу;
- забезпечити стабільну роботу застосунків без аудіоартефактів;
- підвищити ефективність використання ресурсів процессора;

**Висновки**. У роботі досліджено особливості обробки аудіосигналів у реальному часі та визначено основні підходи до її оптимізації. Показано, що ефективність застосунків симуляції гітарних ефектів залежить від оптимізації DSP-алгоритмів, буферизації та використання ресурсів системи.

Список використаних джерел:

[1] U. Zölzer, Digital Audio Signal Processing, 3rd ed., Wiley, 2021.

[2] W. Pirkle, Designing Audio Effect Plug-Ins in C++, 2nd ed., Routledge, 2019.

[3] J. O. Smith, *Introduction to Digital Filters with Audio Applications*, W3K Publishing, 2007. [Online]. Available: https://ccrma.stanford.edu/~jos/filters/ (accessed Apr. 1, 2026).

**ДОДАТОК Б**

Посилання на репозиторій GitHub

https://github.com/DEDever/GuitarEffects