СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 6](#_Toc483899321)

[**1** ОБЗОР ЛИТЕРАТУРЫ 8](#_Toc483899322)

[**1.1** Цифровые аудио-станции и плагины 8](#_Toc483899323)

[**1.2** Octaver 10](#_Toc483899324)

[**1.3** Delay 13](#_Toc483899325)

[**1.4** Reverb 14](#_Toc483899326)

[**2** СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ 16](#_Toc483899327)

[**2.1** Структура программного модуля 16](#_Toc483899328)

[**2.2** Выбор программных средств 17](#_Toc483899329)

[**3** ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 21](#_Toc483899330)

[**3.1** Блок интерфейса пользователя 21](#_Toc483899331)

[**3.2** Блок настройки параметров обработки 25](#_Toc483899332)

[**3.3** Блок преобразования входного сигнала для обработки 25](#_Toc483899333)

[**3.4** Блок эффекта «octaver» 25](#_Toc483899334)

[**3.5** Блок эффекта «delay» 25](#_Toc483899335)

[**3.6** Блок эффекта «reverb» 25](#_Toc483899336)

[**3.7** Блок преобразования обработанных данных в выходной сигнал 25](#_Toc483899337)

[**3.8** Блок эффекта «reverb» 25](#_Toc483899338)

[**4** РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 28](#_Toc483899339)

[**5** ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ 58](#_Toc483899340)

[**6** РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 63](#_Toc483899341)

[**6.1** Системные требования 63](#_Toc483899342)

[**6.2** Процесс инсталляции 63](#_Toc483899343)

[**6.3** Пользовательский интерфейс 67](#_Toc483899344)

[**6.3.1** Элементы управления эффекта Octaver 67](#_Toc483899345)

[**6.3.2** Элементы управления эффекта Delay 68](#_Toc483899346)

[**6.3.3** Элементы управления эффекта Reverb 69](#_Toc483899347)

[**7** ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ И РЕАЛИЗАЦИИ ПРОГРАММНОГО ПРОДУКТА 70](#_Toc483899348)

[**7.1** Характеристика программного продукта 70](#_Toc483899349)

[**7.2** Расчет сметы затрат на разработку и отпускной цены программного продукта 70](#_Toc483899350)

[**7.3** Расчёт экономического эффекта от реализации ПО 75](#_Toc483899351)

[**7.4** Расчет показателей экономической эффективности инвестиций в разработку и реализацию программного продукта 76](#_Toc483899352)

[ЗАКЛЮЧЕНИЕ 80](#_Toc483899353)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 81](#_Toc483899354)

[ПРИЛОЖЕНИЕ А 82](#_Toc483899355)

[ПРИЛОЖЕНИЕ Б 83](#_Toc483899356)

# ВВЕДЕНИЕ

Развитие вычислительной математики и техники привело к тому, что звуки, используемые человечеством для повседневной жизни, стали переводиться из аналоговой формы в цифровую. Цифровой звук имеет перед аналоговым огромные преимущества, хотя нельзя забывать и об определенных его недостатках. Основной ценностью цифрового звука является возможность бесконечно долгого хранения и бесконечного тиражирования материала без потери исходного качества, тогда как у аналогового звука качество теряется при каждой записи-перезаписи. Немаловажно и то, что цифровая техника, в отличие от аналоговой, позволяет добиваться идентичности параметров систем при их массовом производстве и сохраняет эту идентичность при эксплуатации, в то время как характеристики аналоговых изделий обычно отличаются на разных экземплярах и ухудшаются со временем. Кроме того, облегчаются передача звука и его обработка современными цифровыми средствами, в первую очередь, специализированными компьютерами.

Программные модули синтеза и обработки звука широко используются в уже давно популярных жанрах электронной музыки, создавая звуки, непривычные человеческому уху, а потому необычные, интересные. Но это только самое очевидное и, конечно же, не единственное приложение. Прогресс приводит к тому, что становится возможным создание модулей обработки звука, эмулирующих настоящие звукозаписывающие студии с любым оборудованием: педалями эффектов, гитарными усилителями и гитарными кабинетами, микрофонами, комнатами. Кроме того, каждый коммерческий и большинство некоммерческих музыкальных произведений проходит ряд обработок на стадии сведения и мастеринга, где исправляются некоторые дефекты записи, корректируются параметры для соответствия стандартам, добавляются эффекты и другое.

Ещё одним достоинством цифрового звука является возможность эмуляции любого аналогового прибора. В настоящее время любой реальный аналоговый прибор можно эмулировать с довольно высокой точностью, которая определяется в большей степени работой по сбору информации о приборе, нежели ограничениями компьютера. Более того, программные модули позволяют получить эффекты, которые невозможно создать с помощью аналоговых приборов. В настоящее время вычислительных ресурсов даже рядовых персональных компьютеров достаточно, чтобы запускать сразу несколько десятков программных модулей обработки звука в реальном времени.

Прогресс электротехники достиг такого уровня, что недостатки цифрового звука становятся ничтожными и меркнут перед достоинствами. Ведь уже несколько лет изготавливаются и широкодоступны АЦП и ЦАП с параметрами достаточными для того, чтобы разница между исходным аналоговым и оцифрованным звуком не была заметна человеческому уху.

Гибкость, удобство, универсальность, дешевизна программного обеспечения для обработки звука и хорошее качество цифрового звука практически не оставляют причин музыкантам, звукорежиссёрам и мастеринг-инженерам не переходить с аналогового оборудования на цифровое.

Таким образом, исходя из вышеперечисленных достоинств цифрового звука и его цифровой обработки, становится понятным, что создание программного обеспечения для обработки звука является актуальной темой. Этот факт, а также моя любовь к музыке обусловили выбор темы дипломного проекта. Цель проекта: разработать программный модуль обработки звуковой дорожки в реальном времени.

# ОБЗОР ЛИТЕРАТУРЫ

## Цифровые аудио-станции и плагины

Цифровая звуковая рабочая станция – электронная или компьютерная система, предназначенная для записи, хранения, редактирования и воспроизведения цифрового звука. Предусматривает возможность выполнения на ней законченного цикла работ, от первичной записи до получения готового результата. Плагины обычно выполняются в виде разделяемых библиотек [1]. Наиболее часто плагины для обработки звука используются в хост-программе DAW (Digital Audio Workstation – цифровая звуковая рабочая станция), где они применяются к отдельным дорожкам или их группам. Примеры наиболее популярных DAW с кратким описанием в таблице 1.1.

|  |  |
| --- | --- |
| Таблица 1.1 – Примеры и описание наиболее популярных DAW | |
| DAW | Описание |
| FL Studio | Цифровая звуковая рабочая станция (DAW) и секвенсер для написания музыки. Музыка создаётся путём записи и сведения аудио- или MIDI-материала. |
| Cubase | Программное обеспечение для создания, записи и микширования музыки. Обладает мощным аудиодвижком и встроенными профессиональными инструментами |
| Nuendo | Отличается от другой популярной линии программных продуктов — Steinberg Cubase тем, что ориентирована не только на музыкантов, но и на любую профессиональную деятельность по озвучиванию в том числе фильмов, телевизионных программ, реклам, радиопередач и прочего. |
| Ableton Live | Программа используется как для студийной работы (аранжировка, сведение), так и для живой игры (импровизация, DJ-инг), и имеет два режима: «Arrangement View» и «Session View». |
| REAPER | Аудиостанция обладает широкими функциональными возможностями и является развитой, профессиональной рабочей системой для создания, записи, редактирования и микширования аудио и MIDI материала, а также мастеринга композиций. При этом программа имеет относительно небольшие размеры. |
| Logic Pro X | Logic Pro X включает огромную коллекцию высококачественных музыкальных сэмплов, инструментов, эффектов и циклов — всё, что нужно для создания композиций профессионального уровня. |

В программах обработки звука плагины выполняют обработку и создание звуковых эффектов, например, мастеринг, применение эквалайзера и сжатие динамического диапазона [2]. Некоторые плагины изменяют технические характеристики звука: глубину, частоту дискретизации и прочее. Практически все аудио-плагины имеют графический пользовательский интерфейс. В GUI (graphical user interface – графический интерфейс пользователя) присутствуют элементы управления (чаще всего это какие-нибудь ручки), меняющие то, как плагин обрабатывает входящие данные. Часто плагин или standalone (автономное) приложение уже имеет встроенный набор пресетов (от английского preset – сохранённый набор настроек), в которых хранятся положения ручек и других параметров. Также часто имеется возможность сохранять свои собственные пресеты. Наиболее распространённый формат аудио-плагинов – VST.

Обычно виды обработок аудио дорожки классифицируют образом, отражённым в таблице 1.2.

|  |  |
| --- | --- |
| Таблица 1.2 – Типы и виды обработок звука | |
| Тип обработки | Вид обработки |
| Частотная | * filter, equalizer, pitchshifter. |
| Динамическая | noise gate, compressor, limiter, soft clipper, expander. |
| Модуляционная | * chorus, flanger, phaser. |
| Частотно-динамическая | multiband compressor. |
| Пространственная | delay, echo, reverb. |
| Искажения | overdrive, distortion, fuzz. |

Программные модули для обработки звука могут работать либо со стерео, либо с моно дорожкой, либо и с той и другой.

Virtual Studio Technology (VST) — формат ресурсозависимых (native) плагинов реального времени, которые подключаются к звуковым редакторам и музыкальным редакторам, секвенсорам. Формат был разработан совместно Propellerhead и Steinberg, впоследствии Propellerhead отказался от дальнейших работ над VST и дальнейшая разработка осуществлялась исключительно Steinberg. В настоящее время в этом формате существуют тысячи плагинов, он стал одним из самых распространённых для звуковых программ. Приложения VST отличаются от плагинов DirectX по нескольким параметрам; в частности, они существуют для Windows, Mac OS X и Linux. Кроме того, в отличие от ранних версий DirectX, плагины VST обладают развитым интерфейсом автоматизации [3].

В данном дипломном проекте разрабатывается программный модуль обработки моно и стерео дорожек с использованием следующих основных эффектов: delay и reverb. Дополнительной задачей является разработка эффекта octaver.

## Octaver

Octaver – звуковой эффект или соответствующее устройство, добавляющее к сигналу его копию на октаву или две ниже или выше основного тона. Само обозначение применяется преимущественно к обработке звука электрогитары [4].

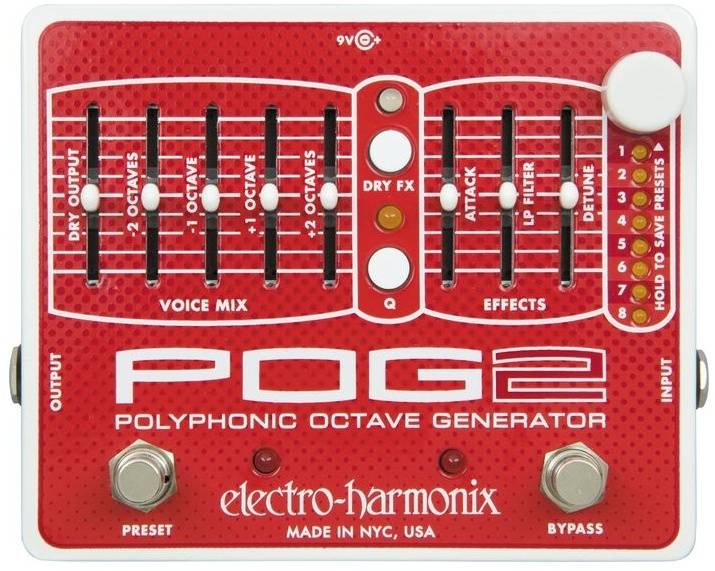


Рисунок 1.1 – Electro-Harmonix POG 2

Octaver является частным случаем эффекта pitch shifter (сдвиг тона). При использовании данного эффекта создаётся впечатление, что вместо одного инструмента играют два в разных октавах. В некоторых педалях эффектов, таких, как, например, Boss OC-2, имеется возможность добавлять два дополнительных тона одновременно – один на октаву ниже основного, другой на две октавы ниже основного. При этом каждый из них можно регулировать по громкости, смешивая в желаемой пропорции. Большинство педалей эффектов типа octaver монофонические, то есть не могут достраивать октавы к аккордам. Полифонические (цифровые) эффекты могут обрабатывать сигнал, содержащий несколько нот. Примером цифрового эффекта octaver является педаль Electro-Harmonix POG 2 (рисунок 1.1).

