НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет електроніки
Акустичних та мультимедійних електронних систем

«На правах рукопису»

УДК 534.3::5+621.3

«До захисту допущено»

Завідувач кафедри
С. А. Найда

“10”червня 2022 р.

Дипломна робота
на здобуття ступеня бакалавра

зі спеціальності 171 Електроніка

на тему: Віртуальний синтазатор на базі VSTi

Виконав: студент 4 курсу, групи ДГ-81

Іващенко Ілля Андрійович _____

Керівник к.т.н. доц. Богданов О. В. _____

Рецензент доц. каф. ЕПС ФЕЛ к.т.н. Волківський В.Б. _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ — 2022 року

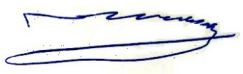
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет електроніки

Кафедра акустичних та мультимедійних електронних систем

Рівень вищої освіти — перший (бакалаврський)

Спеціальність 171 Електроніка

ЗАТВЕРДЖУЮ
Завідувач кафедри
 С. А. Найда

«02» травня 2022 р.

ЗАВДАННЯ
на дипломну роботу студенту
Іващенко Іллі Андрійовичу

1. Тема роботи Віртуальний синтезатор на базі VSTi
керівник роботи к.т.н. доц. Богданов Олексій Вікторович,
затверджені наказом по університету від 06.06.2022р. № 911-с

2. Строк подання студентом роботи 10.06.2022р.

3. Вихідні дані до роботи VST та VSTi, сучасні програмні синтезатори, ADSR огинаюча.

4. Перелік графічного (ілюстративного) матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо) електронна презентація.

6. Дата видачі завдання 02 травня 2022 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Строк виконання етапів роботи	Примітка
1	Огляд робіт за темою	02.05.22 — 16.05.22	Виконано
2	Методи та засоби	17.05.22 — 29.05.22	Виконано
3	Отримані результати	30.05.22 — 03.06.22	Виконано
4	Аналіз результатів	04.06.22 — 07.06.22	Виконано
5	Оформлення роботи	08.06.22 — 09.06.22	Виконано
6			

Студент _____ І. А. Іващенко
(підпис)

Керівник роботи _____ О. В. Богданов
(підпис)

ЗМІСТ

РЕФЕРАТ 6

ABSTRACT.....7

ПЕРЛІК СКОРОЧЕНЬ 8

ВСТУП.....9

1. ОГЛЯД РОБІТ ЗА ТЕМОЮ 11

1.1. СИНТЕЗ ЗВУКУ.....11

1.2. МЕТОДИ СИНТЕЗУ ЗВУКУ.....12

1.3. БУДОВА СИНТЕЗАТОРА.....20

ВИСНОВОК ДО РОЗДІЛУ 23

2. МЕТОДИ ТА ЗАСОБИ 24

2.1. АНАЛІЗ ГОТОВИХ РІШЕНЬ.....24

2.2. АНАЛІЗ МЕТОДІВ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ 29

ВИСНОВОК ДО РОЗДІЛУ 32

3. ОТРИМАНІ РЕЗУЛЬТАТИ.....33

3.1. ПРОГРАМНА РЕАЛІЗАЦІЯ.....33

3.2. АПРОБАЦІЯ РОБОТИ СИНТЕЗАТОРА 36

ВИСНОВОК ДО РОЗДІЛУ 43

4. ОБГОВОРЕННЯ РЕЗУЛЬТАТІВ 44

4.1. АНАЛІЗ ХАРАКТЕРИСТИК СИНТЕЗАТОРА.....44

4.2. ПОРІВНЯННЯ З МОДУЛЬНИМ АНАЛОГОМ.....45

4.3. ГАЛУЗІ ЗАСТОСУВАННЯ ТА ШЛЯХИ РОЗВИТКУ.....46

ВИСНОВОК ДО РОЗДІЛУ 48

ВИСНОВОК.....49

Перелік посилань 50

Додатки 54

Додаток А. Скрипт осцилятора.....55

Додаток Б. Скрипт процесора.....57

Додаток В. Скрипт синтезаторного голосу 63

Додаток Г. Скрипт огинаючої.....66

Додаток Д. Скрипт фільтра 67

Додаток Е. Скрипт графічного інтерфейсу 69

РЕФЕРАТ

Іващенко, І. А. Віртуальний синтезатор на базі VSTi : дипломна робота на здобуття ступеня бакалавра : 171 – Електроніка. – Київ, КПІ ім. Ігоря Сікорського, 2022. – 77 с.

Була поставлена задача розробити простий, доступний і дружній до початківців програмний синтезатор, якій міг би розширити коло зацікавлених музикою людей. А також значно знизити поріг входження до синтезу звуку.

У даній дипломній роботі було розроблено віртуальний синтезатор на базі VSTi. Синтезатор було реалізовано на мові C++ з використанням фреймворку JUCE. На протязі роботи були освоєнні всі базові поняття синтезу, які необхідні для розуміння теми.

Розроблений синтезатор було порівняно по можливостям з модульним аналогом і визначено, що розробка не поступається за характеристиками. Результат роботи задовольнив усім вимогам. У майбутньому синтезатор може покращуватися до нескінченності у тому числі сторонніми розробниками, адже проект було реалізовано у форматі open-source. Деякі можливі варіанти подальшого розвитку також були розглянуті у роботі.

Ключові слова: аудіо бази даних; аудіо інтерфейси користувача; частотна модуляція; синтез сигналів; генератори сигналів; фільтри; осцилятори; акустична інженерія; акустичні застосунки; акустичні хвилі; музика; обробка акустичних сигналів.

ABSTRACT

Ivashchenko, I.A. Virtual Synthesizer Based on VSTi : Bachelor Thesis : 171 – Electronics. – Kyiv, National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”, 2022. – 77 p.

The task was to develop a simple, accessible, and beginner-friendly software synthesizer that could expand the circle of people interested in music. And also significantly reduce the threshold for entry into sound synthesis.

In this thesis, a virtual synthesizer based on VSTi was developed. The synthesizer was implemented in C ++ using the JUCE framework. During the work, all the basic concepts of synthesis that are necessary to understand the topic were mastered.

The developed synthesizer was compared in terms of capabilities with a modular analog synthesizer, and it was determined that the development is not inferior in characteristics. The result of the work met all the requirements. In the future, the synthesizer may be improved indefinitely, including by third-party developers, as the project was implemented in an open-source format. Some possible options for further development were also considered in the paper.

Key words: audio databases; frequency modulation; signal synthesis; signal generators; filters; oscillators; acoustical engineering; acoustic applications; acoustic waves; music; acoustic signal processing.

ПЕРЛІК СКОРОЧЕНЬ

SSA — Sound Synthesis Algorithm

FM — Frequency Modulation

RMI — Rocky Mount Instruments

VCO — Voltage Controlled Oscillator

LFO — Low-Frequency Oscillator

ADSR — Attack Decay Sustain Release

VCF — Voltage Controlled Filter

MIDI - Musical Instrument Digital Interface

DAW – Digital Audio Workstation

VST — Virtual Studio Technology

GUI — Graphical User Interface

ООП – Об’єктно Орієнтовне Програмування

АЧХ – Амплітудно-Частотна Характеристика

ФВЧ – Фільтр Верхніх Частот

ВСТУП

У минулому музичні виконавці та гурти використовували для запису своїх композицій лише ті інструменти, які вони могли собі дозволити. Неважко зробити висновок, що жанр, у якому буде працювати митець, багато в чому залежав від того, скільки грошей він мав. Чи можна тоді вважати, що розвиток музичної індустрії, популярність окремих жанрів і поява нових в деякому сенсі залежить від економічної ситуації в окремих країнах й у світі в цілому?

У 1968 році Венді Карлос випустила свій перший студійний альбом «Switched-On Bach», який складався з декількох композицій Баха, повністю переписаних на синтезаторі Moog Modular. Ця подія дає зрозуміти, що музичний світ стрімко змінюється й темпи розвитку тільки набирають обертів, бо кожного року з’являються нові жанри й напрямки.

Отже, лише за останні 100 років музична індустрія пройшла величезний шлях: від митця, який напряму залежав від своїх статків, до людини, яка створює музичні композиції будь-якого жанру, маючи у своєму розпорядженні лише ноутбук, навушники або гучномовці. Зараз нові жанри з’являються кожного місяця, а нові методи й інструменти для створення своєї музики — кожного дня.

Синтетичний звук став популярним у 80-х не тільки, як елемент музичної композиції. Синтезатори використовувались усюди від кіно, до реклами. Зараз ситуація та сама, «класичний» синтетичний звук виходить у топи світових музичних чартів, активно використовується у кіноіндустрії, у гейм-індустрії, багато популярних мультсеріалів використовують у якості фону тільки синтетичний звук. Отже можна бути впевненим, що робота не втратить актуальності ані зараз, ані у майбутньому.

Ціллю даної роботи є створення синтезатора, який дасть можливість великому колу музикантів, абсолютно безкоштовно отримати потужний

інструмент, який, окрім нових, цікавих звуків у їх композиціях, буде додавати актуальності.

1. ОГЛЯД РОБІТ ЗА ТЕМОЮ

1.1. *СИНТЕЗ ЗВУКУ*

1.1.1. Алгоритми синтезу звуку (SSA - Sound Synthesis Algorithm)

Алгоритм синтезу звуку (SSA) — це алгоритм, який можна реалізувати в цифровому форматі для створення цифрових звукових зразків. SSA може бути розроблений для імітації якогось природного звуку, або створити якісь нові звуки з певними бажаними характеристиками та тембром. Також, звичайним є використання SSA як моделі систем реального світу для дослідження та виведення інформації про них.

Дизайн SSA є дуже складною проблемою. Зазвичай це вимагає ретельного аналізу бажаних характеристик кінцевого звуку та гарного уявлення про доступне моделювання техніки. Людська інтуїція є основним інструментом, який використовується для цих проектів. Деякі з кроків у проектуванні SSA було успішно автоматизовано. Horner et al. використовували генетичні алгоритми для автоматизації оцінки параметрів для FM синтезаторів, що моделюють трубу, тенор та гітарні тони [1].

1.1.2. SSA як топологія

Звичайною практикою є представлення (і зображення) SSA як діаграми топології взаємопов'язаних функціональних блоків [2]. Для адресації використовується термін топологія особливого розташування блоків та їх з'єднання. Топологія та пов'язаний з нею SSA є еквівалентними уявленнями однієї і тієї ж моделі отже, зміна в одному з них буде впливати на інший. (Рисунок 1.1) [2]

$$x(t) = e\sin(a(c + \sin(f)))$$

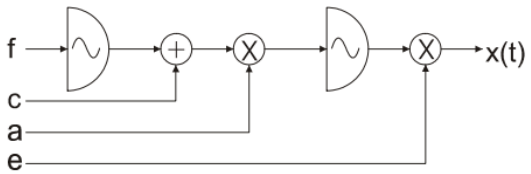


Рисунок 1.1 — Елементарний SSA і його топологія [2]

1.2. *МЕТОДИ СИНТЕЗУ ЗВУКУ*

1.2.1. FM синтез

FM (Frequency Modulation) — синтез аудіосигналу оснований на частотній модуляції. Тобто сигнал, який генерується одним осцилятором модулює сигнал іншого. При цьому сигнал першого також може керуватися вже третім осцилятором. В такий спосіб поєднуються будь-яка кількість генераторів і у будь-який спосіб, що дає можливість отримувати дуже багато нових тембрів.

FM-синтез дозволяє створювати різноманітні тембри, що варіюються від простих квазігармонічних сигналів до характерного «металевого» шуму [3]. Цифровий FM-синтез був основою різних музичних інструментів, починаючи з 1974 року. Під простим FM-синтезом розуміють частотну модуляцію вигляду:

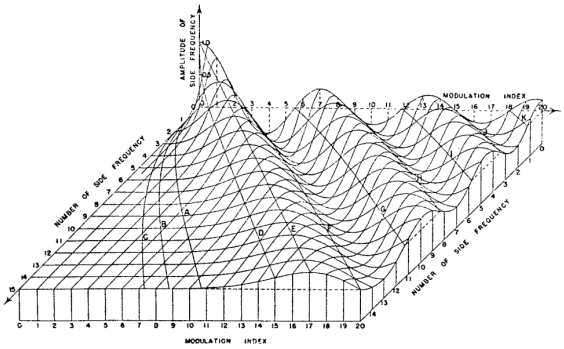
$$S_{FM}(t) = cos\ (\omega_c t + Bsin\omega_m t)$$

Перетворення Фур'є даного виразу дає:

$$F\big(S_{FM}(t)\big) = \sum_{n=-\infty}^{\infty} J_n(B)cos\ (\omega_c + n\omega_m)t$$

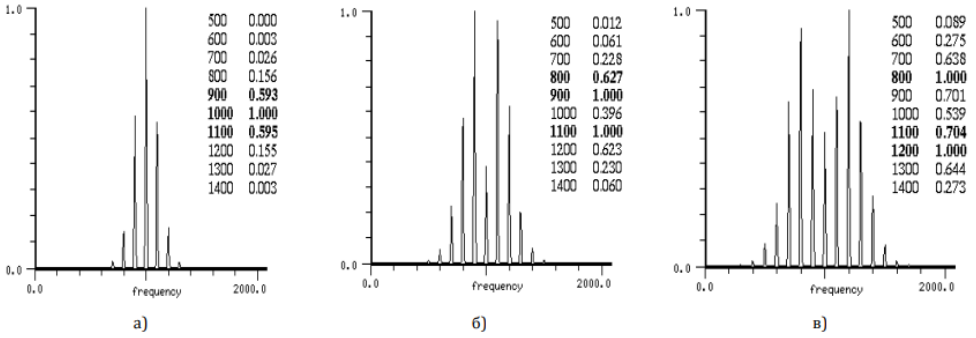
де $J_n(B)$ – функція Бесселя 1-го роду — рішення диференціального рівняння Бесселя, кінцеві у точці x = 0 при цілих та невід'ємних α. Амплітуди спектральних складових змінюються відповідно до функцій Бесселя. Їх значення близькі до нуля, доки індекс модуляції (B) не буд е дорівнює порядку (n), потім вони мають

різкий стрибок і слабшають подібно до синусоїди, що загасає пропорційно $1/x^{1/2}$ [3] (Рисунок 1.2).