Для реализации этого эффекта необходимо получить представление звукового сигнала в частотной области, затем обработать его и перевести во временную область.

Дискретное преобразование Фурье (ДПФ) – одна из двух наиболее распространённых и мощных процедур цифровой обработки сигналов (другая процедура – цифровая фильтрация). ДПФ позволяет анализировать, преобразовывать и синтезировать сигналы такими способами, которые невозможны при непрерывной (аналоговой) обработке [5].

ДПФ – это математическая процедура, используемая для определения гармонического или частотного состава дискретных сигналов. Истоком ДПФ является непрерывное преобразование Фурье X(f), которое определяется как

|  |  |
| --- | --- |
| , | (1.1) |

где – некоторый непрерывный сигнал во временной области.

С появлением и развитием цифровых вычислительных машин, работающих с дискретным представлением сигналов, было разработано ДПФ. Оно определяется как дискретная последовательность X(m) в частотной области:

|  |  |
| --- | --- |
| , | (1.2) |

где – дискретная последовательность значений, полученных дискретизацией во временной области непрерывной переменной ;

– основание натурального логарифма;

– константа;

– мнимая единица.

Точные значения частоты разных синусоид зависят как от частоты дискретизации , с которой был дискретизирован исходный сигнал, так и от количества отсчётов N. Все частоты, соответствующие , кратны основной частоте. N разных частот анализа ДПФ определяются выражением

|  |  |
| --- | --- |
|  | (1.3) |

Практический интерес представляют амплитуда и фаза каждого отсчёта . Если представить произвольный отсчёт ДПФ как сумму действительной и мнимой частей

|  |  |
| --- | --- |
| , | (1.4) |

то амплитуда вычисляется как

|  |  |
| --- | --- |
| , | (1.5) |

а фазовый угол вычисляется как

|  |  |
| --- | --- |
| . | (1.6) |

Чтобы получить исходный сигнал во временной области необходимо выполнить обратное дискретное преобразование Фурье (ОДПФ). Выражение для ОДПФ имеет вид:

|  |  |
| --- | --- |
| . | (1.7) |

В 1965 году была опубликована статья Кули и Тьюки [6], описывающая эффективный алгоритм реализации ДПФ. Этот алгоритм сегодня известен как быстрое преобразование Фурье (БПФ).

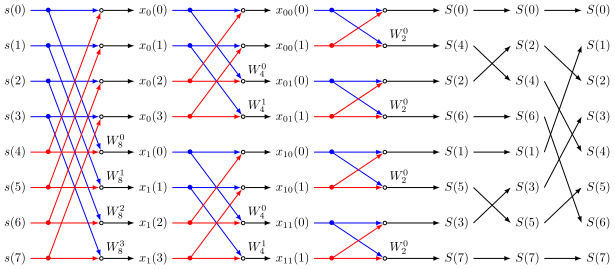


Рисунок 1.2 – Полный граф алгоритма БПФ с прореживанием по частоте для

Алгоритм БПФ по основанию 2 – это эффективный алгоритм вычисления ДПФ, когда длина ДПФ равна натуральной степени двух. Из-за специфической формы элементарных элементов графа он получил название «бабочка». Граф состоит из бабочек. Данная процедура объединения является основной при построении алгоритмов БПФ по основанию два. На рисунке 1.2 представлен полный граф алгоритма БПФ с прореживанием по частоте для N=8 [7].

Операция бабочки в данном алгоритме выполняется в соответствии с формулами 1.8 и 1.9.

|  |  |
| --- | --- |
| ; | (1.8) |

|  |  |
| --- | --- |
| . | (1.9) |
|  |  |

## Delay

Delay – звуковой эффект или соответствующее устройство, имитирующие чёткие затухающие повторы (эхо) исходного сигнала. Эффект реализуется добавлением к исходному сигналу его копии или нескольких копий, задержанных по времени. Под термином delay обычно подразумевается однократная задержка сигнала, в то время как эффект «эхо» - многократные повторы. По принципу действия является частным случаем ревербератора. Отличие заключается в том, что delay имеет одну линию задержки и больший временной интервал (не менее 50-60 мс), который позволяет отделить оригинальный звук от эффекта на слух [8].

Delay – довольно простой эффект для реализации. Входной сэмпл (от английского sample – элемент выборки, замер) сигнала сохраняется в буфер и суммируется с прошлым сэмплом, отстающим в буфере на некоторое число N сэмплов. Обычно прошлый сэмпл ослабляется. И ослабление и длина задержки (N) обычно управляются пользователем [9]. На рисунке 1.3 изображена схема простого эффекта delay.

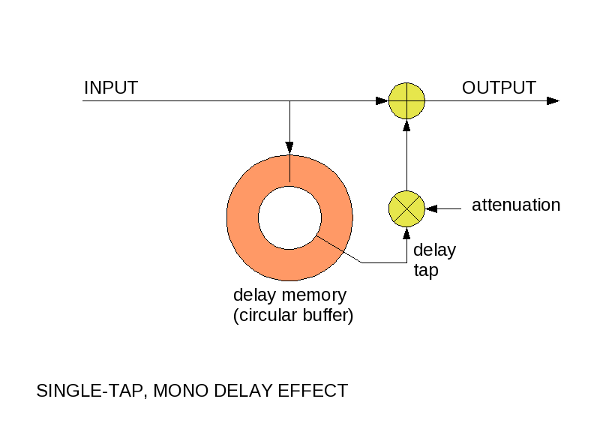


Рисунок 1.3 – Схема простого эффекта delay

Параметры эффекта:

* обратная связь (feedback, regeneration) – при отсутствии обратной связи на выходе будет одна задержка, при увеличении её значения растёт и количество сигналов на выходе;
* время задержки (delay, time) – промежуток времени между исходным сигналом и его задержкой (задержками);
* баланс (balance, mix) – соотношение исходного и задержанного сигналов.

Основные типы эффекта delay:

* slapback – одиночная задержка длительностью до 120 мс;
* echo – более длительная задержка с обратной связью;
* reverse – в цифровых эффектах возможно воспроизведение записанного в буфер сигнала в обратном порядке;
* ping pong – поочерёдное панорамирование задержанного сигнала в левый/правый канал.

В данном проекте реализуется эффект delay, с изменяемой задержкой в пределах от 20 до 1000 мс, с возможностью синхронизации с темпом музыкальной композиции.

## Reverb

Реверберация – это процесс постепенного уменьшения интенсивности звука при его многократных отражениях. Иногда под реверберацией понимается имитация данного эффекта с помощью ревербераторов [10].

Явление реверберации состоит в суперпозиции различных эхосигналов от одного источника звука. Эффект реверберации можно наблюдать в закрытых помещениях после выключения источника звука. Обычно избыточная длительность реверберации приводит к неприятной гулкости, «пустоте» помещения, а недостаточная – к резкому отрывистому звучанию, лишённому музыкальной полноты. Искусственно создаваемая реверберация в определённых пределах способствует улучшению качества звучания, создавая ощущение приятного «резонанса» помещения.

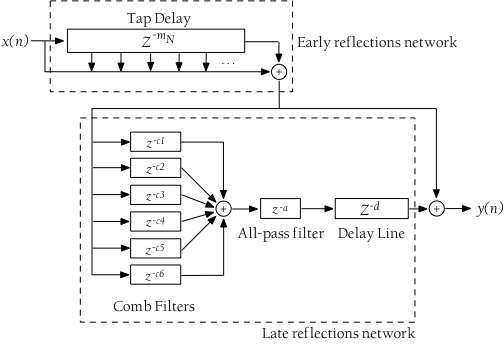


Рисунок 1.4 – Структурная схема ревербератора Мурера

Реверберацию можно разделить на два компонента, которые видны на представлении импульсного отклика в комнате:

1. Ранние отражения – первые отражения, которые мы слышим в течение приблизительно 100 мс вместе с прямым звуком от источника.
2. Поздняя реверберация – слышимый звук после 100 мс, до момента его полного затухания. Поздняя реверберация характеризуется плотной текстурой рассеянных отражений, которые достигают наших ушей несколькими путями. Эти рассеянные отражения находятся не в фазе относительно друг друга, что вызывает эффект гребенчатой фильтрации. Мы воспринимаем этот эффект как «атмосферу», характер места [11].

Структурная схема обработки сигнала ревербератором показана на рисунке 1.4. Такое решение предложил звуковой инженер, музыкант и доктор наук Стэнфордского университета Джеймс Энди Мурер, усовершенствовав алгоритм Шрёдера [12]. Исходя из Алгоритма Мурера, к исходному сигналу добавляется сигнал, прошедший через сеть ранних отражений, состоящую из множества delay -линий. Далее к нему добавляется сигнал обработанный сетью поздних отражений, состоящей из параллельных гребенчатых фильтров и фазового фильтра с delay-линией.

Гребенчатый фильтр — в обработке сигналов электронный фильтр, при прохождении сигнала через который к нему добавляется он сам с некоторой задержкой. В результате получается фазовая компенсация. АЧХ гребенчатого фильтра состоит из ряда равномерно распределённых пиков, так что она выглядит как гребёнка.

В цифровых системах, фильтр задаётся формулой 1.10:

|  |  |
| --- | --- |
| , | (1.10) |

где – коэффициенты;

– запаздывание сигнала;

– входной сигнал;

– выходной сигнал.

На данный момент существуют аналоги отдельных эффектов, реализуемых в данном дипломном проекте, как бесплатные, так и довольно дорогие. Комбинация этих эффектов создаёт новый, необычный, слабо представленный на рынке эффект. В определённых кругах этот эффект называют эффектом shimmer (от английского – мерцание). Целью данного дипломного проекта является создание программного модуля эффекта shimmer с возможностью настройки каждого из подкомпонентов в отдельности.

# СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Разрабатываемый программный модуль разбит на отдельные логически взаимосвязанные блоки, что является необходимым условием для обеспечения гибкости его структуры. При данном подходе становится возможной выборочная модернизация отдельных частей программного кода, с минимальным влиянием на остальные части проекта, либо, в идеальном случае, вовсе без их изменения.

## Структура программного модуля

В соответствии с методологией системного подхода в разработке архитектуры, программный модуль разбивается на совокупность сущностей, представленных на структурной схеме (см. чертеж ГУИР.400201.161 С1).

Структурная схема была составлена исходя из основных стадий обработки сигнала, а также функций, которые должен предоставлять программный модуль обработки звуковой дорожки в реальном времени. Структурная схема состоит из следующих блоков:

* блок преобразования входного сигнала для обработки;
* блок эффекта «octaver»;
* блок эффекта «delay»;
* блок эффекта «reverb»;
* блок преобразования обработанных данных в выходной сигнал;
* блок настройки параметров обработки;
* блок интерфейса пользователя.

*Блок преобразования входного сигнала для обработки* принимает исходный сигнал. Перед передачей сигнала к блокам добавления эффектов в нём устраняются шумы. Это выполняется за счёт фильтра нижних частот, фильтра верхних частот и noise gate. Также корректируется уровень входного сигнала для обработки последующими блоками.

*Блок эффекта «*octaver*»* принимает уже подготовленный к обработке звуковой сигнал. Блок предназначен для добавления гармоник частотой степени двойки к исходному звуковому сигналу. Внутри блока с помощью ДПФ получается сигнал в частотном домене. Далее сигнал обрабатывается и с помощью ОДПФ переводится во временной домен. Затем обработанный сигнал складывается с исходным сигналом в задаваемой пользователем пропорции и подаётся на следующий логический блок.

*Блок эффекта «*delay*»* создаёт копию поступившего на вход сигнала и добавляет её к исходному через некоторое задаваемое пользователем время. Сигнал с выхода снова подаётся на вход этого блока уже с уменьшенным уровнем. Внутри этого блока также происходит обработка сигнала, подаваемого с выхода на вход. С выхода блока эффекта «delay» сигнал поступает на блок эффекта «reverb».

*Блок эффекта «*reverb*»* отвечает за эмуляцию постепенного уменьшения интенсивности звука при его многократных отражениях. Он включает в себя два компонента: блок ранних отражений и блок поздних отражений.

*Блок преобразования обработанных данных в выходной сигнал* стоит последним в тракте следования сигнала. Он включает в себя фильтры низких и высоких частот. Также этот блок отвечает за уровень сигнала, подаваемого на выход.

*Блок интерфейса пользователя* представляет собой совокупность средств, при помощи которых пользователь взаимодействует с программным модулем обработки звуковой дорожки в реальном времени. Для построения интерфейса используются компоненты, встроенные в фреймворк Juce (о фреймворке Juce написано в подразделе 2.2). Данный блок отвечает за получение ввода пользователя и связан лишь с блоком настройки параметров обработки двусторонней связью.

*Блок настройки параметров обработки* предназначен для преобразования ввода пользователя и хранения параметров, влияющих на работу блоков эффектов. Среди таких параметров обязательно будут присутствовать следующие:

* уровень входного сигнала (input level) для блока преобразования входного сигнала для обработки;
* соотношение обработанного сигнала к необработанному (mix) для всех эффектов;
* время задержки (delay time) для эффектов delay и reverb;
* уровень сигнала подаваемого с выход на вход (feedback) для эффекта delay;
* уровень выходного сигнала (output level) для блока преобразования обработанных данных в выходной сигнал.

Данный блок кроме двусторонней связи с блоком интерфейса пользователя имеет исходящие двусторонние связи с каждым из блоков эффектов обработки звуковой дорожки, а также блоками преобразования входного сигнала для обработки и блоком преобразования обработанных данных в выходной сигнал.

## Выбор программных средств

В качестве языка программирования задан язык C++, а в качестве среды разработки – Visual Studio 2012 Express. Для выполнения поставленной задачи необходимо выбрать фреймворк, который совместим с требованиями задания, позволяет избежать низкоуровневого программирования, включающий компоненты графического интерфейса пользователя, а также компоненты для работы со звуком. Фреймворк Juce удовлетворяет всем вышеперечисленным требованиям.

Juce – это открытый кроссплатформенный инструментарий разработки ПО (фреймворк) для языка C++, используемый для разработки GUI приложений и плагинов [13].

Цель Juce – позволить компилировать один и тот же исходный текст в программы, одинаково работающие на Windows, Mac OS X и Linux (последние версии – также iPhone и Android) платформах. Он поддерживает различные среды разработки и компиляторы, такие как GCC, Xcode и Visual Studio.

Juce впервые опубликован в 2004, держателем его кода является британская компания Raw Material Software. Имеет двойную GPL/коммерческую лицензию.

Juce содержит классы, позволяющие программе работать со звуком и графикой. За счёт этого нуждающиеся в дополнительных библиотеках программисты могут использовать только библиотеку Juce, или хотя бы сократить количество сторонних библиотек, которые они используют. На это разработчиков Juce вдохновил JDK языка Java. По их заявлению, они собирались из Juce сделать «что-то подобное для С++».

Наиболее важная особенность Juce по сравнению с другими аналогичными фреймворками — большой набор аудиофункций. Дело в том, что библиотека Juce сначала была разработана как часть аудиосеквенсора Tracktion, и лишь затем стала самостоятельным продуктом. Juce включает в себя поддержку воспроизведения звука через аудио и MIDI интерфейсы, полифонические синтезаторы, понимает файлы распространённых аудиоформатов (таких как WAV, AIFF, FLAC, и Vorbis). Он также содержит интерфейсы-оболочки для построения различных аудио плагинов, таких как эффекты и инструменты VST. Это привело к его широкому распространению в сообществе разработчиков аудио-ПО.

В поставку Juce входят классы-обёртки для создания аудиоплагинов. При сборке аудиоплагина, получается единый бинарный файл, который поддерживает несколько форматов: VST, RTAS, AU. Поскольку весь платформо- и форматозависимый код содержится в классах-обёртках, то пользователь может собирать плагины в формате VST/RTAS/AU для макинтошей и Windows из одного и того же исходного кода.

Имеется также неофициальное ответвление библиотеки, расширенное дополнительными возможностями, поддерживаемое сообществом, которое называется Juced.

На сайте этого варианта фреймворка можно найти также дополнительную документацию по Juce, которая поможет освоить библиотеку [14]. Последняя версия фреймворка на данный момент – 4.3.

Фреймворк Juce имеет встроенную хост-программу Juce Plug-In Host [15]. Она имеет только самый необходимый функционал для хост-программы и требует небольшого объёма вычислительных ресурсов компьютера, что позволяет быстро проверять работу плагина. Кроме того с её помощью можно производить отладку плагина, что очень важно для программиста. Чтобы разрабатываемый плагин обрабатывал входной сигнал необходимо соединить блок Audio Input и Midi Input с входами плагина, а выходы плагина соединить с блоком Audio output. К входам разрабатываемого плагина также подключён плагин AudioFilePlayer, который используется для проигрывания записанной звуковой дорожки. Интерфейс программы Juce Plug-In Host изображён на рисунке 2.1.

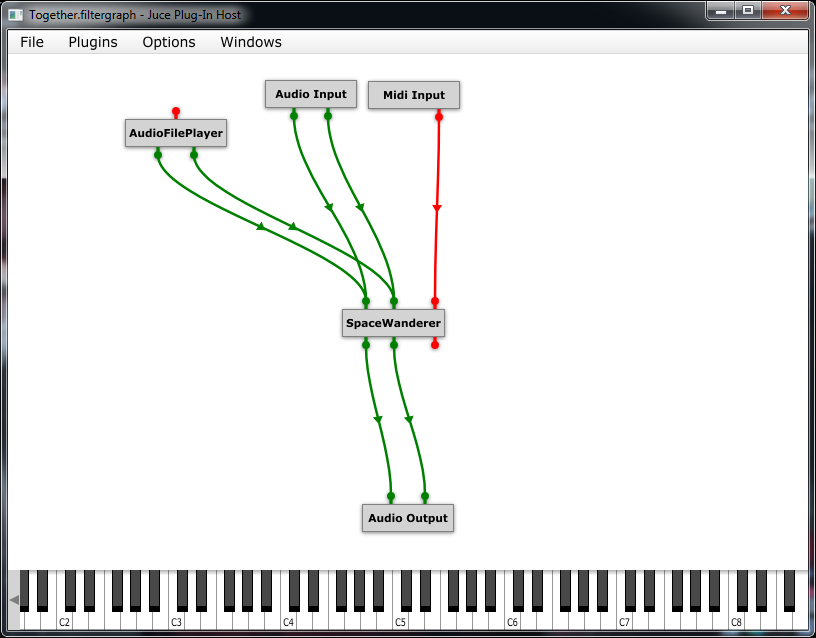


Рисунок 2.1 – Интерфейс хост-программы Juce

Для разработки VST плагинов необходимо скачать пакет VST SDK с официального сайта Steinberg – разработчика этого формата. Текущая версия этого средства разработки – 3.6.7.

Также при разработке программного модуля обработки звуковой дорожки в реальном времени будет использоваться система управления версиями – Git. Выбор сделан в пользу именно этой системы управления версиями из-за ряда причин:

1. Git использует распределённую модель вместо традиционной клиент серверной. Она не нуждается в централизованном хранилище: вся история изменения документов хранится на каждом компьютере в локальном хранилище. При необходимости локальный репозиторий может быть передан командой push на удалённый репозиторий.
2. Постоянное подключение к сети не является необходимым.
3. Git обладает высокой производительностью.
4. Продуманная и уже знакомая мне система команд.

В качестве сервиса предоставляющего хостинг для размещения git-репозиториев выбран веб-сайт GitHub [16].

Таким образом, в данном дипломном проекте при разработке программного модуля обработки звуковой дорожки в реальном времени

* используется модульное программирование;
* в качестве фреймворка выбрана библиотека Juce;
* в качестве формата ресурсозависимого плагина реального времени для DAW выбран VST;
* используется система контроля и управления версиями Git;
* удалённый репозиторий, содержащий пояснительную записку, чертежи, документы и проект программного модуля обработки звуковой дорожкой в реальном времени хранится на сайте GitHub по ссылке https://github.com/Andrewregrets/VST-Plugin.

# ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе подробно рассматривается функционирование программных модулей разработанного программного продукта. Раздел структурирован в соответствии с блоками, выделенными на этапе системного проектирования. В нём перечислены соответствующие классы и большинство их компонентов, состав и отношения которых показаны на диаграмме классов (см. чертёж ГУИР.400201.161 РР.1).

## Блок интерфейса пользователя

Основным классом, который представляет *блок интерфейса пользователя* является ShimmerAudioProcessorEditor (см. рисунок 3.1). Данный класс унаследован от класса AudioProcessorEditor, который предоставляет фреймворк Juce. Данный класс отвечает за обработку пользовательского ввода и отображение графической информации на экран.



Рисунок 3.1 – Организация блока интерфейса пользователя

Класс ShimmerAudioProcessorEditor имеет следующие поля:

* const int window\_width;
* const int window\_height;
* Image background\_image;fbdfnsdmn,gkj.;
* ShimmerAudioProcessor &processor;
* ScopedPointer<Label> InputGainLabel;
* ScopedPointer<Slider> InputGainKnob;
* ScopedPointer<Label> OutputGainLabel;
* ScopedPointer<Slider> OutputGainKnob;
* Octaver\_GUI\_Components octaver\_gui\_components;
* Delay\_GUI\_Components delay\_gui\_components;
* Reverb\_GUI\_Components reverb\_gui\_components.

Константные поля window\_width и window\_height инициализируются в конструкторе данного класса и содержат соответственно ширину и высоту окна.

Поле background\_image необходимо для отображения фонового изображения программного модуля.

Поле processor содержит ссылку на объект типа ShimmerAudioProcessor (см. рисунок 3.2). Это поле инициализируется при создании объектом типа ShimmerAudioProcessor объекта типа ShimmerAudioProcessorEditor и используется для доступа к параметрам обработки звука программным модуля.



Рисунок 3.2 – Агрегирование по ссылке класса ShimmerAudioProcessor классом ShimmerAudioProcessorEditor

Поля InputGainLabel, InputGainKnob, OutputGainLabel, OutputGainKnob – указатели на элементы графического интерфейса, не относящиеся ни к одному из компонентов программного модуля обработки звуковой дорожки в реальном времени. InputGainKnob и OutputGainKnob – ручки, отвечающие соответственно за регулировку входного и выходного уровней сигнала. InputGainLabel и OutputGainLabel – метки для отражения смысла этих ручек.

Класс ScopedPointer предоставлен фреймворком Juce. Данный класс содержит указатель, который автоматически удаляется, когда объект этого класса выходит за свою зону видимости, то есть данный класс следует идиоме RAII (от английского Resource Acquisition Is Initialization – получение ресурса есть инициализация), согласно которой получение ресурса неразрывно совмещается с инициализацией, а освобождение — с уничтожением объекта. RAII является ключевой концепцией для написания безопасного при исключениях кода в языках программирования, где конструкторы и деструкторы автоматических объектов вызываются автоматически, прежде всего – в C++.