[3]

Зі збільшенням індексу модуляції енергія спектру перерозподіляється у бік бічних смуг, що на слух сприймається як підвищення яскравості тембру (Рисунок 1.3).



[3]

Алгоритм цифрового FM-синтезу вперше був запропонований Д. Чоунінгом у Стенфордському університеті у 1967–1968 роках, ліцензований японською компанією Yamaha у 1973 році [4]. Найбільш відомою апаратною реалізацією FM-синтез є синтезатор Yamaha DX7, випущений в 1983 році. Yamaha зупинила випуск апаратних FM-синтезаторів на початку 90-х у зв'язку з переходом виробництва багатофункціональних робочих станцій. В даний час FM-синтез в основному реалізується в програмних синтезаторах, таких як Native

Instruments FM7/FM8, Image-Line Sytrus та ін. В той же час практично будь-який сучасний синтезатор має можливість частотної модуляції між парою генераторів.

1.2.2. Субтрактивний синтез

Наступна блок-схема (Рисунок 1.4) показує спектральну обробку, яка лежить в основі субтрактивного синтезу. На цій діаграмі осцилятор із фіксованою квадратною формою сигналу підключено до фільтру низьких частот. Дві властивості системи відкриті для динамічного керування: висота тону осцилятора і гранична частота фільтра. Не показано блоки постобробки для формування огинаючої амплітуди виходу фільтра [5].

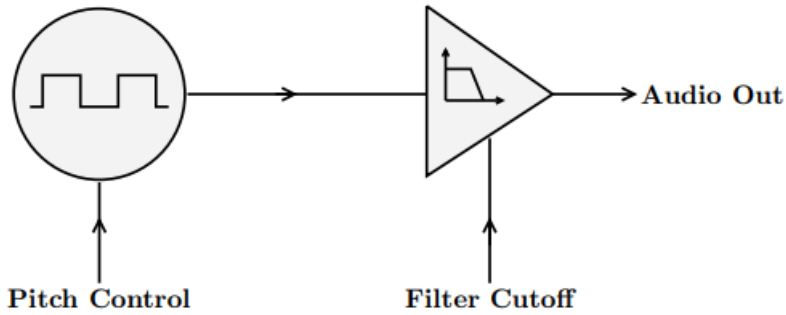
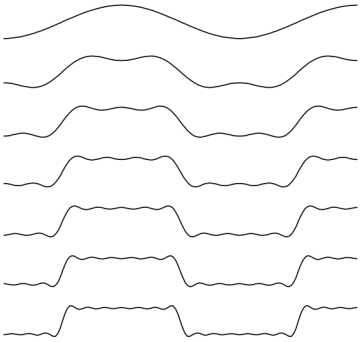


Рисунок 1.4 — Блок-схема простого субтрактивного синтезатора. [5]

Більшість аналогових синтезаторів 70-х років це монофонічні клавішні інструменти. Секція введення з клавіатури цих інструментів генерує аналогову напругу, яка кодується положенням натиснутої клавіші. У типовому синтезаторі цей сигнал керує висотою осцилятора і керуючим сигналом фільтра, так що спектр звуку задається на клавіатурою. Щоб «оживити» цей статичний звук, висота осцилятора та частота фільтра динамічно змінюються в залежності від базової лінії. Зміни можуть відбуватися за допомогою спеціалізованих схем (низькочастотні осцилятори, генератори огинаючої, що запускаються при зниженні ноти) та ручні контролери (колеса, тригери або джойстики, які генерують безперервний сигнал). Прямокутний осцилятор генерує сигнал, який можна описати наступною функцією:

$$square(p) = sin(2\pi r) + \frac{1}{3}sin(3 \cdot 2\pi r) + \frac{1}{5}sin(5 \cdot 2\pi r) + \cdots$$

На наведених графіках можна побачити, як члени ряду додаються, перетворюючись з синусоїди у хвилю більш квадратної форми (Рисунок 1.5) [5].



[5]

Ми можемо записати отриманий ряд більш компактним методом:

$$square(p) = \sum_{k=0}^{\infty} \frac{1}{2k+1} sin\ (2\pi(2k+1)p)$$

Найперші комерційно успішні клавішні синтезатори електронної музики використовували аналогові схеми для реалізації субтрактивного синтезу звуку. У 1970-х роках основними виробниками цих монофонічних інструментів були Moog, Arp, Emu та Roland, вони мали свій фірмовий звук, який залишається «на плаву» і донині [6].

1.2.3. Адитивний синтез

Адитивний синтез використовує додавання декількох хвиль синусоїдальної форми. На наступній схемі відображена проста схема роботи адитивного синтезатора, де на вході осциляторів подані частоти (f_k) і амплітуди (r_k) (Рисунок 1.6).

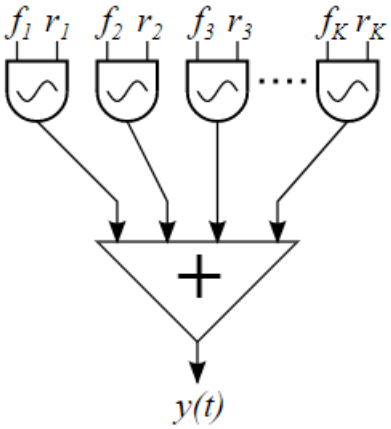


Рисунок 1.6 — Схема роботи адитивного синтезатора

Аналіз Фур’є — це методика, яка використовується для визначення цих точних параметрів тембру на основі загального звукового сигналу; і навпаки, отриманий набір частот і амплітуд називається рядом Фур'є вихідного звукового сигналу.

У випадку музичної ноти найнижча частота її тембру позначається як основна частота звуку. Для простоти ми часто говоримо, що нота грає на цій основній частоті [7], хоча звук цієї ноти також складається з багатьох інших частот. Набір решти частот називається обертонами (або гармоніками, якщо їх частоти ціло кратні основній частоті) звуку [8].

Іншими словами, лише основна частота відповідає за висоту ноти, тоді як обертони визначають тембр звуку. Обертони фортепіано, що грає середню до, будуть дуже відрізнятися від обертонів скрипки, яка грає ту саму ноту; саме це

дозволяє нам розрізняти звуки двох інструментів. Існують навіть незначні відмінності в тембрі між різними версіями одного і того ж інструменту (наприклад, фортепіано проти рояля).

Адитивний синтез має на меті використати цю властивість звуку, щоб створити тембр з нуля. Додаючи чисті частоти (синусоїди) різних частот і амплітуд, ми можемо точно визначити тембр звуку, який ми хочемо створити.

Ряд Фур'є періодичної функції математично виражається так:

$$y(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} [a_k cos(2\pi k f_0 t) - b_k cos(2\pi k f_0 t)]$$

$$y(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} r_k cos\,(2\pi k f_0 t + \phi_k)$$

де: $f_0 = 1/T$ фундаментальна частота, яка зворотньо пропорційна до періоду; $a_k = r_k \cos(\phi_k) = 2f_0 \int_0^P y(t) \cos(2\pi k f_0 t)\, dt,\ \ k \geq 0;$ $b_k = r_k \sin(\phi_k) = -2f_0 \int_0^P y(t) \sin(2\pi k f_0 t)\, dt,\ \ k \geq 1;$ $r_k = \sqrt{a_k^2 + b_k^2}$ амплітуда k-тої гармоніки $\phi_k = atan2(b_k, a_k)$ зміщення фази k-тої гармоніки

Найпростіший гармонічний адитивний синтез можна математично виразити так:

$$y(t) = \sum_{k=1}^{K} r_k cos\,(2\pi k f_0 t + \phi_k)$$

де K – результат синтезу, r_k – амплітуда, kf_0 – частота, ϕ_k – фаза зсуву відповідної k-тої гармоніки з К гармонік, f_0 – це основна частота.

Перший синтезатор, який реалізував адитивний синтез за допомогою цифрових осциляторів з’явився у 1974 році. Синтезатор також мав змінний у часі аналоговий фільтр. RMI (Rocky Mount Instruments) була дочірньою компанією Allen Organ Company, яка випустила перший комерційний цифровий церковний

орган, Allen Computer Organ, у 1971 році, використовуючи цифрову технологію, розроблену північноамериканським Rockwell.

У 1976 компанія Firlight випустила синтезатор Qasar M8, який використовував швидке перетворення Фур'є і дозволяв створювати семпли з інтерактивно намальованих амплітудних огинаючих гармонік (Рисунок 1.7).

Рисунок 1.7 — Qasar M8 [9]

1.2.4. Синтез шляхом фізичного моделювання

Наприклад гітарні інструменти є надзвичайно складними конструкціями, що складаються з набору струн, з'єднаних мостом із тілом, яке потім випромінює акустичну енергію до слухача. Лінійна поведінка тіла та характеристики випромінювання піддалися інтенсивним та чисельним дослідженням за

допомогою крокових методів [10]. Методи синтезу для моделі лінійних гітарних струн включають цифрові хвилеводи, які часто супроводжуються фільтром, що підсумовує вплив тіла та випромінювання [11].

Сильно нелінійна взаємодія зіткнення між струнами та грифом, особливо під дією зупинки або постукування пальцями, не була досліджена так ретельно, як характеристики тіла та випромінювання. Така нелінійна поведінка призводить до делікатного стукання та деренчання, особливо коли пальці рухаються. За простих щипкових і безперервних умов струни відскакують від піднятих ладів, що призведе до сильно залежних від амплітуди тембрів [10].

Динаміку зупинки пальців можна моделювати окремо. При наявності цієї динаміки можна імітувати зміни акордів, ковзаючі акорди барре, а також можливість грати гармоніки, коли пальці легко торкаються струни (Рисунок 1.8).

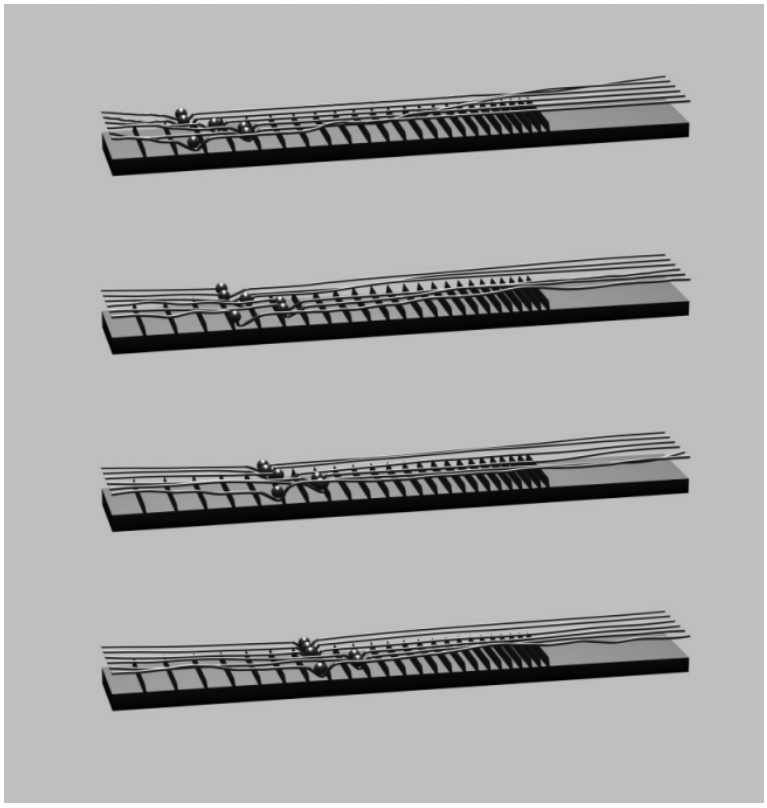


Рисунок 1.8 — Модель шестиструнної гітари під час рухів, що змінюються в часі, включаючи гриф і взаємодію пальців [11]

1.3. *БУДОВА СИНТЕЗАТОРА*

Звичайний середньостатистичний синтезатор має наступні елементи.

- VCO (Voltage Controlled Oscillator)
- LFO (Low Frequency Oscillator)
- ADSR (Attack Decay Sustain Release) огинаюча
- VCF (Voltage Controlled Filter)

1.3.1. VCO

VCO, тобто керований напругою генератор являє собою генератор, частота якого залежить від вхідної напруги. Прикладена вхідна напруга визначає миттєву частоту коливань. Отже, VCO можна використовувати для частотної модуляції (FM) або фазової модуляції (PM) шляхом подачі модулюючого сигналу на вхід керування. VCO також є невід'ємною частиною контуру фазової автосинхронізації. VCO використовуються в синтезаторах для створення форми хвилі, висота якої може регулюватися напругою, що визначається MIDI клавіатурою або іншим вхідним сигналом (рис 1.3.1.).