Рисунок 3.3 – Компоненты графического интерфейса

Поля octaver\_gui\_components, delay\_gui\_components и reverb\_gui\_components содержат элементы графического интерфейса, относящиеся соответственно к эффектам Octaver, Delay и Reverb (см. рисунок 3.3). Каждый из классов этих объектов имеет конструктор, имеющий три параметра. Первый параметр – указатель на объект типа AudioProcessorEditor, необходим для вызова метода addAndMakeVisible(Component\*), который необходимо вызывать для всех компонентов графического интерфейса для их отображения. Второй параметр – указатель на объект типа ButtonListener. Он необходим, так как используется в качестве параметра функции addListener(ButtonListener\*), которая является членом класса TextButton и предназначена для задания объекта, ответственного за обработку событий связанных с объектом класса TextButton. Такими событиями могут быть, например, нажатие кнопки, наведение на кнопку указателя мыши. Третий параметр – указатель на объект типа SliderListener. Он необходим, так как используется в качестве аргумента функции addListener(SliderListener \*), которая является членом класса Slider и предназначена для задания объекта, ответственного за обработку событий связанных с объектом класса Slider. Примером таких событий является изменение значения путём кручения ручки.

Конструктор выделяет память для компонентов графического интерфейса, делает их видимыми, задаёт их параметры визуализации, то есть определяет то, как эти компоненты будут отображаться. В конструкторе также задаются граничные значения параметров, задаваемых ручками.

Кроме однотипного конструктора, классы Octaver\_GUI\_Components, Delay\_GUI\_Components и Reverb\_GUI\_Components имеют метод resized(Rectangle<int> r). Данный метод принимает в качестве параметра объект типа Rectangle<int>. Данный объект описывает прямоугольную область и в данном случае определяет область экрана, в которой будут отображаться компоненты графического интерфейса конкретного эффекта. Подробнее поля и методы этих классов представлены ниже.

Octaver\_GUI\_Components(AudioProcessorEditor\* ae, ButtonListener\* bl, SliderListener\* sl);

void resized(Rectangle<int> r);

ScopedPointer<Label> MixLabel;

ScopedPointer<Label> OctaveLowLabel;

ScopedPointer<Label> OctaveHighLabel;

ScopedPointer<Slider> MixKnob;

ScopedPointer<Slider> OctaveLowKnob;

ScopedPointer<Slider> OctaveHighKnob;

ScopedPointer<TextButton> BypassButton;

ScopedPointer<GroupComponent> groupComponent;

Компоненты графического интерфейса пользователя для эффекта Octaver содержат ручки типа Slider для изменения соотношения исходного сигнала к обработанному, уровня звука на октаву ниже основного и уровня звука на октаву выше основного. Также присутствует названия типа Label для соответствующих ручек, поле groupComponent типа GroupComponent для очерчивания границы компонента Octaver, а также кнопка BypassButton, предназначенная для перевода данного эффекта в пассивное состояние, в котором он не влияет на звук.

Delay\_GUI\_Components(AudioProcessorEditor\* ae, ButtonListener\* bl, SliderListener\* sl);

void resized(Rectangle<int> r);

ScopedPointer<GroupComponent> groupComponent;

ScopedPointer<Label> timecodeDisplayLabel;

ScopedPointer<Label> DelayLabel;

ScopedPointer<Label> MixLabel;

ScopedPointer<Label> FeedbackLabel;

ScopedPointer<Slider> MixKnob;

ScopedPointer<Slider> FeedbackKnob;

ScopedPointer<Slider> DelayKnob;

ScopedPointer<TextButton> BypassButton;

ScopedPointer<TextButton> SynchButton;

ScopedPointer<TextButton> DottedButton;

ScopedPointer<TextButton> SecondDottedButton;

Компоненты графического интерфейса пользователя для эффекта Delay содержат ручки и метки, поясняющие их функции. Поле timecodeDisplayLabel предназначено для отображения текущей позиции воспроизведения дорожки, а также темпа, которые принимаются из host-программы. Поле FeedbackKnob – ручка, задающее параметр обратной связи в эффекте Delay, то есть она определяет количество копий исходного сигнала, которые будут воспроизведены, с интервалом времени между повторами, задаваемым ручкой DelayKnob. Кнопка BypassButton переводит компонент эффекта Delay в пассивное состояние. Кнопка SynchButton предназначена для перевода компонента эффекта Delay в синхронный режим. в котором время между повторами основывается на темпе музыкальной композиции, задаваемом host-программой. Кнопка DottedButton будет добавлять половину ко времени между повторами, а кнопка SecondDotted – четверть.

Reverb\_GUI\_Components(AudioProcessorEditor\* ae, ButtonListener\* bl, SliderListener\* sl);

void resized(Rectangle<int> r);

ScopedPointer<Label> DecayLabel;

ScopedPointer<Label> MixLabel;

ScopedPointer<Slider> DecayKnob;

ScopedPointer<Slider> MixKnob;

ScopedPointer<TextButton> BypassButton;

ScopedPointer<GroupComponent> groupComponent;

Данный класс имеет поля BypassButton, groupComponent, MixKnob, назначение которых описано выше для классов компонентов графического интерфейса других блоков эффектов. Поле DecayKnob представляет собой ручку, регулирующую затухание эмулируемых блоком эффекта Reverb отражений.

Класс ShimmerAudioProcessorEditor имеет следующие основные методы:

* + - ShimmerAudioProcessorEditor (ShimmerAudioProcessor&);
    - ~ShimmerAudioProcessorEditor();
    - void paint (Graphics&) override;
    - void resized() override;
    - void sliderValueChanged (Slider\* sliderThatWasMoved) override;
    - void buttonClicked (Button\* buttonThatWasClicked) override;
    - void timerCallback();
    - const String setSyncLabel(int sl);
    - void updateTimecodeDisplay (AudioPlayHead::CurrentPositionInfo pos);

В конструкторе ShimmerAudioProcessorEditor()вызываются конструкторы компонентов графического интерфейса, задаются их параметры визуализации.

В конструкторе также вызываются следующие функции:

* + - setSize (window\_width,window\_height);
    - startTimer(200);
    - background\_imageImageCache::getFromMemory( Images::background\_jpg, Images::background\_jpgSize).

Функция setSize() устанавливает размер окна в соответствии с константными значениями полей window\_width и window\_height.

Функция startTimer() начинает отсчёт таймера. В качестве параметра данная функция принимает время в миллисекундах. По истечении этого времени будет вызвана функция timerCallback().

Функция getFromMemory() класса ImageCache загружает фоновое изображение из файла в памяти, или возвращает его, если изображение уже кэшировано. Фоновое изображение программного модуля обработки звуковой дорожки в реальном времени определяется в классе Images и задаётся статическим константным массивом типа unsigned char.

В конструкторе также задаются значения ручек в соответствии с параметрами блоков эффектов, которые находятся в объекте типа ShimmerAudioProcessor. Это осуществляется вызовом метода getParameter(), определённого в классе ShimmerAudioProcessor используя ссылку на объект processor данного типа.

Класс ShimmerAudioProcessor унаследован от следующих абстрактных классов (см. рисунок 3.4):

* AudioProcessorEditor;
* SliderListener;
* Timer;
* ButtonListener.



Рисунок 3.4 – Наследование класса ShimmerAudioProcessorEditor

Класс ShimmerAudioProcessorEditor является основным, как и класс ShimmerAudioProcessor, унаследованный от AudioProcessor. Два этих класса создаются автоматически при создании проекта и являются неотъемлемой его частью.

Основные виртуальные методы, которые включены в класс ShimmerAudioProcessor при создании проекта, и которые необходимо переопределить:

* void paint (Graphics&);
* void resized().

Метод paint(Graphics&) вызывается, когда область некоторого компонента графического интерфейса нуждается в перерисовке.

g.drawImage (background\_image, 0, 0, getWidth(), getHeight(), 200, 200, getWidth(), getHeight());

В данном методе вызывается функция drawImage(), прорисовывающая фоновое изображение программного модуля. В качестве параметров в неё передаётся ссылка на объект background\_image типа Image, в котором хранится информация об изображении, координаты левого верхнего уровня прямоугольника, в котором будет отображаться изображение, ширина и высота этого прямоугольника, координаты прямоугольника исходного изображения, его ширина и высота.

void ShimmerAudioProcessorEditor::resized()

{

int text\_height = 30;

int knob\_size = 80;

int gain\_area\_width = 100;

int octaver\_height = 175;

int delay\_height = 200;

int reverb\_height = 175;

Rectangle<int> window\_rectangle = getLocalBounds();

Rectangle<int> gain\_area =

window\_rectangle.removeFromLeft(gain\_area\_width);

Rectangle<int> octaver\_rectangle =

window\_rectangle.removeFromTop(octaver\_height);

Rectangle<int> delay\_rectangle =

window\_rectangle.removeFromTop(delay\_height);

Rectangle<int> reverb\_rectangle =

window\_rectangle.removeFromTop(reverb\_height);

octaver\_gui\_components.resized(octaver\_rectangle);

delay\_gui\_components.resized(delay\_rectangle);

reverb\_gui\_components.resized(reverb\_rectangle);

InputGainLabel->setBounds

(gain\_area.removeFromTop(text\_height));

InputGainKnob->setBounds

(gain\_area.removeFromTop(knob\_size));

OutputGainKnob->setBounds

(gain\_area.removeFromBottom(knob\_size));

OutputGainLabel->setBounds

(gain\_area.removeFromBottom(text\_height));

}

В методе resized() задаются используемые значения размеров компонентов графического интерфейса, вызываются методы resized() объектов компонентов графического интерфейса каждого эффекта. Также в данном методе задаются границы меток и ручек входного и выходного усилителей сигнала.

В фреймворке Juce обработка событий компонентов графического интерфейса реализована системой broadcaster-listener, которая соответствует поведенческому шаблону проектирования наблюдатель (на английском Observer). Данный шаблон создает механизм у класса, который позволяет получать экземпляру объекта этого класса оповещения от других объектов об изменении их состояния, тем самым наблюдая за ними. В конкретном случае компоненты графического интерфейса добавляют слушателей или наблюдателей с помощью функции AddListener(). Когда происходит некоторое событие, связанное с некоторым компонентом графического интерфейса, объект оповещает об этом своего слушателя. Слушатель, в свою очередь, ответственен за обработку события.

Данный класс также унаследован от абстрактных классов SliderListener и ButtonListener. Соответственно перегружены их методы sliderValueChanged (Slider\* sliderThatWasMoved) и buttonClicked (Button\* buttonThatWasClicked).

Функция sliderValueChanged (Slider\* sliderThatWasMoved) вызывается, когда пользователь изменяет положение ручки, а значит, и значение. Функция обработчик вызывается лишь в том случае, если до изменения пользователем положения ручки у соответствующего графического компонента была вызвана функция addListener(SliderListener \*), в которую был передан указатель на объект класса, унаследованного от SliderListener.

Функция buttonClicked (Button\* buttonThatWasClicked) вызывается когда пользователь нажимает на кнопку. Аналогично функция обработчик вызывается лишь в том случае, если до нажатия кнопки у соответствующего графического компонента была вызвана функция addListener(ButtonListener\*), в которую был передан указатель на объект класса, унаследованного от ButtonListener.

Класс ShimmerAudioProcessorEditor также унаследован от абстрактного класса Timer. Соответственно переопределена его виртуальная функция timerCallback(). Внутри вызывается функция updateTimecodeDisplay(), предназначенная для отображения на графическом интерфейсе обновлённой информации, полученной от host-программы.

## Блок настройки параметров обработки



Рисунок 3.5 – Организация блока настройки параметров обработки

Блок настройки параметров обработки реализован в классе ShimmerAudioProcessor (см. рисунок 3.5). Данный блок предназначен для преобразования ввода пользователя и хранения параметров, влияющих на работу блоков эффектов.

Пользовательские параметры хранятся в массиве UserParams[NumParams] типа float. Для понимания того, за что отвечает параметр была введена структура Parameters.

enum Parameters{

DelayBypass = 0,

DelayTime,

DelayFeedback,

DelayMix,

DelaySynch,

DelayDot,

DelaySecondDot,

ReverbBypass,

ReverbMix,

ReverbDecay,

InputGain,

OutputGain,

NumParams

};

NumParams – размер массива параметров обработки. В структуре Parameters это значение следует за названием номера последнего элементов, а значит равно количеству параметров. В массиве хранятся как положения ручек, которые представляют собой значения типа float, так и информация о состоянии кнопок, которая представляет собой значения типа bool. Принято решение, что если кнопка нажата, то значение true типа bool сохраняется в массиве параметров как значение 1.0f типа float. Значение false типа bool сохраняется в массиве как значение 0.0f типа float. Ещё одним параметром является поле hostBPM типа double, представляющее собой темп композиции. Темп задаётся host-программой.