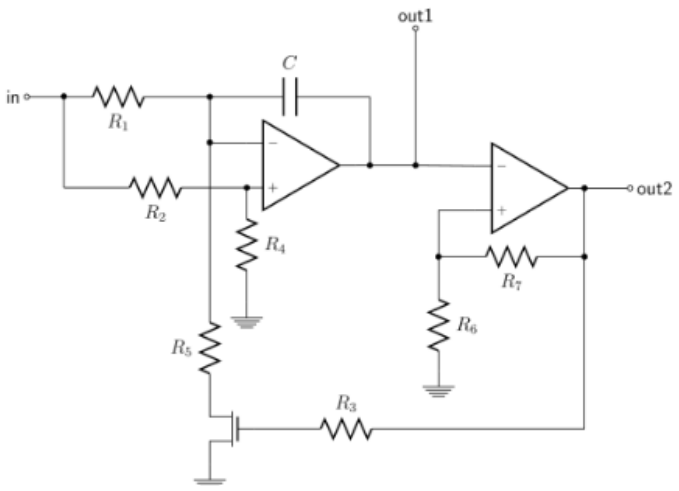


Рисунок 1.9 — Схема VCO [12]

VCO мають можливість вибору форми хвилі, зазвичай це квадрат, трикутник, синусоїда і пила, дещо рідше зустрічається можливість генерації шумового сигналу.

1.3.2. LFO

LFO — генератор низьких частот, тобто таких, які не чутні людському уху (нижче 20 Гц). Така особливість даного типу генераторів частот дає змогу змінювати властивості сигналу, який генерує VCO при цьому не створюючи зайвих звуків. Прикладом застосування LFO можуть буди тремоло та вібрато. При модуляції частоти основного генератора сигналу генератором низьких частот виникає ефект тремоло. При модуляції гучності основного генератора виникає ефект вібрато. Варто відмітити, що LFO може керувати будь якими параметрами, а отже має велику популярність, як складовий елемент звичайних синтезаторів, так і як окремий елемент модульного синтезатора.

1.3.3. ADSR

У звуці та музиці огинаюча описує, як звук змінюється в часі. Це може стосуватися таких елементів, як амплітуда (гучність), частоти (з використанням фільтрів) або висота. Наприклад, клавіша фортепіано при натисканні й утриманні створює майже миттєвий початковий звук, гучність якого поступово зменшується до нуля (Рисунок 1.10).

Параметрами ADSR-огинаючої є:

- Attack характеризує час, за який сигнал досягне своєї максимальної гучності з повної тиші.
- Decay характеризує час, за який сигнал повернеться на заданий параметром Sustain рівень гучності.
- Sustain задає рівень звуку на якому буде звучати сигнал.
- Release характеризує час, за який сигнал повернеться до повної тиші.

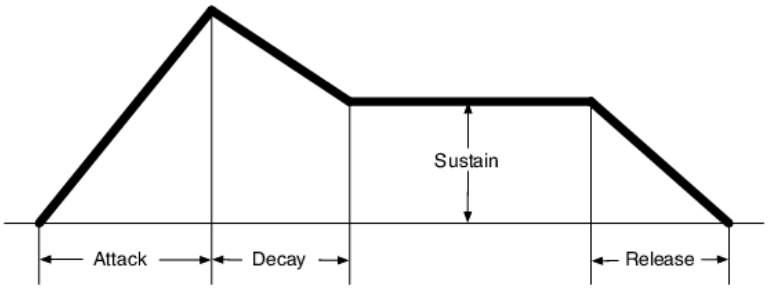


Рисунок 1.10. — ADSR огинаюча [13]

Фільтр, керований напругою - електронний фільтр, робочі характеристики якого (в першу чергу частота зрізу) можуть змінюватись вхідною керуючою напругою [14].

Як вже було сказано вище, здебільшого, несуча частота у таких фільтрах керує частотою зрізу, також зазвичай зустрічаються реалізації, у яких несуча частота керує резонансом (добротністю фільтра). Велика кількість VCF має можливість вибору між фільтром низьких частот, високих частот, смуговим фільтром та режекторним. Не зважаючи на назву, фільтри часто використовуються не для того, щоб прибрати зайві частоти, а для отримання нових характеристик гармонік на частоті зрізу (Рисунок 1.11).

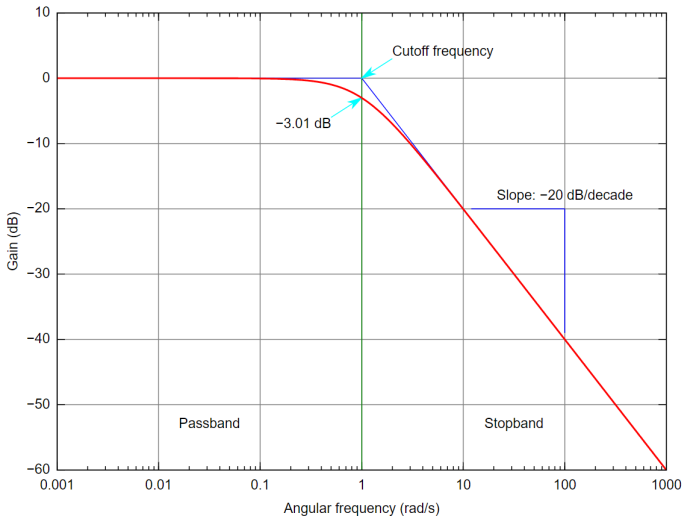


Рисунок 1.11 — Частота зрізу VCF, що показує перехідну характеристику Баттерворта [15]

ВИСНОВОК ДО РОЗДІЛУ

Отже у першому розділі було проведено пошукову роботу, у ході якої було визначено існуючі методи синтезу звуку, а також про їх будову. Розглянуто кожен обраний для висвітлення метод і механізм роботи кожної складової синтезатора з математичної точки зору, що дасть значну перевагу при виборі методу синтезу, який буде застосований мною при створенні віртуального синтезатора у ході виконання дипломної роботи, а також при виборі засобів реалізації розглянутих математичних моделей.

2. МЕТОДИ ТА ЗАСОБИ

2.1. *АНАЛІЗ ГОТОВИХ РІШЕНЬ*

2.1.1. KORG MS-20 V

Дана модель віртуального синтезатора є бездоганною емуляцією реального синтезатора компанії «KORG» розробленою інженерами з «Arturia» (Рисунок 2.1).

Рисунок 2.1 — KORG MS-20 V [16]

Розглядаючи особливості цього музичного інструмента можна виділити наступні характеристики [16]:

- Можливість як вбудованого рішення у якості плагіна у DAW (Digital Audio Workstation), так і окремо з MIDI (Musical Instrument Digital Interface) клавіатурою.
- Два окремих VCO, які окрім «класичних» форм генерованих хвиль мають ще режим генерації шумів, синхронізації та кільцевий режим.
- LFO модулятор з морфінгом форми хвилі.

- Два генератора огибаючої з режимом циклування і швидким режимом.
- Вбудований 3-канальний 12-кроковий секвенсор.
- 16 ефектів, та 4 слоти для них, які можна налаштувати послідовно, або паралельно.
- 6-ти голосна поліфонія з режимом унісону.
- Напівмодульність.

Отже, підсумовуючи можна сказати, що даний апарат має безмежні можливості і гнучкість. Але слід відмітити, що людина, яка вперше відкрила DAW і завантажила саме цей синтезатор може сприйняти інтерфейс дуже перевантаженим і складним, тобто даний апарат не для новачків. Також мінусом є велика вартість інструмента і не самі низькі вимоги до потужності ПК.

2.1.2. Plaits Macro-oscillator

Модуль VCO від компанії «Mutable Instruments». Постачається як у фізичному форматі, так і у якості безкоштовного плагіна для середовища VCV Rack 2 (Рисунок 2.2)

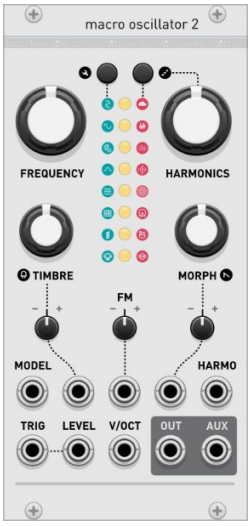


Рисунок 2.2 — Plaits Macro-oscillator 2 [17]

розглянутого вище. Але у порівнянні з іншими VCO даний модуль не поступається жодному аналогу.

Його характерною рисою є перемикачі вибору з восьми форм хвиль генерованого сигналу, та ще з восьми шумових та перкусійних сигналів. Також є такі важливі можливості [17]:

- Частотна модуляція.
- Налаштування морфінгу та тембру.
- Параметри «Model» і «Harmo», які дають можливість керувати тембром і морфінгом відповідно за допомогою інших вхідних сигналів.
- Вихід «Trig» за допомогою якого можна керувати огинаючою сигналу за допомогою іншого, керуючого сигналу.
- Вихід «V / OCT» за допомогою якого можна керувати частотою генерованого сигналу, за допомогою іншого вхідного сигналу.

Виходячи з розглянутих параметрів можна виділити, що даний інструмент є незамінним у модульному синтезі і знайде своє місце у будь-якому «патчі». З значних переваг підкреслю, що віртуальна версія Plaits є абсолютно безкоштовною і малого того, має відкритий код, що є дуже значною перевагою саме у рамках моєї роботи. Plaits має лише один мінус, це його абсолютна марність у якості окремого інструмента.

2.1.3. Spectrasonics Omnisphere

Віртуальний синтезатор «Omnisphere» від компанії «Spectrasonics» вже давно займає перші місця у звукорежисурі. Причиною цього є висока стабільність, постійні оновлення інструмента, та абсолютна кроссплатформенність (Рисунок 2.3).

Інструмент, який давно став стандартом має ряд переваг серед конкурентів, серед цих переваг бібліотека з 14000 готових пресетів, які є не тільки копіями класичних музичних інструментів, але й копіями інших існуючих синтезаторів, а також наступні переваги [18]:

- Функція «Sound Match», яка знаходить схожі звуки у великій бібліотеці пресетів.
- 58 ефектів, кожен з яких унікальний і має можливість індивідуального налаштування.
- Можливість використовувати сторонні аудіо, у якості пресета для гранулярного синтезу.
- Дуже гнучкі фільтри.
- 8 LFO та 12 огинаючих.

Omnisphere фактично позбавляє свого власника від необхідності користуватися будь-якими іншими продуктами. Але не зважаючи на свої плюси, даний інструмент має ціну в 499$, що буде великим мінусом для багатьох, також має високі вимоги до потужності ПК, 8 Гб, або вище оперативної пам’яті, та 64 Гб вільної пам’яті комп’ютера.

2.1.4. 3x Osc

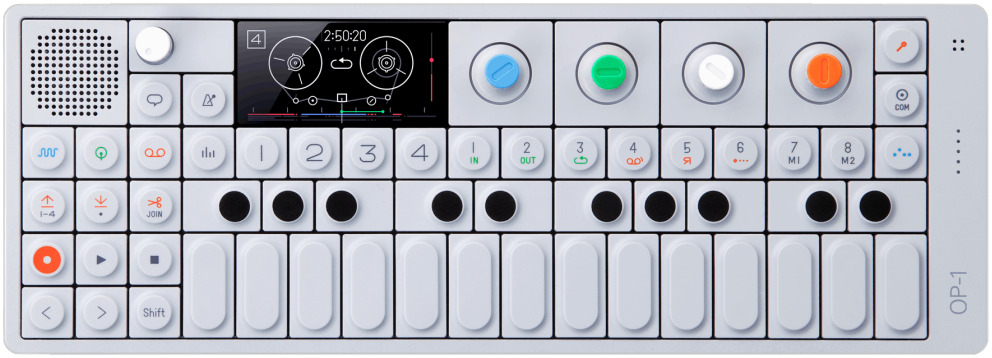
Цей базовий синтезатор в FL Studio це перше, що приходить на згадку, коли мова йде про класичний електронний звук. Він є незамінним, якщо вам треба прописати саб партію до вашого басу, або додати простих лідів у вашу доріжку (Рисунок 2.4)

[19]

Даний синтезатор використовує субтрактивний синтез. З 2000-го року він був використаний у понад одному мільйоні проектів [19]. Його особливістю є три осцилятора, які мають абсолютно однакові налаштування серед яких: грубість, м'якість, панорамування, вибір форми хвилі, налаштування фази, та детюна. Це всі налаштування, які має даний синтезатор. Його головна перевага — простота. Плюсом і одночасно мінусом цього синтезатора є те, що це базовий синтезатор у FL Studio, тобто вам не треба купувати його окремо, але в іншому середовищі скористатися їм не вдасться.

2.1.5. Teenage engineering OP-1

Портативний синтезатор від шведської компанії «Teenage engineering» є одним з тих інструментів, які привертають увагу більшого круга людей до індустрії синтезу звуку своїм сучасним зовнішнім виглядом і портативністю (Рисунок 2.5).



[20]

Можливості цього синтезатора можна перераховувати дуже довго. Секвенсор, семплер, MIDI клавіатура, синтезатор, мікшер та навіть пульт, для контролю світла, цей інструмент має можливість виконувати будь-які функції пов’язані з музикою. Чотири енкодери можуть бути переналаштовані під різні потреби і функції. Даний інструмент може працювати як окремо, з навушниками, моніторами, або вбудованим динаміком, так і у парі з ПК та зручним для вас DAW, або з мобільним пристроєм зі встановленим застосунком від Teenage engeneerig. З мінусів можна відмітити лише його високу вартість.

2.2. *АНАЛІЗ МЕТОДІВ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ*

2.2.1. VST.NET

VST.NET дає можливість роботи зі звуком використовуючи мову програмування C# [21]. Плюсом є те, що ми маємо прямий доступ до коду, а отже необмежені можливості у модернізації, та подальшому обслуговуванні створеного плагіна. Однак, остання версія VST.NET дає можливість створювати плагіни версії VST (Virtual Studio Technology) 2.4, у той час, як більшість виробників переходять до формату VST 3. Також варто відмітити, що C# є надбудовою C++, отже програмування на цій мові у будь-якому випадку буде повільнішою за програмування на C++.

2.2.2. Iplug2

Даний фреймворк працює з мовою C++, а отже вже має бал переваги над VST.NET. Має можливості створення плагінів у більшості сучасних форматів, а також створення застосунків для IOS [22]. Цей інструмент має досить велику бібліотеку класів, які дають можливість досить гнучкого програмування плагіну. Також даний інструмент дає великі можливості налаштування графічного інтерфейсу плагіну. З мінусів можна відмітити не дуже дружню і зручну документацію.

2.2.3. VST SDK

VST SDK є фреймворком від компанії Steinberg, яка в свою чергу створила такий формат, як VST [23]. Фреймворк, як і більшість конкурентів, працює з мовою C++, відрізняється даний інструментарій тим, що не має чіткої ієрархії класів і не дає таких можливостей, як інші, зі старту. Тобто багато чого треба прописувати саморуч. Також, оскільки фреймворк працює на низькому рівні програмування, ніяких можливостей для реалізації графічного інтерфейсу він не дає. В цілому можна підсумувати, що VST SDK є дуже потужним фреймворком, але не є дружнім до початківців і потребує не тільки базових знань у теорії звуку і базових знань програмування, але й деяких досить складних алгоритмів і форм мови C++.

2.2.4. JUCE

Перш за все варто відмітити, що даний фреймворк є одним з найпопулярніших у сфері аудіо програмування. Він дуже легко встановлюється, має зручний додаток, який дозволяє контролювати файли у яких безпосередньо відбувається створення плагіну, а також управляти бібліотеками та класами, які вам знадобляться, або ж не знадобляться у ході розробки.

JUCE має бездоганну та зручну документацію, яка дає змогу розібратися у механізмі роботи кожного наявного класу навіть новачку. Також на сайті

виробника є ряд уроків по програмуванню у фреймворці [24]. Даний плагін має певну ієрархію класів і відповідає усім принципам об’єктно орієнтовного програмування.

ВИСНОВОК ДО РОЗДІЛУ

У ході аналізу готових рішень, було виявлено велику складність більшості синтезаторів, а отже великий поріг входу нових людей у індустрії. У ході роботи хотілося би розробити синтезатор, який дасть можливість розширити коло користувачів, тобто не буде занадто складним у використанні. З іншого боку, залишити свій продукт відкритим для модернізації від інших розробників також є достатньо важливим критерієм, тому це було враховувано при виборі методів, та інструментів, які будуть використані у ході роботи.

Провівши пошукову роботу, було виділено декілька найбільш популярних фреймворків, які дозволяють розробляти VST плагіни. До розгляду не було взято нодові середовища розробки, адже хотілося би працювати безпосередньо з кодом, що дає більші можливості, та більшу гнучкість у наданні унікальності синтезатору.

Вимогами до фреймворку були сучасність, можливість роботи з GUI (Graphical User Interface), як вже було зазначено, хотілося б, щоб інші розробники мали можливість редагувати код, а отже фреймворк повинен бути відомим, відкритість коду, а також добре реалізована ієрархія класів. У ході аналізу визначено, що окрім всіх цих вимог, JUCE також відповідає основним принципам ООП, а отже буде мати високу стабільність і швидкість у роботі.

3. ОТРИМАНІ РЕЗУЛЬТАТИ

3.1. ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1.1. Осцилятор та частотна модуляція

Для кожного синтезатора найголовнішим і основним елементом є осцилятор. Маючи лише один генератор сигналу синусоїдальної форми вже можна отримати звук.

Оскільки, як було вказано вище, реалізовуватись синтезатор буде відповідно до деяких принципів ООП. Цей і всі наступні елементи, будуть створюватись відокремлено один від одного, на скільки це можливо, виносячи елемент в окремий клас і створюючи для нього дочірні методи.

Згідно з офіційною документацією JUCE [25], було визначено, що для того, щоб створити клас осцилятора, потрібно прописати у ньому декілька базових функцій і передати їм відповідні параметри.

Перш за все осцилятор повинен мати певну форму хвилі. У Додаток А. Скрипт осцилятора показано, що вибір форми хвилі було реалізовано за допомогою функції ініціалізації. Таким чином було реалізовано можливість вибору користувачем форми хвилі (синусоїдальна, пилкоподібна, міандр, або трикутник).

Всі змінні параметри, до яких має доступ користувач були внесені у дерево значень, як показано у Додаток Б. Скрипт процесора. Це дозволяє оновляти введені користувачем параметри у кожному окремому аудіоблоці, а також дає змогу зберегти принцип інкапсуляції, та при цьому без перешкод передавати всі необхідні параметри у потрібні функції. Так, в свою чергу, функція підготовки осцилятора отримує всі параметри, які використовуються у класі осцилятора.

І, нарешті, осцилятор повинен отримувати значення частоти, яку він буде генерувати, що реалізується відповідною функцією (Додаток А. Скрипт осцилятора), яка в свою чергу переводить отриманий MIDI сигнал у Герци.

Реалізація FM складової майже в точності копіює реалізацію самого осцилятора. Фактично, оператор частотної модуляції – це і є осцилятор, але його параметри не керуються MIDI сигналом, а мають статичну, обрану користувачем частоту, та глибину (амплітуду) коливань. Саме тому було прийняте рішення не виносити модулятор у окремий клас, а прописати функцію залежності основної частоти від частоти модулятора. У Додаток А. Скрипт осцилятора можна побачити, що для модулятора була обрана синусоїдальна форма хвилі.

Задля запобігання отримання від'ємної частоти, через занадто велику глибину хвилі яка модулює, було додано алгоритм, який перемножує частоту на -1 у тому випадку, якщо вона приймає від'ємне значення.

Окремим класом було винесено «синтезаторний голос», тобто звук, який отримується з кожним новим натисканням клавіши. Логічно, що у даному класі будується вся логістика між створеними елементами, та функціями. Як показано у Додаток В. Скрипт синтезаторного голосу, всі параметри з дерева значень потрапляють до класу осцилятора і назад саме тут.

3.1.2. ADSR огинаюча

Огинаючу сигналу також було реалізовано згідно до офіційної документації [26]. Даний елемент реалізований у JUCE таким чином:

- Обрані користувачем параметри, а саме час за який сигнал дійде до максимального значення, час за який сигнал повернеться до встановленого рівня сигналу, рівень сигналу і час затухання, передаються у дерево значень.
- З дерева значень вони потрапляють у голос синтезатора, де за допомогою функції «applyEnvelopeToBuffer» огинаюча накладається на аудіобуфер.
- Аудіоблок повертається у процесор.

Рівень сигналу регулюється простим перемноженням рівня сигналу, який є на даний момент часу на число від 0.1 до 1.

Шляхом тестів було визначено, що оптимальні максимальні значення для параметрів огинаючої наступні: 1 секунда на атаку, 1 секунда на повернення до встановленого рівня сигналу та 3 секунди на затухання.

3.1.3. Фільтр

Сам фільтр, додається так само, як і інші елементи створенням нового класу та додаванням відповідних функцій і їх параметрів у дерево значень [27]. Як показано у Додаток Д. Скрипт фільтра, у фільтра є можливість вибору між фільтром низьких частот, фільтром високих частот, та смуговим фільтром, аналогічно, до реалізації вибору форми хвилі.

Фільтр має можливість регулювання частоти зрізу у діапазоні 20Гц – 20 кГц та можливість регулювання резонансу на частоті зрізу.

Використання конструкції дерева значень дає можливість не створювати новий клас для кожної нової огинаючої, а просто додати параметри нової огинаючої у дерево значень.

Огинаюча фільтра реалізована таким чином, що кожен наступний семпл, значення частоти зрізу перемножується на деяке значення модулятора, який в свою чергу змінюється у часі за принципом «ADSR» від 0.1 до 1.

Для того, щоб уникнути помилки, під час якої частота зрізу виходить за межі встановленого діапазону, було додано алгоритм, який повертає частоті зрізу значення 20 Гц, якщо вона опускається нижче цього значення, або значення 20 кГц, коли частота підіймається вище цього значення.

3.1.4. Процесор та графічний інтерфейс

Всі окремі елементи об’єднуються у великому блоці процесору. Саме у цьому блоці з’являється звук. У Додаток Б. Скрипт процесора видно, що саме тут знаходиться дерево значень, саме звідси всі значення передаються у відповідні

блоки і саме процесор відповідає за те, в якому стані зараз знаходиться голос синтезатора.

Для того, щоб отримати поліфонію, для функції додавання голосу було використано базовий цикл «for» з лічильником, максимальне значення якого відповідає кількості одночасно натиснутих клавіш.

Як видно з Додаток Б. Скрипт процесора, всі основні методи, які були передані до синтезаторного голосу викликаються саме через нього.

Згідно до Додаток Е. Скрипт графічного інтерфейсу, графічний інтерфейс був реалізований окремо для кожного елемента, з ціллю слідування принципам ООП.

Як було визначено раніше, JUCE має дуже гнучкі можливості у створенні графічного інтерфейсу. У даному випадку всі елементи були ретельно пророблені в окремих класах, та потім поєднані у головному скрипті графічного інтерфейсу.

Відповідно до реальних аналогів, для регулювання частот модуляції і зрізу фільтра, а також для регулювання глибини модулюючої хвилі і резонансу фільтра були обрані обертальні перемикачі, для регулювання параметрів ADSR огинаючих були обрані вертикальні слайдери та вікна вибору для перемикання форми хвилі та типу фільтра.

3.2. ПЕРЕВІРКА РОБОТИ СИНТЕЗАТОРА

3.2.1. Осцилятор та частотна модуляція

Для перевірки точності перетворення MIDI сигналу у частоту, було проведено ряд експериментів, які полягали у натисканні клавіши Ля різних октав і порівнянні отриманих частот зі стандартами (Рисунок 3.1).

Рисунок 3.1 – Нота Ля першої октави

Як можна побачити, Ля першої октави майже точно відповідає стандарту 440 Гц.

Для чистоти експерименту було проведено ще декілька порівнянь (Рисунок 3.2).

Рисунок 3.2 – Нота Ля другої октави

Нота Ля другої октави має більшу розбіжність зі стандартом 880 Гц, але відносно того, що і сама частота удвічі більша, погрішність є такою ж, як і у першому експерименті.

Останнім експериментом було перевірено на відповідність ноту Ля малої октави (Рисунок 3.3).

Рисунок 3.3 – Нота Ля малої октави

Як можна побачити, розбіжність від стандарту 8 Гц, що є великою похибкою відносно 220 Гц, але таке таку похибку можна пояснити не зовсім коректним відображенням частоти сигналу, більш вірне значення можна побачити унизу Рисунка 3.3.

Також було проведено ряд досліджень, щодо форми хвилі (Рисунок 3.4).

Рисунок 3.4 – Отримані форми хвиль

Як можна побачити з осцилограм, всі форми хвиль відповідають заданим параметрам.

Для перевірки працездатності частотної модуляції було проведено експеримент у якому значення частоти модулятора було встановлено на відмітці 290 Гц, а глибину модуляції на 6.5 (Рисунок 3.5)

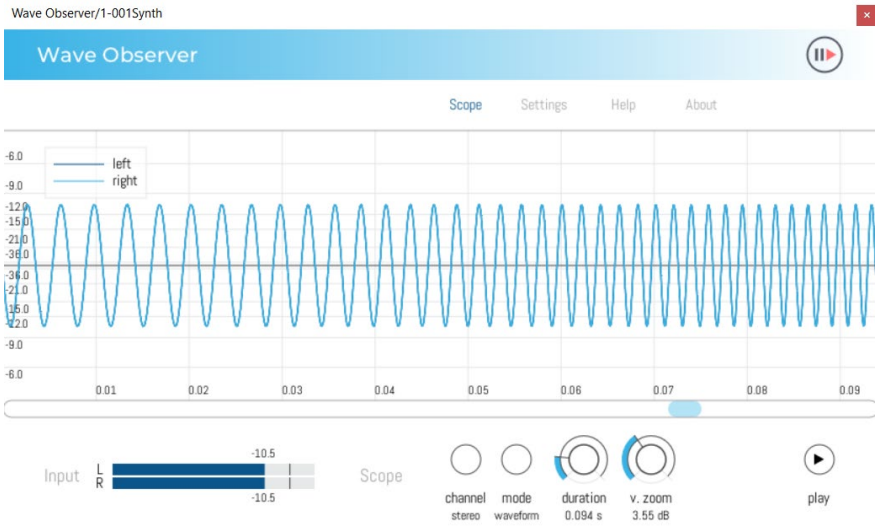


Рисунок 3.5 – Осцилограма частотної модуляції

Як видно з осцилограми частота коливань сигналу дійсно змінюється відповідно до встановлених значень.

3.2.2. ADSR огинаюча

Перевірка працездатності огинаючої була проведена шляхом порівняння осцилограм сигналів, у стані коли наростання і затухання встановлено на позначці 0.1 секунда та коли вони становлять одну секунду (Рисунок 3.6).