Массив параметров UserParams[NumParams] инициализируется значениями в конструкторе ShimmerAudioProcessor() после инициализации объектов классов, представляющих работу звуковых эффектов.

UserParams[DelayBypass] = delay.getByPass();

UserParams[DelayTime] = delay.getDelayTimeMS();

UserParams[DelayFeedback] = delay.getFeedback();

UserParams[DelayMix] = delay.getMix();

UserParams[DelaySynch] = 0.0f;

UserParams[DelayDot] = 0.0f;

UserParams[DelaySecondDot] = 0.0f;

UserParams[ReverbBypass] = reverb.getBypass();

UserParams[ReverbMix] = reverb.getMix();

UserParams[ReverbDecay] = reverb.getDecayFactor();

UserParams[InputGain] = 1.0f;

UserParams[OutputGain] = 1.0f;

//default host BPM to 120, default in most DAWs

hostBPM = 120;

Методы класса ShimmerAudioProcessor, которые работают как интерфейс между блоком интерфейса пользователя и блоками эффектов указаны ниже.

float getParameter (int index);

void setParameter (int index, float newValue);

float calculateDelayTap(float tap);

Функция getParameter() в качестве параметра принимает значение типа int, которое определяет индекс параметра, который будет возвращён вызывающей функции.

Функция setParameter() устанавливает новое значение элемента с номером index массива параметров UserParams[], выполняет обработку и преобразование входных данных во внутренние параметры программного модуля. Данная функция вызывается при обработке пользовательского ввода. Логика обработки входных данных зависит от устанавливаемого параметра. Например, пользователь устанавливает ручку Delay в новое значение. Вызывается обработчик события изменения значения ручки, в котором, в свою очередь, вызывается метод setParameter(). В ячейку памяти UserParams[DelayTime] заносится новое значение newValue и вызывается функция SetDelay(float d\_ms) объекта класса Delay, который представляет собой модуль эффекта Delay. Данная функция устанавливает значение задержки равным передаваемому параметру. Типичное взаимодействие пользователя и объектов программного модуля обработки звуковой дорожки с течением времени представлено на диаграмме последовательности (см. чертёж ГУИР.400201.161 РР.2).

Класс ShimmerAudioProcessor содержит в качестве полей объекты классов Delay и SReverb, которые соответственно описывают работу блоков эффектов Delay и Reverb.

## Класс DelayLine

Перед рассмотрением классов Delay и Reverb, необходимо описать класс DelayLine, так как он является их неотъемлемой частью: цифровые эффекты Delay и Reverb основаны на простой DelayLine (см. рисунок 3.6). Класс DelayLine представляет собой обёртку буфера значений типа float.

Поле buffer типа float\* – указатель на область памяти, где будут храниться задержанный сигнал. Как известно, оцифрованный звук представляет собой массив сэмплов. Для создания буфера задержки сигнала определённой длительности в миллисекундах, необходимо знать частоту цифрового сигнала. Поле sample\_rate типа int хранит значение частоты дискретизации сигнала. Типичными значениями данного поля являются числа 44100, 48000, 96000, 192000.



Рисунок 3.6 – Класс DelayLine и его непосредственные связи с другими классами

Поля MAX\_DELAY\_SAMPLES и MAX\_DELAY\_MS задают максимальную длину буфера задержки в сэмплах и миллисекундах соответственно. Поля delay\_ms и delay\_samples задают текущую длину буфера задержки в миллисекундах и сэмплах оответственно. Позиция, с которой будет производиться следующая запись в буфер, задаётся полем writePos, а позиция, с которой будет производиться следующее чтение, задаётся полем readPos.

class DelayLine

{

public:

//constructor / destructor

DelayLine(float d\_ms\_max = 0.0f, int sr = 44100,

float d\_ms = 0.0f);

~DelayLine();

//getters

float getDelayTimeMS();

float getMaxDelayTimeMS();

//setters

void setDelayTimeMS(float time\_ms);

void setDelay(float d\_ms);

//business functions

float readDelay();

void writeDelay(float in);

float next(const float in);

void resetDelayLine();

void clearBuffer();

private:

int sample\_rate;

int writePos, readPos, MAX\_DELAY\_SAMPLES;

float delay\_ms, delay\_samples, fraction,

MAX\_DELAY\_MS;

float \*buffer;

};

## Блок эффекта «delay»

Блок эффекта «delay» представлен классом Delay. Данный класс представляет собой обёртку для класса DelayLine.

class Delay

{

public:

Delay(float d\_ms\_max = 0.0f, int sr = 44100, float

d\_ms = 800.0f, float feedback = 0.0f, float

mixLevel = 0.3f, bool bp = false);

~Delay();

float next(float in);

float getDelayTimeMS();

float getFeedback();

float getMix();

bool getByPass();

void setFeedback(float f\_pct);

void setMix(float m\_pct);

void setByPass(bool bp);

void setDelay(float d);

private:

DelayLine delay\_line;

float feedback, mixLevel;

bool delay\_bypass;

}

Класс Delay кроме поля delay\_line класса DelayLine содержит параметры, которые задаются и изменяются пользователем. Поле bypass типа bool определяет, в каком состоянии находится блок эффекта: пассивном или активном. Поле Feedback типа float отвечает за обратную связь эффекта Delay, то есть он определяет коэффициент ослабления каждой последующей копии сигнала. Поле mixLevel типа float отвечает за соотношение между исходным или «сухим» сигналом и обработанным эффектом Delay или «влажным» сигналом.

## Блок эффекта «reverb»

Блок эффекта «reverb» задан классом SReverb. Его структура отражена на рисунке 3.7. Цифровой эффект Reverb содержит следующие фильтры:

* + - всепропускающий фильтр;
    - гребенчатый фильтр;
    - фильтр нижних частот.

Соответственно эти фильтры реализованы в классах:

* + - AllPass;
    - Comb;
    - LowPass.

Класс SReverb кроме полей вышеуказанных классов содержит параметры, которые задаются и управляются пользователем. Поле decayFactor типа float определяет коэффициент затухания звуковых отражений. Поле mix типа float определяет соотношение необработанного сигнала к обработанному. Поле bypass типа bool определяет активен ли в данный момент времени эффект Reverb. Активным называется состояние эффекта, при котором он обрабатывает входной сигнал и подаёт его на выход. При значении bypass равном true, модуль эффекта Reverb не вносит изменений в сигнал.



Рисунок 3.7 – Класс SReverb и его компоненты

Класс SReverb имеет единственную бизнес-функцию, которая принимает очередной сэмпл цифрового сигнала и выдаёт сэмпл обработанного сигнала.

class SReverb

{

public:

//constructor / destructor

SReverb(const int sr = 44100, const float rt60

= 3.0f, const float cDelay1 = 29.7, const float cDelay2 = 37.1, const float cDelay3 = 41.1, const float cDelay4 = 43.7, const float aDelay1 = 5.0, const float aDelay2 = 1.7, const float aGain1 = 0.707, const float aGain2 = 0.5, const float lCutoff1 = 2300.0f, const float mx = 0.3f); //const float lCutoff1 = 2300.0f

~SReverb();

//getters

float getDecayFactor();

float getCombDelay(const int id);

float getAllpassDelay(const int id);

float getAllpassGain(const int id);

float getLowpassCutoff(const int id);

bool getBypass();

float getMix();

//setters

void setDecayFactor(const float df);

void setCombDelay(const int id, const float sr,

const float d\_ms);

void setAllpassGain(const int id, const float

g);

void setAllpassDelay(const int id, const int

sr, const float d\_ms);

void setLowpassCutoff(const int id, const int

sr, const float cf\_hz);

void setBypass(bool bp);

void setMix(float value);

//business methods

float next(const float in);

private:

float decayFactor, ALLPASS\_GAIN\_LIMIT;/

float mix;

bool bypass;

Comb \*combs[NUM\_COMBS];

Allpass \*allpasses[NUM\_ALLPASSES];

Lowpass \*lowpasses[NUM\_LOWPASSES];

};

В конструкторе класса SReverb, инициализируются многочисленные буферы DelayLine, содержащиеся в фильтрах Comb и AllPass. Параметры гребенчатых и всепропускающих фильтров установлены экспериментальным путём, отталкиваясь от значений указанных на рисунке 3.8. Переменная rt60 является стандартным измерением времени реверберации, и определяется как время, необходимое для уменьшения звукового давления на 60 dB, с момента внезапного окончания генерируемого тестового сигнала.

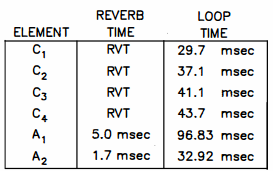


Рисунок 3.8 – Значения параметров реверберации для симуляции концертного зала среднего размера

## Всепропускающий фильтр

Всепропускающий фильтр представлен классом AllPass.

class Allpass{

public:

//constructor

Allpass(const int sr, const float d\_ms, const

float d\_ms\_max, const float g);

~Allpass();

//getters

float getGain();

float getDelayTimeMS();

//setters

void setGain(const float g);

void setDelayTimeMS(float time\_ms);

//business methods

float next(const float in);

private:

float gain;

DelayLine \*delay;

};

Работа данного фильтра основывается на DelayLine. Данный фильтр имеет единственную бизнес-функцию next(float in), которая принимает на свой вход очередной сэмпл сигнала и возвращает на выход сэмпл типа float обработанного сигнала. Кроме приватного поля delay\*, класс содержит поле gain типа float. Поле gain отвечает за усиления сигнала внутри цепочки фильтра (см. рисунок 3.9). Данный параметр рассчитывается в классе Reverb на основе времени реверберации и времени циклической задержки.

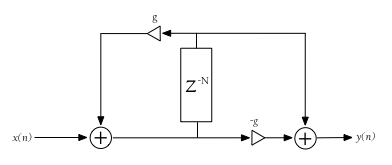


Рисунок 3.9 – Структура всепропускающего фильтра

## Гребенчатый фильтр

class Comb{

public:

//constructor / destructor

Comb(const int sr, const float d\_ms, const float d\_ms\_max, const float g);

~Comb();

//getters

float getGain();

float getDelayTimeMS();

//setters

void setGain(const float g);

void setDelayTimeMS(float time\_ms);

//business methods

float next(const float in);

private:

float gain;

DelayLine \*delay;

Lowpass \*lpFilter;

};

Гребенчатый фильтр имеет такой же интерфейс, как и всепропускающий фильтр: конструктор с таким же составом параметров, функции для получения и установки значений private полей, бизнесс-функцию next(const float in). Однако внутреннее устройство фильтров отличается и соответствует структуре, указанной на рисунке 3.10.

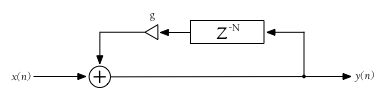


Рисунок 3.10 – Структура гребенчатого фильтра

## Фильтр низких частот

Фильтр нижних частот реализован довольно простым классом Lowpass. В качестве параметров конструктор этого класса принимает частоту дискретизации и частоту среза в герцах. Как и вышеописанные модули имеет одну бизнес-функцию next(float in),которая возвращает сэмпл отфильтрованного сигнала.

class Lowpass{

public:

//constructor

Lowpass(const int sr, const float cf\_hz);

//getters

float getCutoff();

//setters

void setCutoff(const int sr, const float cf\_hz);

//business methods

float next(const float in);

private:

float cutoff, coef, prev;

};

## Блоки преобразования входного и выходного сигналов

Блоки преобразования входного и выходного сигналов представляют собой ручки усиления уровня соответствующих сигналов. Данные блоки довольно просты и реализуется в рамках функции processBlock() класса ShimmerAudioProcessor.

# РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

Обработка звука изначально закреплена за классом ShimmerAudioProcessor. Данный класс имеет две функции, в которых происходит непосредственная обработка аудиоинформации:

* processBlock ();
* processBlockBypassed ().

В одно и то же время функционирует лишь один из указанных выше методов. Функция processBlockBypassed () вызывается тогда, когда весь плагин находится в режиме Bypass, а функция processBlock (), когда плагин активен.

void ShimmerAudioProcessor::processBlock

(AudioSampleBuffer& buffer, MidiBuffer& midiMessages)

{

if(result.bpm != 0)

this->hostBPM = result.bpm;

int numSamples = buffer.getNumSamples(); //THIS IS NUM SAMPLES PER CHANNEL

float\* channelDataLeft = buffer.getWritePointer(0);

float\* channelDataRight =

buffer.getWritePointer(1);

const float\* inChannelDataLeft =

buffer.getReadPointer(0);

const float\* inChannelDataRight =

buffer.getReadPointer(1);

float out\_l = 0.0f, out\_r = 0.0f;

for(int i = 0; i < numSamples; i++)

{

out\_l = delay.next(channelDataLeft[i]);

out\_l = reverb.next(out\_l);

channelDataLeft[i] = out\_l \*

UserParams[OutputGain];

channelDataRight[i] = channelDataLeft[i];

}

for (int i = getNumInputChannels(); i < getNumOutputChannels(); ++i)

{

buffer.clear (i, 0, buffer.getNumSamples());

}

//get the host BPM and sync playhead to it

updateCurrentTimeInfoFromHost();

}

Разработка программных модулей – аналог разработки принципиальной схемы аппаратного дипломного проекта. В этом разделе подробно описываются уже внутренние алгоритмы ключевых процедур и функций с разбиением на отдельные подразделы. Здесь же описывается реализация наиболее интересных алгоритмов, например, алгоритмов шифрования. Данный раздел может сопровождать чертежи схем программ и

содержать ссылки на них.

# ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Тестирование является одним из важнейших этапов разработки программного обеспечения. Исключением не являются и программные модули обработки звука. Тестирование позволяет выявить проблемы и ситуации, в которых поведение программного модуля является неправильным ещё на стадии разработки. Таким образом, проводя тестирование, разработчики убеждаются в исправности написанного кода, а также в соответствии поведения программного продукта требованиям заказчика.

Также тестирование позволяет продемонстрировать заказчикам, что программный продукт завершён и соответствует требованиям, а значит, готов к реализации на рынке.

Тестирование программного модуля обработки звуковой дорожки в реальном времени проводилось в операционных системах Windows и Mac в следующих цифровых звуковых рабочих станциях:

* Cubase 7 Elements;
* Pro Tools;
* Apple Logic Pro X;
* Ableton Live 7.

При тестировании программного модуля обработки звуковой дорожки в реальном времени использовался метод чёрного ящика. Данный метод предполагает тестирование поведения объекта с точки зрения внешнего мира, то есть подразумевается отсутствие знаний о внутреннем устройстве программного модуля.

При разработке программного модуля обработки звуковой дорожки в реальном времени был использованы приёмы тестирования, называемые «эквивалентное разбиение», «анализ граничных значений» и «предположение об ошибке».

Метод «предположение об ошибке» в значительной степени основан на интуиции опытного разработчика. Идея заключается в том, что составляется список, включающий все ошибки и исключительные ситуации, которые могут возникнуть. Далее на основе этого списка создаются тесты.

Граничные условия – это ситуации, возникающие на высших и нижних границах входных классов эквивалентности. В файле исходных текстов DelayLine.cpp присутствует следующий код:

assert(time\_ms <= MAX\_DELAY\_MS);//check bound on delay time

Функция assert() подключается заголовочным файлом <cassert>. Она оценивает выражение, которое передается ей в качестве аргумента. Если аргумент-выражение этого макроса в функциональной форме равно нулю (т.е. выражение ложно), сообщение записывается на стандартное устройство вывода ошибок и вызывается функция abort(), работа программы прекращается. При отладке программы, мы можем неограниченное количество раз вызывать assert(). После отладки программы, не будет надобности в вызванных нами функциях assert(). Чтобы отключить все функции assert() в языке С++ определён макрос NDEBUG. То есть, необходимо включить строку #define NDEBUG в код программы.

В случае, указанном выше, оценивается верхнее граничное значение размера буфера задержки в методе setDelayTimeMS() и конструкторе класса DelayLine. В ходе разработки модуля эффекта Delay программного модуля обработки звуковой дорожки в реальном времени данное выражение позволило обнаружить ошибку, которая возникла при добавлении кода, обрабатывающего нажатие кнопок Dotted и Second Dotted, которые увеличивают время между повторами в 1,5 и в 1,25 раз соответственно. Отсутствие данной проверки привело бы к обращению к области памяти, не выделенной программой. Это бы могло привести к непредсказуемым последствиям.

Use Case (от английского – вариант использования) — это сценарная техника описания взаимодействия. С помощью Use Case может быть описано и пользовательское требование, и требование к взаимодействию систем, и описание взаимодействия людей и компаний в реальной жизни.

В разработке ПО эту технику часто применяют для проектирования и описания взаимодействия пользователя и системы, поэтому название Use Case часто воспринимает как синоним требования пользователя к решению определенной задачи в системе.

Тестовый случай (Test Case) – это артефакт, описывающий совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или её части. Тестовый случай имеет следующую структуру:

* действие;
* ожидаемый результат;
* результат теста.

Каждый тест кейс должен иметь 3 части:

* + - cписок действий, которые приводят систему к состоянию, пригодному для проведения основной проверки, или условий, выполнение которых говорит о том, что система находится в пригодном для проведения основного теста состояния;
    - cписок действий, переводящих систему из одного состояния в другое, для получения результата, на основании которого можно сделать вывод о удовлетворении реализации, поставленным требованиям;
    - cписок действий, переводящих систему в первоначальное состояние (состояние до проведения теста - initial state).

Таблица 5.1 – Основные тесты программного модуля

|  |  |  |
| --- | --- | --- |
| Требования | Тест | Тест пройден |
| 1 | 2 | 3 |
| Настраиваемое время задержки должно соответство-вать отобра-жаемому на экране. | 1. Убедиться, что компонент находится в активном состоянии в режиме, не синхронизированным с темпом host-программы (отсутствуют кнопки Dotted и Second Dotted). 2. Установить значение ручки Delay равным 400, ручки Feedback – 1, ручки Mix – 0,5. 3. Воспроизвести звук щелчка. 4. Измерить время от оригинального звука до 10 повтора. 5. Разделить время на количество повторов и убедиться, что получившееся значение равно установленному ручкой. | да |
| При нажатии на кнопку Synchronize интерфейс должен измениться. | 1. Убедиться, что компонент находится в активном состоянии в режиме, не синхронизированным с темпом host-программы (отсутствуют кнопки Dotted и Second Dotted). 2. Нажать на кнопку Synchronize; 3. Убедиться, что на экране появились дополнительные кнопки Dotted и Second Dotted,название ручки Delay изменилось на Denominator. 4. Убедиться, что диапазон задаваемых значений этой ручкой изменился с 0..2000 на 0..5. | да |
| Кнопки Bypass должны отключать соответствующие им модули обработки. | 1. Убедиться, что компоненты находятся в активном режиме, ручки Mix, Delay и Decay установлены в ненулевые значения. 2. Воспроизвести звук щелчка. 3. Нажать на кнопку Bypass компонента Delay. 4. Убедиться, что после нажатия кнопки Bypass компонент необрабатывает звук. 5. Снова нажать на кнопку Bypass компонента. 6. Повторить все вышеперечисленные действия для компонента Reverb. | да |

*Продолжение таблицы 5.1*

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| Ручки Mix должны изменять соотношение между необработанным сигналом и обработанным. | 1. Установить значения всех ручек Mix в ноль. 2. Воспроизвести звук щелчка. 3. Подождать 5 секунд и убедиться, что уровень выходного сигнала равен нулю. 4. Установить значения всех ручек Mix в значения, отличные от нуля. 5. Убедиться, что звук обрабатывается модулями эффектов. | да |
| Кнопка Dotted должна добавлять 1/2 от времени между повторами, а кнопка Second Dotted должна добавлять 1/4 от времени между повторами. | 1. Убедиться, что компонент находится в активном состоянии в режиме, синхронизированным с темпом host-программы (присутствуют кнопки Dotted и Second Dotted). 2. Нажать на кнопку Dotted. 3. Установить значение ручки Delay равным 4, а ручки Feedback – 100. 4. Воспроизвести звук щелчка. 5. Засечь время от оригинального звука до 10 повтора. 6. Разделить время на количество повторов. 7. Убедиться, что полученное значение равно 2/3 от темпа host-программы. 8. Нажать на кнопку Dotted, а затем на кнопку Second Dotted. 9. Воспроизвести звук щелчка. 10. Засечь время от оригинального звука до 20 повтора. 11. Разделить время на количество повторов. 12. Убедиться, что полученное значение равно 4/5 от темпа host-программы. | да |

В таблице 5.1 представлены некоторые тесты, основанные на пользовательских требованиях, которые были пройдены программным модулем обработки звуковой дорожки в реальном времени. Для начала тестирования программного модуля обработки звуковой дорожки в реальном времени необходимо в цифровой рабочей звуковой станции создать проект с одной моно-дорожкой, на которой будет записан кратковременный звук щелчка. Во вкладке Inserts дорожки нажать на пустой слот, выбрать плагин Shimmer и убедиться, что отобразилось его окно. Если окно не появилось, необходимо щёлкнуть мышкой по появившемуся названию Shimmer во вкладке Inserts.

В таблице 5.1 указана лишь доля реально проводимых тестов. Более детально проверялось функционирование ручек Input Gain и Output Gain, Mix, Decay, а также работа неявных для пользователя фильтров. Для тестирования их работы был использован программный модуль компании Spectrum Analysis Voxengo SPAN PLUS. Данный модуль имеет спектральный анализатор, измеритель пикового и действующих значений уровня сигнала, детектор превышения сигналом уровня 0 dBFs.

Таким образом, проанализировав результаты испытаний и тестирования, можно сделать вывод, что модуль обработки звуковой дорожки в реальном времени является надёжным и работоспособным. Программный модуль точно выполняет свою функцию в соответствии с параметрами, задаваемыми пользователем.

# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## Системные требования

Требования к аппаратному обеспечению в большей степени диктуются используемой DAW, нежели разработанным программным модулем. Требования к программному обеспечению зависят от используемой DAW. В качестве примера в таблице 6.1 приведены системные требования новой цифровой звуковой рабочей станции фирмы Steinberg Cubase Pro 9.

Формат программного модуля – VST. Данный формат широко распространён и поддерживается всеми известными DAW: от популярных и функциональных Cubase и Pro Tools, до малоизвестной и легковесной TunaFish.

Таблица 6.1 – Системные требования Cubase Pro 9

|  |  |
| --- | --- |
| Mac OS X | Windows |
| OS X 10.11 / macOS Sierra | 64-Bit Windows 7 / 8.x / 10 |
| Многоядерный процессор 64-bit Intel или AMD (рекомендован Intel i5 или более быстрый) | |
| 4 GB ОЗУ (рекомендовано 8 GB или больше) | |
| 18 GB свободного места на жёстком диске | |
| разрешение экрана 1366 x 768 (рекомендовано 1920 x 1080) | |
| Графическая карта с DirectX 10 и поддержкой WDDM 1.1 (для ОС Windows) | |
| USB порт для USB-eLicenser (управление лицензиями) | |
| Аудиоинтерфейс, поддерживаемый ОС (для высокой производительности, низких задержек в звуке рекомендуется поддержка ASIO) | |
| Соединение с интернетом (для активации, регистрации аккаунта и продукта, дополнительных загрузок в ходе инсталляции) | |

Файл разработанного модуля обработки звука Shimmer.dll имеет размер всего 5 MiB.