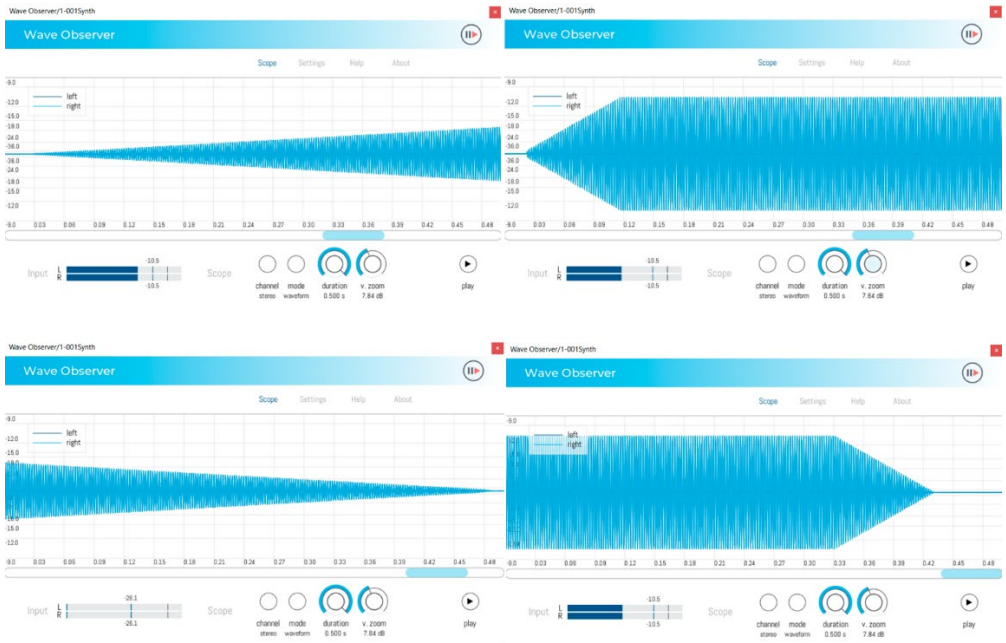


Рисунок 3.6 – Осцилограми наростання і затухання (ліворуч 1 секунда, праворуч 0.1 секунда)

Час наростання і затухання сигналу співпадає зі встановленими значеннями.

3.2.3. Фільтр та його огинаюча

Для того, щоб визначити точність передачі параметрів фільтра було проведено експеримент, у якому параметри фільтра були виставлені наступним чином (Рисунок 3.7).

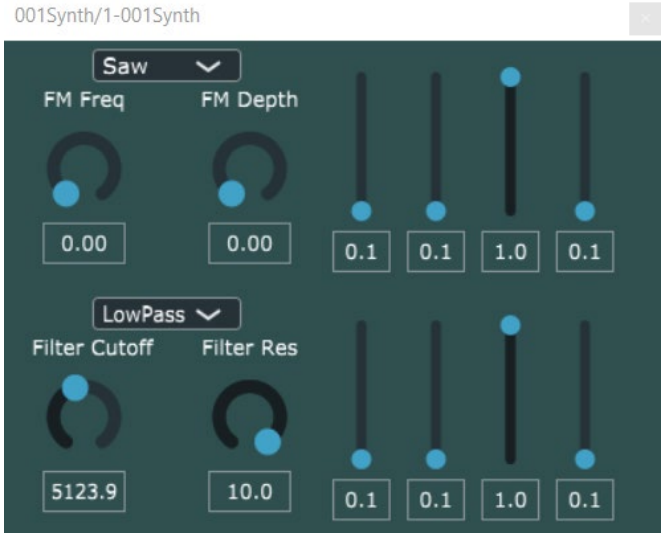


Рисунок 3.7 – Параметри фільтра

При аналізі амплітудно-частотної характеристики сигналу було визначено, що встановлені параметри відповідають дійсності (Рисунок 3.8).

Рисунок 3.8 – АЧХ з увімкненим фільтром

При встановлені наростання у положення 0.5 секунд, а затухання у положення 1 секунди була отримана наступна амплітудно-частотна характеристика (Рисунок 3.9).

Рисунок 3.1 – АЧХ з огинаючою фільтра

У порівнянні з попереднім графіком, можна визначити, що резонансна частота змінювалась у часі.

ФВЧ з частотою зрізу 880 Гц має наступну АЧХ (Рисунок 3.10).

Рисунок 3.10 – АЧХ фільтра високих частот

Смуговий фільтр з частотою зрізу 3500 Гц має наступну АЧХ (Рисунок 3.11).

Рисунок 3.11 – АЧХ смугового фільтра

ВИСНОВОК ДО РОЗДІЛУ

У ході розробки було виявлено ряд проблем, рішення яких було описане у даному розділі, або буде описане у наступному. Було створено повністю готовий, стабільно працюючий синтезатор з можливістю вибору форми хвилі, частотною модуляцією, трьома фільтрами та двома огинаючими до фільтра і до самого сигналу.

У ході тестів було визначено, що усі інтегровані елементи працюють вірно, та відповідно, до встановлених вимог і характеристик.

4. ОБГОВОРЕННЯ РЕЗУЛЬТАТІВ

4.1. АНАЛІЗ ХАРАКТЕРИСТИК СИНТЕЗАТОРА

Розроблений синтезатор отримав наступні характеристики:

- Чотири форми генерованого сигналу.
- Можливість частотної модуляції з налаштуванням глибини модуляційної хвилі.
- ADSR огинаюча.
- Фільтр з можливістю вибору між низькочастотним, високочастотним та смуговим. Перемикачі регулювання частоти зрізу та резонансу.
- Огинаюча фільтру.
- Можливість роботи у якості VST плагіна у DAW, або у самостійному режимі.
- П’ятиклавішна поліфонія.
- Відкритий код.

З особливостей можна відмітити те, що при налаштуванні ручки глибини модуляції у позицію 0.0, ручка частоти модуляції може виконувати функцію детюна сигналу, або керувати його висотою. При правильному налаштуванні параметрів модуляції можна отримати приємне вібрато.

З наявних недоліків можна виділити те, що для того, щоб затухання працювало належним чином повзунки обох огинаючих повинні бути встановленими в однакове положення. Також, при занадто «агресивному» налаштуванні параметрів модуляції можна отримати артефакти, та помилки у роботі програми, але у більшості випадків ця проблема вирішується зменшенням рівня сигналу.

4.2. ПОРІВНЯННЯ З МОДУЛЬНИМ АНАЛОГОМ

Для порівняння у середовищі VCV Rack 2 було створено модель синтезатора, який на скільки це можливо повторює функціонал і характеристики розробленого синтезатора (Рисунок 4.1).

Рисунок 4.1 – Модель розробленого синтезатора

Перше, що хотілося б відзначити у ході порівняння – це складність сприйняття інтерфейсу змодельованого синтезатора недосвідченими людьми. А простота використання була одною з головних вимог до розробки.

Хоча отримана модель має можливість вбудовування її у різні DAW, все ж розроблений у ході роботи синтезатор у цьому плані значно виграє.

Функція детюна, та налаштування висоти тону випадає, через відсутність ручки регулювання глибини модуляції. Огинаюча фільтра працює дуже слабко і не завжди коректно.

У ході порівняння було визначено деяку схожість процесу програмної реалізації синтезатора у JUCE з поєднанням модулів у VCV Rack 2. Фактично, було створено ті ж самі модулі ADSR, VCO, VCF та LFO, але замість дротів, вони були поєднані у блоках голосу синтезатора та процесора.

4.3. ГАЛУЗІ ЗАСТОСУВАННЯ ТА ШЛЯХИ РОЗВИТКУ

Розроблений синтезатор має широку сферу застосування. Здебільшого у галузі звукового-дизайну.

У ході тестів було отримано безліч цікавих звуків, від тих, які могли б дуже добре вписатись у SciFy фільми, до звуків, схожих на смичкові інструменти. Окрім звукового-дизайну розробка може додавати унікальності музичним композиціям у майже будь-якому жанрі.

Варто відмітити, що в поєднанні з базовими плагінами ефектів у будь-якій DAW, наприклад реверберації, синтезатор значно набирає потенціалу.

У поєднанні з плагінами, які були використані мною у ході апробації, синтезатор може використовуватись у навчальних цілях. Його простота, дає змогу дуже прозоро, та швидко пояснити на реальних, візуальних прикладах такі поняття, як частотна модуляція, форма сигналу, висота тону, фільтри, частота зрізу, резонансна частота, АЧХ та багато інших речей. Те, що розробка має відкритий код, також можна використати у навчальних цілях. Варто відмітити, що синтезатор дає змогу проводити як очне, так і дистанційне навчання, що є дуже актуальним сьогодні.

У ході пошукової роботи було визначено декілька найпопулярніших методів синтезу звуку. У ході порівняння синтезатора з модульним аналогом було визначено збіжність з’єднання модулів у модульному синтезі і написання класів у JUCE. Отже, маючи достатньо уяви синтезатор може бути модернізовано до нескінченості.

Один лише концепт частотної модуляції дає можливість створювати будь-які ланцюжки з операторів. Якщо відійти від дотримання канонів можна створити гібридний варіант, продублювавши отриманий синтезатор для отримання адитивного синтезатора. Додавши фільтру частотну модуляцію, буде отримана варіація субтрактивного синтезатора.

Якщо обговорювати канонічний обраний шлях доступності та простоти, у майбутньому можна розробити адаптивну систему навчання навичкам синтезу, яка буде ускладнюватись в залежності від рівня володіння синтезатором. Окремі модулі для кожного елемента дають таку можливість.

ВИСНОВОК ДО РОЗДІЛУ

У четвертому розділі було висвітлено наявні переваги та недоліки синтезатора. Визначено, що розробка нічим не поступається модульному аналогу.

Визначено, що основними галузями застосування синтезатора є музична галузь, звуковий-дизайн та освітня галузь. Обрано, та описано деякі з найбільш продуктивних шляхів модернізації, та подальшого розвитку проекту.

ВИСНОВОК

У ході виконання роботи, було проведено детальний аналіз методів синтезу та їх математичних моделей. Розглянута максимально широка вибірка існуючих рішень, від віртуальних модулів осцилятора, до одних з найпопулярніших реальних моделей синтезаторів.

Проведено глибокий розбір інструментаріїв для програмної реалізації проекту. З урахуванням вимог, аналізу існуючих рішень та розглянутих методів синтезу був обраний фреймворк який найбільше задовільнив усі поставлені вимоги.

Проведена розробка з урахуванням принципів ООП. Проведена детальна апробація результатів розробки і визначено всі основні переваги та недоліки проекту. Багато часу було присвячено тестуванням синтезатора, як окремо, так і в поєднанні з різними плагінами ефектів. Визначено основні галузі застосування синтезатора. Визначено, що розробка не поступається модульному аналогу. Визначено основні шляхи подальшого розвитку.

У ході роботи були досягнені усі поставлені задачі. Синтезатор вийшов простим у використанні та доступним будь-якому музиканту, або саунд-дизайнеру.

Перелік посилань

[1] A. Horner, J. Beauchamp, and L. Haken, "Machine Tongues XVI: Genetic Algorithms and Their Application to FM Matching Synthesis," *Computer Music Journal,* vol. 17, no. 4, pp. 17-29, 1993.

[2] R. Boulanger, The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing,and Programming, Cambridge: MIT Press, 1999.

[3] J. Chowning, "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation," *The Journal of the Audio Engineering Society,* vol. 21, no. 7, pp. 526-534, 1973.

[4] J. Chowning and D. Bristov, Fm Theory and Applications: By Musicians for Musicians, Tokyo: Yamaha Music Foundation, 1987.

[5] J. Lazzaro and J. Wawrzynek, "Subtractive Synthesis without Filters," in *Audio Anecdotes II—Tools*, New York, A K Peters/CRC Press, 2004, pp. 55-64.

[6] J. Moorer, "The Synthesis of Complex Audio Spectra by Means of Discrete Summation Formulae," *Journal of the Audio Engineering Society,* vol. 24, pp. 717-727, 1976.

[7] R. Mottola, "Table of Musical Notes and Their Frequencies and Wavelengths," Liutaio Mottola, 9 January 2020. [Online]. Available: https://www.liutaiomottola.com/formulae/freqtab.htm. [Accessed 29 April 2022].

[8] «Fundamental Frequency and Harmonics,» the Physics Classroom, [Онлайновий]. Available: https://www.physicsclassroom.com/class/sound/Lesson-4/Fundamental-Frequency-and-Harmonics.

[9] T. Noakes, «Synth of the Month:Fairlight CMI,» [Онлайновий]. Available: https://synthhero.com/FAIRLIGHT-CMI. [Дата звернення: 29 Квітень 2022].

[10] G. Derveaux, "Time-Domain Simulation of a Guitar: Model and Method," *Journal of the Acoustical Society of America,* vol. 114, no. 6, pp. 3368-3383, 2003.

[11] M. Laurson, "Methods for Modeling Realistic Playing in Acoustic Guitar Synthesis," *Computer Music Journal,* vol. 25, no. 3, pp. 38-49, 2001.

[12] «File:Voltage controlled oscillator (VCO) diagram.jpg,» [Онлайновий]. Available: https://commons.wikimedia.org/wiki/File:Voltage_controlled_oscillator_(VCO)_diagram.jpg. [Дата звернення: 28 Травень 2022].

[13] T. Kvifte, «The elements in an ADSR envelope,» [Онлайновий]. Available: https://www.researchgate.net/figure/The-elements-in-an-ADSR-envelope_fig22_270819567. [Дата звернення: 29 Травень 2022].

[14] R. Rabenstein and L. Trautmann, "Multirate Simulations of String Vibrations Including Nonlinear Fret-String Intrections Using the Functional Transformation Method," *EURASIP Journal on Advances in Signal Processing,* vol. 2004, no. 7, pp. 949-963, 2004.

[15] «File:Butterworth response.png,» [Онлайновий]. Available: https://commons.wikimedia.org/w/index.php?curid=202308. [Дата звернення: 28 Квітень 2022].

[16] «KORG MS-20 V,» Arturia, [Онлайновий]. Available: https://www.arturia.com/products/software-instruments/korg-ms-20-v/overview. [Дата звернення: 29 Травень 2022].

[17] «Audible Instruments Macro Oscillator 2,» VCV, [Онлайновий]. Available: https://library.vcvrack.com/AudibleInstruments/Plaits. [Дата звернення: 29 Квітень 2022].

[18] «Omnisphere,» Specrasonics, [Онлайновий]. Available: https://www.spectrasonics.net/products/omnisphere/index.php. [Дата звернення: 29 Квітень 2022].

[19] «3x Osc,» Image Line, [Онлайновий]. Available: https://www.image-line.com/fl-studio-learning/fl-studio-online-manual/html/plugins/3x%20Osc.htm. [Дата звернення: 29 Квітень 2022].

[20] «OP-1 portable synthesizer,» Teenage engeneering, [Онлайновий]. Available: https://teenage.engineering/store/op-1/. [Дата звернення: 29 Квітень 2022].

[21] obiwanjacobi, «VST.NET,» [Онлайновий]. Available: https://github.com/obiwanjacobi/vst.net. [Дата звернення: 29 Квітень 2022].

[22] «iPlug 2 - C++ audio plug-in framework,» [Онлайновий]. Available: https://iplug2.github.io/. [Дата звернення: 29 Квітень 2022].

[23] «Developer: VST 3 SDK,» Steinberg, [Онлайновий]. Available: https://forums.steinberg.net/c/developer/vst-3-sdk/105. [Дата звернення: 29 Квітень 2022].

[24] «JUCE,» [Онлайновий]. Available: https://juce.com/. [Дата звернення: 29 Квітень 2022].

[25] «dsp::Oscillator< SampleType > Class Template Reference,» JUCE, [Онлайновий]. Available: https://docs.juce.com/master/classdsp_1_1Oscillator.html. [Дата звернення: 8 Червень 2022].

[26] «ADSR Class Reference,» [Онлайновий]. Available: https://docs.juce.com/master/classADSR.html. [Дата звернення: 8 Квітень 2022].

[27] «dsp::StateVariableFilter::Filter< SampleType > Class Template Reference,» JUCE, [Онлайновий]. Available: https://docs.juce.com/master/classdsp_1_1StateVariableFilter_1_1Filter.html. [Дата звернення: 8 Квітень 2022].

Додатки

Додаток А. Скрипт осцилятора

```
/*
  ==============================================================================

    OscilatorBack.cpp
    Created: 25 May 2022 5:35:51pm
    Author:  Илья

  ==============================================================================
*/

#include "OscBack.h"

void OscBack::prepareToPlay(juce::dsp::ProcessSpec& spec)
{
	fmOperator.prepare(spec);
	prepare(spec);
}

void OscBack::setWaveType(const int choice)
{
	switch (choice)
	{
	case 0:
		initialise([](float x) {return std::sin(x); });
//Sine Wave
		break;
	case 1:
		initialise([](float x) {return x / juce::MathConstants<float>::pi; }); //
Saw Wave
		break;
	case 2:
		initialise([](float x) {return x < 0.0f ? -1.0f : 1.0f; });
// Square Wave
		break;
	case 3:
		initialise([](float x) {return abs(x / juce::MathConstants<float>::pi);
});
// Triangle Wave
		break;
	default:
		jassertfalse;
		break;
	}

}

void OscBack::setWaveFrequency(const int midiNoteNumber)
{
	setFrequency(juce::MidiMessage::getMidiNoteInHertz(midiNoteNumber) + fmMod);
	lastMidiNote = midiNoteNumber;
}

void OscBack::getNextAudioBlock(juce::dsp::AudioBlock<float>& block)
{
	for (int chanel = 0; chanel < block.getNumChannels(); ++chanel)
	{
		for (int sample = 0; sample < block.getNumSamples(); ++sample)
		{
			fmMod = fmOperator.processSample(block.getSample(chanel,
sample))*fmDepth;
		}
```

```
        }
        process(juce::dsp::ProcessContextReplacing<float>(block));
}

void OscBack::setFmParams(const float depth, const float freq)
{
        fmOperator.setFrequency(freq);
        fmDepth = depth;
        auto FreqNow = juce::MidiMessage::getMidiNoteInHertz(lastMidiNote) + fmMod;
        setFrequency(FreqNow >= 0 ? FreqNow : FreqNow * -1.0f);
}
/*
  ==============================================================================

    OscilatorBack.h
    Created: 25 May 2022 5:35:51pm
    Author:  Илья

  ==============================================================================
*/

#pragma once
#include <JuceHeader.h>

class OscBack : public juce::dsp::Oscillator<float>
{
public:
    void prepareToPlay(juce::dsp::ProcessSpec& spec);
    void setWaveType(const int choice);
    void setWaveFrequency(const int midiNoteNumber);
    void getNextAudioBlock(juce::dsp::AudioBlock<float>& block);
    void setFmParams(const float depth, const float freq);

private:
    juce::dsp::Oscillator<float> fmOperator{ [](float x) {return std::sin(x); } };
    float fmMod{ 0.0f };
    float fmDepth{ 0.0f };
    int lastMidiNote{ 0 };
};
```

Додаток Б. Скрипт процесора

```
// PluginProcessor.cpp

#include "PluginProcessor.h"
#include "PluginEditor.h"

//==============================================================================
ClassSynthAudioProcessor::ClassSynthAudioProcessor()
#ifndef JucePlugin_PreferredChannelConfigurations
     : AudioProcessor (BusesProperties()
                     #if ! JucePlugin_IsMidiEffect
                      #if ! JucePlugin_IsSynth
                       .withInput  ("Input",  juce::AudioChannelSet::stereo(), true)
                      #endif
                       .withOutput ("Output", juce::AudioChannelSet::stereo(), true)
                     #endif
                       ),apvts (*this, nullptr, "Parameters", createParameters())
#endif
{
    synth.addSound(new SynthSound());
    for (int i = 0; i < 5; i++)
    {
        synth.addVoice(new SynthVoice());
    }
}

ClassSynthAudioProcessor::~ClassSynthAudioProcessor()
{

}

//==============================================================================
const juce::String ClassSynthAudioProcessor::getName() const
{
    return JucePlugin_Name;
}

bool ClassSynthAudioProcessor::acceptsMidi() const
{
   #if JucePlugin_WantsMidiInput
    return true;
   #else
    return false;
   #endif
}

bool ClassSynthAudioProcessor::producesMidi() const
{
   #if JucePlugin_ProducesMidiOutput
    return true;
   #else
    return false;
   #endif
}

bool ClassSynthAudioProcessor::isMidiEffect() const
{
   #if JucePlugin_IsMidiEffect
    return true;
   #else
```

```
    return false;
   #endif
}

double ClassSynthAudioProcessor::getTailLengthSeconds() const
{
    return 0.0;
}

int ClassSynthAudioProcessor::getNumPrograms()
{
    return 1;   // NB: some hosts don't cope very well if you tell them there are 0
programs,
                // so this should be at least 1, even if you're not really
implementing programs.
}

int ClassSynthAudioProcessor::getCurrentProgram()
{
    return 0;
}

void ClassSynthAudioProcessor::setCurrentProgram (int index)
{
}

const juce::String ClassSynthAudioProcessor::getProgramName (int index)
{
    return {};
}

void ClassSynthAudioProcessor::changeProgramName (int index, const juce::String&
newName)
{
}

//==============================================================================
void ClassSynthAudioProcessor::prepareToPlay (double sampleRate, int samplesPerBlock)
{
    synth.setCurrentPlaybackSampleRate(sampleRate);

    for (int i = 0; i < synth.getNumVoices(); i++)
    {
        if (auto voice = dynamic_cast<SynthVoice*>(synth.getVoice(i)))
        {
            voice->prepareToPlay(sampleRate, samplesPerBlock,
getTotalNumOutputChannels());
        }
    }
}

void ClassSynthAudioProcessor::releaseResources()
{
    // When playback stops, you can use this as an opportunity to free up any
    // spare memory, etc.
}

#ifndef JucePlugin_PreferredChannelConfigurations
bool ClassSynthAudioProcessor::isBusesLayoutSupported (const BusesLayout& layouts)
const
{
  #if JucePlugin_IsMidiEffect
    juce::ignoreUnused (layouts);
    return true;
```

```
  #else
    // This is the place where you check if the layout is supported.
    // In this template code we only support mono or stereo.
    // Some plugin hosts, such as certain GarageBand versions, will only
    // load plugins that support stereo bus layouts.
    if (layouts.getMainOutputChannelSet() != juce::AudioChannelSet::mono()
     && layouts.getMainOutputChannelSet() != juce::AudioChannelSet::stereo())
        return false;

    // This checks if the input layout matches the output layout
   #if ! JucePlugin_IsSynth
    if (layouts.getMainOutputChannelSet() != layouts.getMainInputChannelSet())
        return false;
   #endif

    return true;
  #endif
}
#endif

void ClassSynthAudioProcessor::processBlock (juce::AudioBuffer<float>& buffer,
juce::MidiBuffer& midiMessages)
{
    juce::ScopedNoDenormals noDenormals;
    auto totalNumInputChannels  = getTotalNumInputChannels();
    auto totalNumOutputChannels = getTotalNumOutputChannels();


    for (auto i = totalNumInputChannels; i < totalNumOutputChannels; ++i)
        buffer.clear (i, 0, buffer.getNumSamples());

    for (int i = 0; i < synth.getNumVoices(); ++i)
    {
        if (auto voice = dynamic_cast<SynthVoice*>(synth.getVoice(i)))
        {
            auto& oscWaveType = *apvts.getRawParameterValue("OSC");

            auto& fmFreq = *apvts.getRawParameterValue("FMFREQ");
            auto& fmDepth = *apvts.getRawParameterValue("FMDEPTH");

            auto& attack = *apvts.getRawParameterValue("ATTACK");
            auto& decay = *apvts.getRawParameterValue("DECAY");
            auto& sustain = *apvts.getRawParameterValue("SUSTAIN");
            auto& release = *apvts.getRawParameterValue("RELEASE");

            auto& filterType = *apvts.getRawParameterValue("FILTER");
            auto& cutoff = *apvts.getRawParameterValue("FILTERCUTOFF");
            auto& resonance = *apvts.getRawParameterValue("FILTERRES");

            auto& modAttack = *apvts.getRawParameterValue("MODATTACK");
            auto& modDecay = *apvts.getRawParameterValue("MODDECAY");
            auto& modSustain = *apvts.getRawParameterValue("MODSUSTAIN");
            auto& modRelease = *apvts.getRawParameterValue("MODRELEASE");

            voice->getOscilator().setWaveType(oscWaveType);
            voice->getOscilator().setFmParams(fmFreq, fmDepth);
            voice->updateAdsr(attack.load(), decay.load(), sustain.load(),
release.load());
            voice->updateFilter(filterType.load(), cutoff.load(), resonance.load());
            voice->updateModAdsr(modAttack, modDecay, modSustain, modRelease);
        }
    }
    synth.renderNextBlock(buffer, midiMessages, 0, buffer.getNumSamples());
```

```
}

//==============================================================================
bool ClassSynthAudioProcessor::hasEditor() const
{
    return true; // (change this to false if you choose to not supply an editor)
}

juce::AudioProcessorEditor* ClassSynthAudioProcessor::createEditor()
{
    return new ClassSynthAudioProcessorEditor (*this);
}

//==============================================================================
void ClassSynthAudioProcessor::getStateInformation (juce::MemoryBlock& destData)
{
    // You should use this method to store your parameters in the memory block.
    // You could do that either as raw data, or use the XML or ValueTree classes
    // as intermediaries to make it easy to save and load complex data.
}

void ClassSynthAudioProcessor::setStateInformation (const void* data, int sizeInBytes)
{
    // You should use this method to restore your parameters from this memory block,
    // whose contents will have been created by the getStateInformation() call.
}

//==============================================================================
// This creates new instances of the plugin..
juce::AudioProcessor* JUCE_CALLTYPE createPluginFilter()
{
    return new ClassSynthAudioProcessor();
}

juce::AudioProcessorValueTreeState::ParameterLayout
ClassSynthAudioProcessor::createParameters()
{
    std::vector<std::unique_ptr<juce::RangedAudioParameter>> params;

    //Oscillator and Frequency Modulator
    params.push_back(std::make_unique<juce::AudioParameterChoice>("OSC", "Oscillator",
juce::StringArray{ "Sine", "Saw",
        "Square", "Triangle"}, 0));
    params.push_back(std::make_unique<juce::AudioParameterFloat>("FMFREQ", "FM
Frequency", juce::NormalisableRange<float>
    {0.0f, 5000.0f, 0.01f, 0.3f}, 0.0f));
    params.push_back(std::make_unique<juce::AudioParameterFloat>("FMDEPTH", "FM
Depth", juce::NormalisableRange<float>
    {0.0f, 1000.0f, 0.01f, 0.3f}, 0.0f));

    //ADSR
    params.push_back(std::make_unique<juce::AudioParameterFloat>("ATTACK", "Attack",
juce::NormalisableRange<float>
        {0.1f, 1.0f, 0.1f}, 0.1f));
    params.push_back(std::make_unique<juce::AudioParameterFloat>("DECAY", "Decay",
juce::NormalisableRange<float>
        {0.1f, 1.0f, 0.1f}, 0.1f));
    params.push_back(std::make_unique<juce::AudioParameterFloat>("SUSTAIN", "Sustain",
juce::NormalisableRange<float>
        {0.1f, 1.0f, 0.1f }, 1.0f));
    params.push_back(std::make_unique<juce::AudioParameterFloat>("RELEASE", "Release",
juce::NormalisableRange<float>
        {0.1f, 3.0f, 0.1f }, 0.4f));
```

```
    //Filter ADSR
    params.push_back(std::make_unique<juce::AudioParameterFloat>("MODATTACK",
"Attack", juce::NormalisableRange<float>
    {0.1f, 1.0f, 0.1f}, 0.1f));
    params.push_back(std::make_unique<juce::AudioParameterFloat>("MODDECAY", "Decay",
juce::NormalisableRange<float>
    {0.1f, 1.0f, 0.1f}, 0.1f));
    params.push_back(std::make_unique<juce::AudioParameterFloat>("MODSUSTAIN",
"Sustain", juce::NormalisableRange<float>
    {0.1f, 1.0f, 0.1f }, 1.0f));
    params.push_back(std::make_unique<juce::AudioParameterFloat>("MODRELEASE",
"Release", juce::NormalisableRange<float>
    {0.1f, 3.0f, 0.1f }, 0.4f));

    //Filter

    params.push_back(std::make_unique<juce::AudioParameterChoice>("FILTER", "Filter",
juce::StringArray{ "LowPass", "HighPass",
        "BandPass" }, 0));
    params.push_back(std::make_unique<juce::AudioParameterFloat>("FILTERCUTOFF",
"Filter Cutoff", juce::NormalisableRange<float>
    {20.0f, 20000.0f, 0.1f, 0.6f}, 200.0f));
    params.push_back(std::make_unique<juce::AudioParameterFloat>("FILTERRES", "Filter
Resonace", juce::NormalisableRange<float>
    {1.0f, 10.0f, 0.1f}, 1.0f));


    return{ params.begin(),params.end() };
      }


// PluginProcessor.h

#pragma once

#include <JuceHeader.h>
#include "SynthVoice.h"
#include "SynthSound.h"


//==============================================================================
/**
*/
class ClassSynthAudioProcessor  : public juce::AudioProcessor
{
public:
    //==========================================================================
    ClassSynthAudioProcessor();
    ~ClassSynthAudioProcessor() override;

    //==========================================================================
    void prepareToPlay (double sampleRate, int samplesPerBlock) override;
    void releaseResources() override;

   #ifndef JucePlugin_PreferredChannelConfigurations
    bool isBusesLayoutSupported (const BusesLayout& layouts) const override;
   #endif

    void processBlock (juce::AudioBuffer<float>&, juce::MidiBuffer&) override;

    //==========================================================================
    juce::AudioProcessorEditor* createEditor() override;
    bool hasEditor() const override;
```

```
    //==============================================================================
    const juce::String getName() const override;

    bool acceptsMidi() const override;
    bool producesMidi() const override;
    bool isMidiEffect() const override;
    double getTailLengthSeconds() const override;

    //==============================================================================
    int getNumPrograms() override;
    int getCurrentProgram() override;
    void setCurrentProgram (int index) override;
    const juce::String getProgramName (int index) override;
    void changeProgramName (int index, const juce::String& newName) override;

    //==============================================================================
    void getStateInformation (juce::MemoryBlock& destData) override;
    void setStateInformation (const void* data, int sizeInBytes) override;

    juce::AudioProcessorValueTreeState apvts;

private:

    juce::Synthesiser synth;
    juce::AudioProcessorValueTreeState::ParameterLayout createParameters();


    //==============================================================================
    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (ClassSynthAudioProcessor)
};
```