## Процесс инсталляции

Процесс инсталляции программного модуля обработки звуковой дорожки в реальном времени указан для DAW Cubase 7 Elements на операционной системе Windows 7 Service Pack 1. Для других операционных систем и цифровых звуковых рабочих станций процесс инсталляции может отличаться, но основные этапы будут такими же.

Алгоритм добавления и использования разработанного плагина в Cubase следующий:



Рисунок 6.1 – Создание проекта в Cubase 7 Elements

1. Скопировать с диска файл Shimmer.dll в папку с VST плагинами на вашем компьютере. В данном случае была выбрана папка D:\vst\_plugins\_folder.
2. Открыть цифровую звуковую рабочую станцию (2 раза кликнуть на исполняемый файл Cubase LE AI Elements 7.exe) и создать в ней проект. Для этого в меню More (см. рисунок 6.1) выбрать пункт Empty, указать папку, в которой будет создан проект, нажать Create.



Рисунок 6.2 – Открытие меню Plug-in Information

1. Открыть меню с информацией о плагинах. В строке меню выбрать пункт Devices, в выпадающем списке выбрать Plug-in Information (см. рисунок 6.2).
2. Указать путь к папке, в которой находится файл Shimmer.dll. Во вкладке VST Plug-ins нажать кнопку VST 2.x Plug-in Path. В появившемся окне нажать кнопку Add (см. рисунок 6.3). В появившемся окне выбрать местоположение папки и нажать OK. Снова нажать OK. Нажать кнопку Update. Как видно на рисунке 6.4, в списке доступных плагинов появился файл с именем Shimmer.dll.

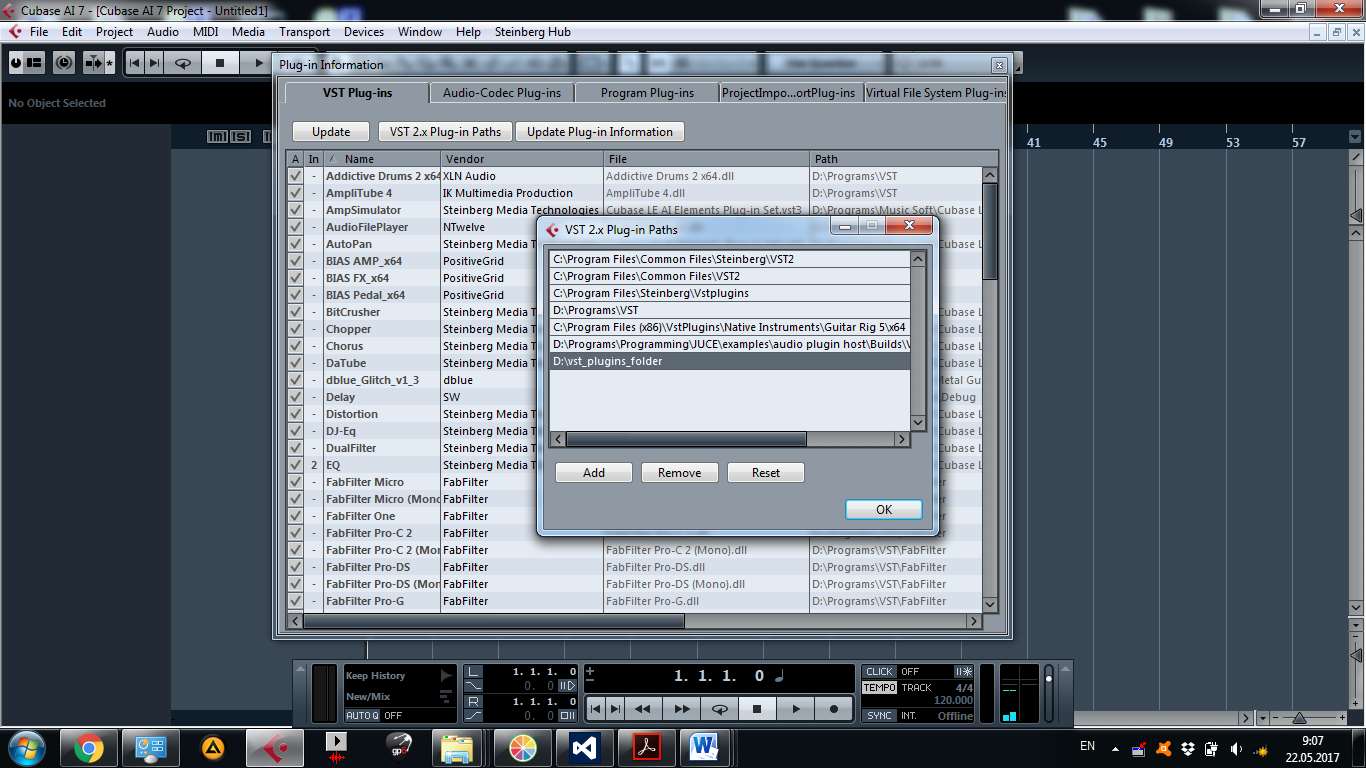


Рисунок 6.3 – Окно VST 2.x Plug-in Paths

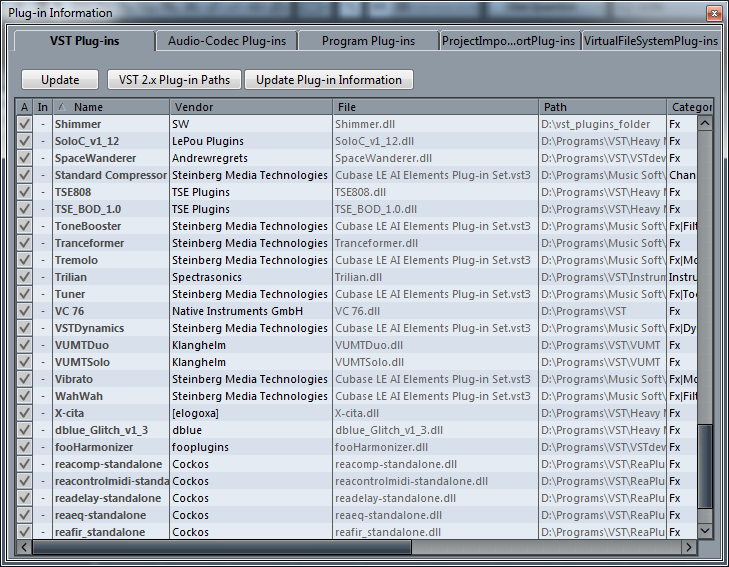


Рисунок 6.4 – Окно Plug-in Information

1. Чтобы использовать плагин на аудиодорожке нужно её создать. Нужно нажать правой кнопкой по области дорожек и выбрать Add Audio Track … (см. рисунок 6.5). В появившемся окне выбрать конфигурацию дорожки Mono, указать любое имя и нажать Add Track (см. рисунок 6.6).

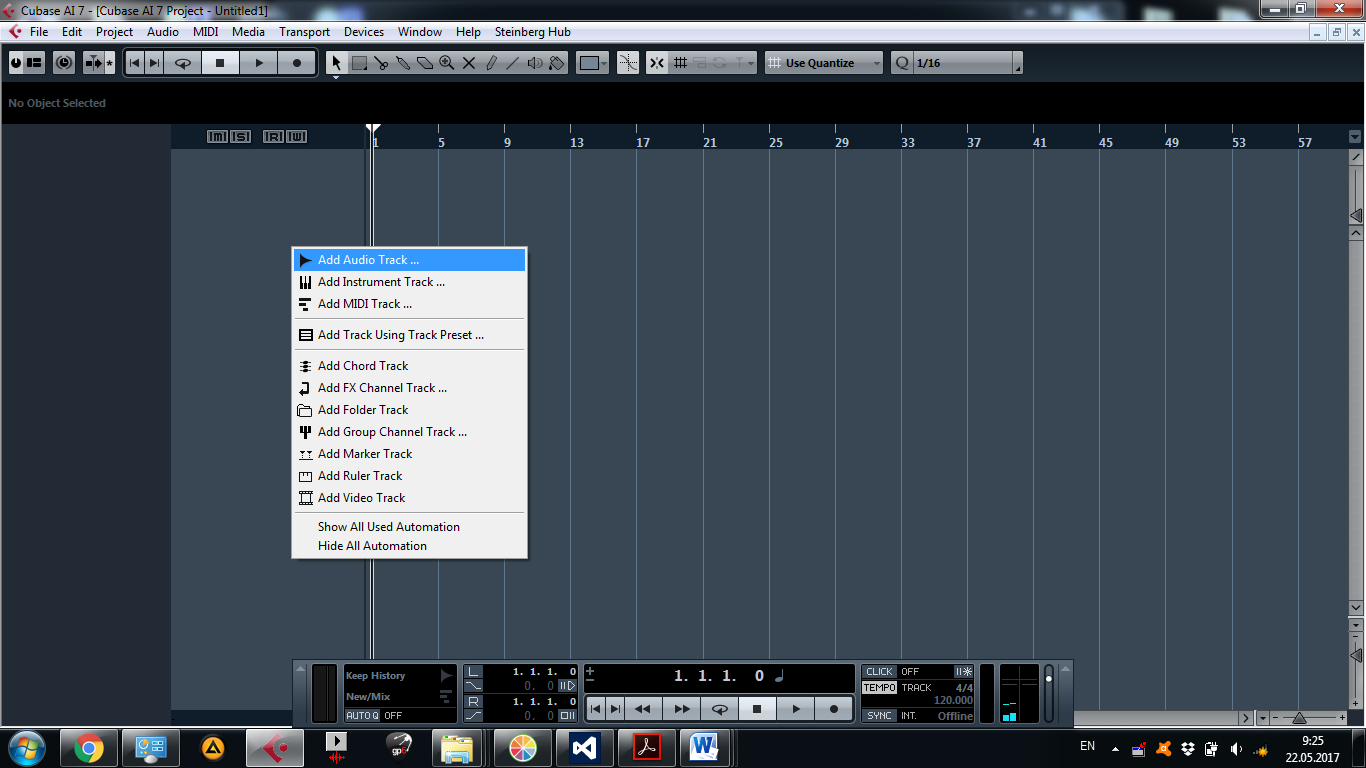


Рисунок 6.5 – Добавление звуковой дорожки 1

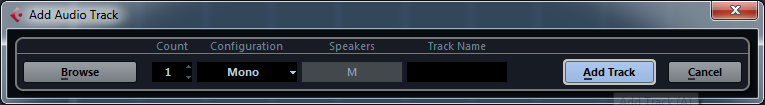


Рисунок 6.6 – Добавление звуковой дорожки 2



Рисунок 6.7 – Применение плагина к звуковой дорожке

1. Применить плагин к звуковой дорожке. Для этого необходимо выбрать дорожку, в меню Inserts нажать на пустой слот, выбрать необходимый плагин (см. рисунок 6.7). Теперь звуковая дорожка обрабатывается плагином в реальном времени.

## Пользовательский интерфейс

Графический пользовательский интерфейс представлен на рисунке 6.8

Программный модуль, как и описывалось ранее, состоит из трёх компонентов: Delay, Octaver, Reverb. Интерфейс представляет собой три секции, соответствующие этим трём компонентам.



Рисунок 6.8 – Графический интерфейс программного модуля обработки звуковой дорожки в реальном времени.

* + 1. Элементы управления эффекта Octaver

Компонент Octaver имеет следующие элементы графического интерфейса:

* кнопка Bypass;
* ручка Mix;
* ручка Octave low;
* ручка Octave high;

При нажатой кнопке Bypass, исходный звуковой сигнал не будет претерпевать обработку эффектом Delay и поступит на вход следующего компонента без изменений.

Ручка Mix регулирует соотношение обработанного сигнала эффектом Octaver к необработанному.

Ручка Octave low регулирует громкость добавляемого сигнала, который на октаву ниже основного.

Ручка Octave high регулирует громкость добавляемого сигнала, который на октаву выше основного.