Додаток В. Скрипт синтезаторного голосу

```
/*
  ==============================================================================

    SynthVoice.cpp
    Created: 23 May 2022 10:08:36pm
    Author:  Илья

  ==============================================================================
*/

#include "SynthVoice.h"

bool SynthVoice::canPlaySound(juce::SynthesiserSound* sound)
{
    return dynamic_cast<juce::SynthesiserSound*> (sound) != nullptr;
}

void SynthVoice::startNote(int midiNoteNumber, float velocity, juce::SynthesiserSound*
sound, int currentPitchWheelPosition)
{
    osc.setWaveFrequency(midiNoteNumber);
    adsr.noteOn();
    modAdsr.noteOn();
}

void SynthVoice::stopNote(float velocity, bool allowTaiOff)
{
    adsr.noteOff();
    modAdsr.noteOff();

    if (!allowTaiOff || !adsr.isActive())
        clearCurrentNote();
}

void SynthVoice::controllerMoved(int controllerNumber, int newControllerValue)
{

}

void SynthVoice::pitchWheelMoved(int newPitchWheelValue)
{

}

void SynthVoice::prepareToPlay(double sampleRate, int samplesPerBlock, int
outputChanels)
{
    juce::dsp::ProcessSpec spec;
    spec.maximumBlockSize = samplesPerBlock;
    spec.sampleRate = sampleRate;
    spec.numChannels = outputChanels;

    osc.prepareToPlay(spec);
    adsr.setSampleRate(sampleRate);
    filter.prepareToPlay(sampleRate, samplesPerBlock, outputChanels);
    modAdsr.setSampleRate(sampleRate);
    gain.prepare(spec);

    gain.setGainLinear(0.3f);
```

```
    isPrepared = true;

}

void SynthVoice::updateAdsr (const float attack, const float decay, const float
sustain, const float release)
{
    adsr.updateADSR(attack, decay, sustain, release);
}

void SynthVoice::renderNextBlock(juce::AudioBuffer< float >& outputBuffer, int
startSample, int numSamples)
{
    jassert(isPrepared);

    if (!isVoiceActive())
        return;

    synthBuffer.setSize(outputBuffer.getNumChannels(), numSamples, false, false,
true);
    modAdsr.applyEnvelopeToBuffer(synthBuffer, 0, numSamples);
    synthBuffer.clear();

    juce::dsp::AudioBlock<float> audioBlock{ synthBuffer };
    osc.getNextAudioBlock(audioBlock);
    adsr.applyEnvelopeToBuffer(synthBuffer, 0, synthBuffer.getNumSamples());
    filter.process(synthBuffer);
    gain.process(juce::dsp::ProcessContextReplacing<float>(audioBlock));

    for (int chanel = 0; chanel < outputBuffer.getNumChannels(); ++chanel)
    {
        outputBuffer.addFrom(chanel, startSample, synthBuffer, chanel, 0, numSamples);

        if (!adsr.isActive())
            clearCurrentNote();
    }
}

void SynthVoice::updateFilter(const int filterType, const float cutoff, const float
resonance)
{
   float modulator = modAdsr.getNextSample();
   filter.updateParameters(filterType, cutoff, resonance, modulator);
}

void SynthVoice::updateModAdsr(const float attack, const float decay, const float
sustain, const float release)
{
    modAdsr.updateADSR(attack, decay, sustain, release);
}


/*
  ==============================================================================

    SynthVoice.h
    Created: 23 May 2022 10:08:36pm
    Author:  Илья

  ==============================================================================
*/

#pragma once
```

```
#include <JuceHeader.h>
#include "SynthSound.h"
#include "BackEnd/OscBack.h"
#include "BackEnd/AdsrBack.h"
#include "BackEnd/FilterBack.h"

class SynthVoice :public juce::SynthesiserVoice
{
public:
    bool canPlaySound(juce::SynthesiserSound* sound) override;
    void startNote(int midiNoteNumber, float velocity, juce::SynthesiserSound* sound,
int currentPitchWheelPosition) override;
    void stopNote(float velocity, bool allowTaiOff) override;
    void controllerMoved(int controllerNumber, int newControllerValue) override;
    void pitchWheelMoved(int newPitchWheelValue) override;
    void prepareToPlay(double sampleRate, int samplesPerBlock, int outputChanels);
    void renderNextBlock(juce::AudioBuffer< float >& outputBuffer, int startSample,
int numSamples) override;

    void updateAdsr(const float attack, const float decay, const float sustain, const
float release);
    void updateFilter(const int filterType, const float cutoff, const float
resonance);
    void updateModAdsr(const float attack, const float decay, const float sustain,
const float release);

    OscBack& getOscilator() { return osc; }
private:
    juce::AudioBuffer<float> synthBuffer;

    OscBack osc;
    AdsrBack adsr;
    FilterBack filter;
    AdsrBack modAdsr;
    juce::dsp::Gain<float> gain;

    bool isPrepared{ false };
};


/*
  ==============================================================================

    SynthSound.h
    Created: 23 May 2022 10:09:31pm
    Author:  Илья

  ==============================================================================
*/

#pragma once

#include <JuceHeader.h>

class SynthSound :public juce::SynthesiserSound
{
public:
    bool appliesToNote(int midiNoteNumber) override { return true; }
    bool appliesToChannel(int midiChannel) override { return true; }

};
```

Додаток Г. Скрипт огинаючої

```
/*
  ==============================================================================

    AdsrBack.cpp
    Created: 25 May 2022 2:43:30pm
    Author:  Илья

  ==============================================================================
*/

#include "AdsrBack.h"
void AdsrBack::updateADSR(const float attack, const float decay, const float sustain,
const float release)
{
    adsrParams.attack = attack;
    adsrParams.decay = decay;
    adsrParams.sustain = sustain;
    adsrParams.release = release;

    setParameters(adsrParams);
}

/*
  ==============================================================================

    AdsrBack.h
    Created: 25 May 2022 2:43:30pm
    Author:  Илья

  ==============================================================================
*/

#pragma once

#include <JuceHeader.h>

class AdsrBack : public juce::ADSR
{
public:

    void updateADSR(const float attack, const float decay, const float sustain, const
float release);

private:

    juce::ADSR::Parameters adsrParams;

};
```

Додаток Д. Скрипт фільтра

```
/*
  ==============================================================================

    FilterBack.cpp
    Created: 27 May 2022 12:06:55am
    Author:  Илья

  ==============================================================================
*/

#include "FilterBack.h"

void FilterBack::prepareToPlay(double sampleRate, int samplesPerBlock, int
numChannels)
{
    filter.reset();

    juce::dsp::ProcessSpec spec;
    spec.maximumBlockSize = samplesPerBlock;
    spec.sampleRate = sampleRate;
    spec.numChannels = numChannels;

    filter.prepare(spec);

    isPrepared = true;
}

void FilterBack::process(juce::AudioBuffer<float>& buffer)
{
    jassert(isPrepared);

    juce::dsp::AudioBlock <float> block{ buffer };
    filter.process(juce::dsp::ProcessContextReplacing<float> {block});
}

void FilterBack::updateParameters(const int filterType, const float frequency, const
float resonance, const float modulator)
{
    switch (filterType)
    {
    case 0:
        filter.setType(juce::dsp::StateVariableTPTFilterType::lowpass);
        break;
    case 1:
        filter.setType(juce::dsp::StateVariableTPTFilterType::highpass);
        break;
    case 2:
        filter.setType(juce::dsp::StateVariableTPTFilterType::bandpass);
        break;
    }

    float modFreq = frequency * modulator;
    modFreq = std::fmin(std::fmax(modFreq, 20.0f), 20000.0f);

    filter.setCutoffFrequency(modFreq);
    filter.setResonance(resonance);
}

void FilterBack::reset()
{
    filter.reset();
```

```
      }
/*
  ==============================================================================

    FilterBack.h
    Created: 27 May 2022 12:06:55am
    Author:  Илья

  ==============================================================================
*/

#pragma once
#include <JuceHeader.h>

class FilterBack
{
public:
    void prepareToPlay(double sampleRate, int samplesPerBlock, int numChannels);
    void process(juce::AudioBuffer<float>& buffer);
    void updateParameters(const int filterType, const float frequency, const float
resonance, const float modulator = 1.0f);
    void reset();

private:
    juce::dsp::StateVariableTPTFilter<float> filter;
    bool isPrepared{ false };
      };
```

Додаток Е. Скрипт графічного інтерфейсу

```
/*
  ==============================================================================

    OscFront.cpp
    Created: 25 May 2022 6:54:52pm
    Author:  Илья

  ==============================================================================
*/

#include <JuceHeader.h>
#include "OscFront.h"

//==============================================================================
OscFront::OscFront(juce::AudioProcessorValueTreeState& apvts, juce::String
waveSelectId, juce::String fmFreqId, juce::String fmDepthId)
{
    juce::StringArray choices{ "Sine", "Saw", "Square", "Triangle" };
    oscWaveSelect.addItemList(choices, 1);
    addAndMakeVisible(oscWaveSelect);
    oscWaveSelectAttachment =
std::make_unique<juce::AudioProcessorValueTreeState::ComboBoxAttachment>(apvts,waveSel
ectId, oscWaveSelect);

    setSliderParams(fmFreqSlider, fmFreqLabel, apvts, fmFreqId,
fmFreqSliderAttachment);
    setSliderParams(fmDepthSlider, fmDepthLabel, apvts, fmDepthId,
fmDepthSliderAttachment);
}

OscFront::~OscFront()
{
}

void OscFront::paint (juce::Graphics& g)
{

}

void OscFront::resized()
{
    oscWaveSelect.setBounds(getWidth()/2 - 45, 5, 90, 20);

    fmFreqSlider.setBounds(0, 45, 100, 90);
    fmFreqLabel.setBounds(fmFreqSlider.getX(), fmFreqSlider.getY() - 20,
fmFreqSlider.getWidth(), 20);

    fmDepthSlider.setBounds(fmFreqSlider.getRight(), 45, 100, 90);
    fmDepthLabel.setBounds(fmDepthSlider.getX(), fmDepthSlider.getY() - 20,
fmDepthSlider.getWidth(), 20);
}

void OscFront::setSliderParams(juce::Slider& slider, juce::Label& label,
juce::AudioProcessorValueTreeState& apvts,
    juce::String paramId,
std::unique_ptr<juce::AudioProcessorValueTreeState::SliderAttachment>& attachment)
{
    slider.setSliderStyle(juce::Slider::SliderStyle::Rotary);
    slider.setTextBoxStyle(juce::Slider::TextBoxBelow, true, 50, 25);
    addAndMakeVisible(slider);
```

```
    attachment =
std::make_unique<juce::AudioProcessorValueTreeState::SliderAttachment>(apvts, paramId,
slider);

    label.setColour(juce::Label::ColourIds::textColourId, juce::Colours::white);
    label.setFont(15.0f);
    label.setJustificationType(juce::Justification::centred);
    addAndMakeVisible(label);
}
/*
  ==============================================================================

    OscFront.h
    Created: 25 May 2022 6:54:52pm
    Author:  Илья

  ==============================================================================
*/

#pragma once

#include <JuceHeader.h>

//==============================================================================
/*
*/
class OscFront  : public juce::Component
{
public:
    OscFront(juce::AudioProcessorValueTreeState& apvts,juce::String waveSelectId,
juce::String fmFreqId, juce::String fmDepthId);
    ~OscFront() override;

    void paint (juce::Graphics&) override;
    void resized() override;

private:
    juce::ComboBox oscWaveSelect;
    std::unique_ptr<juce::AudioProcessorValueTreeState::ComboBoxAttachment>
oscWaveSelectAttachment;

    juce::Slider fmFreqSlider;
    std::unique_ptr<juce::AudioProcessorValueTreeState::SliderAttachment>
fmFreqSliderAttachment;
    juce::Label fmFreqLabel{ "FM Freq", "FM Freq" };

    juce::Slider fmDepthSlider;
    std::unique_ptr<juce::AudioProcessorValueTreeState::SliderAttachment>
fmDepthSliderAttachment;
    juce::Label fmDepthLabel{ "FM Depth", "FM Depth" };

    void setSliderParams(juce::Slider& slider, juce::Label& label,
juce::AudioProcessorValueTreeState& apvts,
        juce::String paramId,
std::unique_ptr<juce::AudioProcessorValueTreeState::SliderAttachment>& attachment);

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (OscFront)
};
/*
  ==============================================================================

    AdsrFront.cpp
    Created: 25 May 2022 2:42:58pm
    Author:  Илья
```

```
  ==============================================================================
*/

#include <JuceHeader.h>
#include "AdsrFront.h"

//==============================================================================
AdsrFront::AdsrFront(juce::AudioProcessorValueTreeState& apvts, juce::String attackId,
juce::String decayId, juce::String sustainId
    , juce::String releaseId)
{
    attackAttachment =
std::make_unique<juce::AudioProcessorValueTreeState::SliderAttachment>
        (apvts, attackId, attackSlider);
    decayAttachment =
std::make_unique<juce::AudioProcessorValueTreeState::SliderAttachment>
        (apvts, decayId, decaySlider);
    sustainAttachment =
std::make_unique<juce::AudioProcessorValueTreeState::SliderAttachment>
        (apvts, sustainId, sustainSlider);
    releaseAttachment =
std::make_unique<juce::AudioProcessorValueTreeState::SliderAttachment>
        (apvts, releaseId, releaseSlider);

    setSliderParams(attackSlider, attackSliderLabel);
    setSliderParams(decaySlider, decaySliderLabel);
    setSliderParams(sustainSlider, sustainSliderLabel);
    setSliderParams(releaseSlider, releaseSliderLabel);


}

AdsrFront::~AdsrFront()
{
}

void AdsrFront::paint (juce::Graphics& g)
{

}

void AdsrFront::resized()
{
    const auto bounds = getLocalBounds().reduced(10);
    const auto padding = 10;
    const auto sliderWidth = bounds.getWidth() / 4 - padding;
    const auto sliderHeight = bounds.getHeight();
    const auto sliderStartX = 0;
    const auto sliderStartY = 0;

    attackSlider.setBounds(sliderStartX, sliderStartY, sliderWidth, sliderHeight);
    decaySlider.setBounds(attackSlider.getRight() + padding, sliderStartY,
sliderWidth, sliderHeight);
    sustainSlider.setBounds(decaySlider.getRight() + padding, sliderStartY,
sliderWidth, sliderHeight);
    releaseSlider.setBounds(sustainSlider.getRight() + padding, sliderStartY,
sliderWidth, sliderHeight);
}
void AdsrFront::setSliderParams(juce::Slider& slider, juce::Label& label)
{
    slider.setSliderStyle(juce::Slider::SliderStyle::LinearVertical);
    slider.setTextBoxStyle(juce::Slider::TextBoxBelow, true, 50, 25);
    addAndMakeVisible(slider);
```

```
    label.setColour(juce::Label::ColourIds::textColourId, juce::Colours::white);
    label.setFont(15.0f);
    label.setJustificationType(juce::Justification::centred);
    addAndMakeVisible(label);
}
/*
  ==============================================================================

    AdsrFront.h
    Created: 25 May 2022 2:42:58pm
    Author:  Илья

  ==============================================================================
*/

#pragma once

#include <JuceHeader.h>

//==============================================================================
/*
*/
class AdsrFront  : public juce::Component
{
public:
    AdsrFront(juce::AudioProcessorValueTreeState& apvts, juce::String attackId,
juce::String decayId, juce::String sustainId
        , juce::String releaseId);
    ~AdsrFront() override;

    void paint (juce::Graphics&) override;
    void resized() override;

private:
    void setSliderParams(juce::Slider& slider, juce::Label& label);

    juce::Slider attackSlider;
    juce::Slider decaySlider;
    juce::Slider sustainSlider;
    juce::Slider releaseSlider;

    juce::Label attackSliderLabel{ "A", "A" };
    juce::Label decaySliderLabel{ "D", "D" };
    juce::Label sustainSliderLabel{ "S", "S" };
    juce::Label releaseSliderLabel{ "R", "R" };

    std::unique_ptr<juce::AudioProcessorValueTreeState::SliderAttachment>
attackAttachment;
    std::unique_ptr<juce::AudioProcessorValueTreeState::SliderAttachment>
decayAttachment;
    std::unique_ptr<juce::AudioProcessorValueTreeState::SliderAttachment>
sustainAttachment;
    std::unique_ptr<juce::AudioProcessorValueTreeState::SliderAttachment>
releaseAttachment;

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (AdsrFront)
};

/*
  ==============================================================================

    FilterFront.cpp
    Created: 27 May 2022 12:07:18am
```

```
    Author:  Илья

  ==============================================================================
*/

#include <JuceHeader.h>
#include "FilterFront.h"

//==============================================================================
FilterFront::FilterFront(juce::AudioProcessorValueTreeState& apvts, juce::String
filterSelectId, juce::String filterCutoffId,
    juce::String filterResonaceId)
{
    juce::StringArray choices{ "LowPass", "HightPass", "BandPass" };
    filterSelect.addItemList(choices, 1);
    addAndMakeVisible(filterSelect);

    fillterSelectAttachment =
std::make_unique<juce::AudioProcessorValueTreeState::ComboBoxAttachment>(apvts,
filterSelectId, filterSelect);

    setSliderParams(filterCutoffSlider, filterCutoffLabel, apvts, filterCutoffId,
filterCutoffSliderAttachment);
    setSliderParams(filterResonaceSlider, filterResonaceLabel, apvts,
filterResonaceId, filterResonaceSliderAttachment);
}

FilterFront::~FilterFront()
{
}

void FilterFront::paint (juce::Graphics& g)
{

}

void FilterFront::resized()
{
    filterSelect.setBounds(getWidth() / 2 - 45, 5, 90, 20);

    filterCutoffSlider.setBounds(0, 45, 100, 90);
    filterCutoffLabel.setBounds(filterCutoffSlider.getX(), filterCutoffSlider.getY() -
20, filterCutoffSlider.getWidth() + 10, 20);

    filterResonaceSlider.setBounds(filterCutoffSlider.getRight(), 45, 100, 90);
    filterResonaceLabel.setBounds(filterResonaceSlider.getX(),
filterResonaceSlider.getY() - 20, filterResonaceSlider.getWidth(), 20);
}

void FilterFront::setSliderParams(juce::Slider& slider, juce::Label& label,
juce::AudioProcessorValueTreeState& apvts,
    juce::String paramId,
std::unique_ptr<juce::AudioProcessorValueTreeState::SliderAttachment>& attachment)
{
    slider.setSliderStyle(juce::Slider::SliderStyle::Rotary);
    slider.setTextBoxStyle(juce::Slider::TextBoxBelow, true, 50, 25);
    addAndMakeVisible(slider);
    attachment =
std::make_unique<juce::AudioProcessorValueTreeState::SliderAttachment>(apvts, paramId,
slider);

    label.setColour(juce::Label::ColourIds::textColourId, juce::Colours::white);
    label.setFont(15.0f);
    label.setJustificationType(juce::Justification::centred);
```

```
    addAndMakeVisible(label);
}

/*
  ==============================================================================

    FilterFront.h
    Created: 27 May 2022 12:07:18am
    Author:  Илья

  ==============================================================================
*/

#pragma once

#include <JuceHeader.h>

//==============================================================================
/*
*/
class FilterFront  : public juce::Component
{
public:
    FilterFront(juce::AudioProcessorValueTreeState& apvts, juce::String
filterSelectId, juce::String filterCutoffId,
        juce::String filterResonaceId);
    ~FilterFront() override;

    void paint (juce::Graphics&) override;
    void resized() override;

private:
    juce::ComboBox filterSelect;
    std::unique_ptr<juce::AudioProcessorValueTreeState::ComboBoxAttachment>
fillterSelectAttachment;

    juce::Slider filterCutoffSlider;
    std::unique_ptr<juce::AudioProcessorValueTreeState::SliderAttachment>
filterCutoffSliderAttachment;
    juce::Label filterCutoffLabel{ "Filter Cutoff", "Filter Cutoff" };

    juce::Slider filterResonaceSlider;
    std::unique_ptr<juce::AudioProcessorValueTreeState::SliderAttachment>
filterResonaceSliderAttachment;
    juce::Label filterResonaceLabel{ "Filter Res", "Filter Res" };

    void setSliderParams(juce::Slider& slider, juce::Label& label,
juce::AudioProcessorValueTreeState& apvts,
        juce::String paramId,
std::unique_ptr<juce::AudioProcessorValueTreeState::SliderAttachment>& attachment);

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (FilterFront)
};

/*
  ==============================================================================

    This file contains the basic framework code for a JUCE plugin editor.

  ==============================================================================
*/

#include "PluginProcessor.h"
#include "PluginEditor.h"
```

```
//==============================================================================
ClassSynthAudioProcessorEditor::ClassSynthAudioProcessorEditor(ClassSynthAudioProcesso
r& p)
    : AudioProcessorEditor(&p), audioProcessor(p), osc(audioProcessor.apvts, "OSC",
"FMFREQ", "FMDEPTH")
    , adsr(audioProcessor.apvts, "ATTACK", "DECAY", "SUSTAIN", "RELEASE")
    , filter(audioProcessor.apvts, "FILTER", "FILTERCUTOFF", "FILTERRES")
    , modAdsr(audioProcessor.apvts, "MODATTACK", "MODDECAY", "MODSUSTAIN",
"MODRELEASE")
{
    setSize (400, 300);
    addAndMakeVisible(osc);
    addAndMakeVisible(adsr);
    addAndMakeVisible(filter);
    addAndMakeVisible(modAdsr);
}

ClassSynthAudioProcessorEditor::~ClassSynthAudioProcessorEditor()
{
}

//==============================================================================
void ClassSynthAudioProcessorEditor::paint (juce::Graphics& g)
{
    g.fillAll(juce::Colours::darkslategrey);
}

void ClassSynthAudioProcessorEditor::resized()
{
    filter.setBounds(0, getHeight() / 2, getWidth() / 2, getHeight() / 2);
    osc.setBounds(0, 0, getWidth() / 2, getHeight() / 2);
    adsr.setBounds(getWidth() / 2 , 10, getWidth()/2, getHeight()/2);
    modAdsr.setBounds(getWidth() / 2 , 160, getWidth() / 2, getHeight() / 2);

}

/*
  ==============================================================================

    This file contains the basic framework code for a JUCE plugin editor.

  ==============================================================================
*/

#pragma once

#include <JuceHeader.h>
#include "PluginProcessor.h"
#include "FrontEnd/AdsrFront.h"
#include "FrontEnd/OscFront.h"
#include "FrontEnd/FilterFront.h"

//==============================================================================
/**
*/
class ClassSynthAudioProcessorEditor  : public juce::AudioProcessorEditor
{
public:
    ClassSynthAudioProcessorEditor (ClassSynthAudioProcessor&);
    ~ClassSynthAudioProcessorEditor() override;

    //==============================================================================
    void paint (juce::Graphics&) override;
```

```
    void resized() override;

private:
    ClassSynthAudioProcessor& audioProcessor;
    OscFront osc;
    AdsrFront adsr;
    FilterFront filter;
    AdsrFront modAdsr;


    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (ClassSynthAudioProcessorEditor)
};
```