* + 1. Элементы управления эффекта Delay

Эффект Delay управляется следующими элементами графического интерфейса:

* кнопка Bypass;
* ручка Mix;
* ручка Delay;
* ручка Feedback
* кнопка Synchronize
* кнопка Dotted;
* кнопка Second dotted.

При нажатой кнопке Bypass, звуковой сигнал, поступающий от блока Octaver, не будет претерпевать обработку эффектом Delay и поступит на вход следующего компонента Reverb без изменений.

Ручка Mix определяет количество эффекта в процентах, то есть громкость задержанного сигнала. Пределы изменения: от нуля до ста.

Ручкой Delay задаётся время, спустя которое сигнал будет повторяться. Пределы изменения этого параметра: от нуля до двух тысяч миллисекунд.

Ручка Feedback отвечает за обратную связь эффекта Delay. При минимальном значении равном нулю, компонент будет добавлять одну копию исходного сигнала. При максимальном – бесконечное количество копий. Если значение этого параметра находится в пределах от минимального, до максимального, громкость периодических копий сигнала будет постепенно уменьшаться.

Кнопка Synched меняет режим работы эффекта Delay на синхронный и обратно. В синхронном режиме программный модуль обработки звуковой дорожки в реальном времени получает темп от хост-программы, то есть от цифровой звуковой рабочей станции, и синхронизирует копии сигнала с полученным темпом. В этом режиме ручка Delay изменяет своё предназначение: с её помощью задаётся делитель темпа.

Кнопка Dotted добавляет одну вторую от времени между долями к синхронизированным с темпом повторам.

Кнопка Second dotted добавляет одну четверть от времени между долями к синхронизированным с темпом повторам.

* + 1. Элементы управления эффекта Reverb

Эффект Reverb управляется следующими элементами графического интерфейса:

* кнопка Bypass;
* ручка Decay;
* ручка Mix.

При нажатой кнопке Bypass, звуковой сигнал не будет претерпевать обработку эффектом Reverb и поступит на выход.

Ручка Mix определяет соотношение необработанного сигнала к обработанному эффектом сигналу. Этот параметр измеряется в процентах, то есть пределы изменения: от нуля до ста.

Ручкой Decay задаётся время отражений.

# ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ И РЕАЛИЗАЦИИ ПРОГРАММНОГО ПРОДУКТА

## Характеристика программного продукта

Целью дипломного проекта является создание программного модуля обработки звуковой дорожки в реальном времени. Программный продукт может быть использован на обычных настольных компьютерах и ноутбуках во всех популярных цифровых звуковых рабочих станциях. Разработка ПО осуществляется IT-компанией.

Потенциальные пользователи программного модуля обработки звуковой дорожки в реальном времени:

* + - музыканты, имеющие возможность подключить свой инструмент к компьютеру для получения цифрового представления звука;
    - диджеи, осуществляющие воспроизведение записанных на звуковые носители музыкальных произведений с изменением материала техническими средствами в реальном времени;
    - звукорежиссёры, обрабатывающие записанные звуки или музыкальные произведения.

В связи с распространением цифровой обработки звука с помощью персонального компьютера на рынке есть достаточный спрос на данный программный продукт. Программный модуль обработки звуковой дорожки в реальном времени разрабатывается для свободной реализации на рынке информационных технологий копиями.

Исходя из тенденций на рынке модулей обработки звука, а также дополнительного маркетингового исследования, приложение будет востребовано на рынке в течение четырех лет: в 2017 году планируется реализовать 60 копий, в 2018 году планируется реализовать 70 копий, в 2019 планируется реализовать 70 копий, в 2020 планируется реализовать 50 копий. Итого 250 копий.

Экономическая целесообразность инвестиций в разработку и реализацию представленного программного продукта определяется на основе расчета таких показателей, как:

* чистая дисконтированная стоимость (ЧДД);
* срок окупаемости инвестиций (Ток);
* рентабельность инвестиций (Ри).

## Расчет сметы затрат на разработку и отпускной цены программного продукта

Расчет величины основной заработной платы участников команды осуществляется по формуле 7.1:

|  |  |
| --- | --- |
|  | (7.1) |
|  |  |

где *n* – количество исполнителей занятых в разработке;

Т1ч*i* – часовая тарифная ставка i-го исполнителя, руб.;

Кт*i* – тарифный коэффициент разряда i-го исполнителя;

Тч – количество часов работы в день, ч;

Фэ*i* – эффективный фонд рабочего времени i-го исполнителя;

К – коэффициент премирования, принятый равным 1,3.

Тарифная ставка первого разряда (на 01.03.2017) на предприятии составляет 120 рублей. Среднемесячная расчетная норма рабочего времени на 2017 год составляет 22 дня. Коэффициент премирования равен 1,3.

В процессе разработки приложения принимала участие команда из двух исполнителей: инженера-программиста 10 разряда с тарифным коэффициентом 2,48, и руководителя проекта 12 разряда с тарифным коэффициентом 3,25. Руководитель проекта ставит задачи перед инженером-программистом, среди которых следующие работы: проектирование интерфейса, написание кода, отладка и тестирование программного продукта.

Результаты расчёта основной заработной платы команды разработчиков представлены в таблице 7.1.

Таблица 7.1 – Расчет затрат на основную заработную плату команды разработчиков

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Участник  команды | Раз-  ряд | Тариф-  ный коэффи-  циент | Месячная  тарифная  ставка, руб. | Дневная тарифная  ставка, руб. | Плано-вый фонд рабочего времени, дн. | Заработная  плата, руб. |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Руково-дитель проекта | 12 | 3,25 | 390 | 17,73 | 25 | 443,18 |
| Инженер-програм-  мист | 10 | 2,48 | 297,60 | 13,53 | 50 | 676,36 |
| Итого |  |  |  |  |  | 1119,55 |
| Премия (30%) |  |  |  |  |  | 335,86 |

*Продолжение таблицы 7.1*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Основная заработная плата (Зо) |  |  |  |  |  | 1455,41 |

Затраты на дополнительную заработную плату команды разработчиков включает выплаты, предусмотренные законодательством о труде (оплата отпусков, льготных часов, времени выполнения государственных обязанностей и других выплат, не связанных с основной деятельностью исполнителей) и определяется по формуле 7.2:

|  |  |
| --- | --- |
|  | (7.2) |

где – норматив дополнительной заработной платы, 10%.

Размер дополнительной заработной платы исполнителей составит:

Отчисления на социальные нужды (в фонд социальной защиты населения и на обязательное страхование) определяются в соответствии с действующими законодательными актами по формуле 7.3:

|  |  |
| --- | --- |
|  | (7.3) |

где – норматив отчислений на социальные нужды, 34,6%.

Размер отчислений в фонд социальной защиты населения и на обязательное страхование составляет:

Расходы по статье «Машинное время» включают оплату машинного времени, необходимого для разработки и отладки ПО и определяются по формуле 7.4:

|  |  |  |
| --- | --- | --- |
|  |  | (7.4) |

где Цм – цена одного машино-часа, руб.;

Тч – количество часов работы в день;

Тдн – время, затраченное на разработку проекта, дн.

Расходы по статье «Машинное время», на разработку программного продукта составляют:



Размер прочих затрат, включающий затраты на аренду помещения, освещение и отопление, вычисляется по формуле 7.5:

|  |  |
| --- | --- |
|  | (7.5) |

Норматив прочих затрат принят равным 10%. Тогда прочие затраты на разработку программного продукта составляют:

Затраты по статье «Накладные расходы» связаны с необходимостью содержания аппарата управления, вспомогательных хозяйств и опытных производств. Определяются по формуле:

|  |  |
| --- | --- |
|  | (7.6) |

где Нн – норматив накладных расходов, рассматриваемых в целом по организации (50%).

Общая сумма расходов по смете (Ср) определяется как сумма выше рассчитанных показателей:

|  |  |  |
| --- | --- | --- |
|  |  | (7.7) |

Подставив значения в формулу расчета расходов по смете 7.7, получим:

В дополнение к выше рассчитанным параметрам, определяются расходы на сопровождение и адаптацию программного продукта (Зса):

|  |  |  |
| --- | --- | --- |
|  |  | (7.8) |

где Нрса – это норматив расходов обусловленных сопровождением и адаптацией ПО (5%).

Используя формулу 7.8, определим значение расходов:

Общая сумма расходов на разработку приложения, как полная себестоимость программного продукта (Сп) определяется по формуле 7.9:

|  |  |  |
| --- | --- | --- |
|  |  | (7.9) |

Общая сумма расходов на разработку:

В таблице 7.2 указаны рассчитанные значения затрат на разработку программного продукта.

Таблица 7.2 – Смета затрат на разработку программного обеспечения

|  |  |  |
| --- | --- | --- |
| Статья затрат | Условное обозначение | Сумма, руб. |
| Основная заработная плата |  | 1455,41 |
| Дополнительная заработная плата |  | 145,54 |
| Отчисления на социальные нужды |  | 553,93 |
| Затраты по статье «Машинное время» |  | 720,00 |
| Прочие затраты |  | 145,54 |
| Затраты по статье «Накладные расходы» |  | 727,70 |
| Сумма расходов по смете |  | 3748,12 |
| Затраты на сопровождение и адаптацию |  | 187,41 |
| Полная себестоимость |  | 3935,53 |

Прогнозируемая прибыль программного продукта рассчитывается по формуле 7.10:

|  |  |  |
| --- | --- | --- |
|  |  | (7.10) |

где Пр – прибыль от реализации программного продукта;

Ур – планируемый уровень рентабельности (15%);

Сп – себестоимость программного продукта.

Получаем:

Прогнозируемая отпускная цена программного продукта будет определяться по формуле 7.11:

|  |  |  |
| --- | --- | --- |
|  |  | (7.11) |

Подставляя значения в формулу (7.10), получаем прогнозируемую отпускную цену программного продукта:

## Расчёт экономического эффекта от реализации ПО

Экономический эффект организации-разработчика программного обеспечения в данном случае заключается в получении прибыли от его продажи множеству потребителей. Прибыль от реализации в данном случае напрямую зависит от объемов продаж, цены реализации и затрат на разработку ПО.

На основании анализа цен на аналогичные программные продукты цена программного модуля обработки звуковой дорожки в реальном времени будет составлять 50 руб.

Запланированное количество проданных копий – 250. Прибыль от продажи одной копии рассчитывается по формуле 7.12:

|  |  |  |
| --- | --- | --- |
|  |  | (7.12) |

где Ц – цена реализации одной копии;

НДС – сумма налога на добавленную стоимость;

Зр – сумма расходов на разработку и реализацию;

N – количество реализованных копий.

В связи с тем, что компания-исполнитель является резидентом ПВТ и по текущему законодательству РБ такие предприятия освобождаются от уплаты налога на прибыль и на добавленную стоимость, в подсчёте прибыли от продажи одной копии программного продукта НДС участвовать не будет.

Итого, прибыль за продажу одной копии программного продукта:

Суммарная прибыль за год рассчитывается, как прибыль от продажи одной копии на количество копий:

|  |  |  |
| --- | --- | --- |
|  |  | (7.13) |

где Пед – прибыль от продажи одной копии, руб;

N – количество копий за год.

Прибыль от продажи программного продукта по годам, рассчитанная по формуле 7.13 составляет:

Так как компания-исполнитель по текущему законодательству освобождается от уплаты налога на прибыль, чистая прибыль по годам составит:

## Расчет показателей экономической эффективности инвестиций в разработку и реализацию программного продукта

В ходе реализации программного продукта чистая прибыль в конечном итоге возмещает расходы на разработку. Однако полученные при этом суммы результатов (прибыли) и затрат по годам следует привести к единому времени, а именно началу расчётного года. Для этого необходимо использовать дисконтирование путем умножения соответствующих результатов и затрат на коэффициент дисконтирования соответствующего года t, который определяется по формуле 7.14:

|  |  |
| --- | --- |
|  | (7.14) |

где – норма дисконта (в долях единиц), равная или больше средней процентной ставки по банковским депозитам действующей на момент осуществления расчетов, равная 0.14;

– порядковый номер года периода реализации инвестиционного проекта (предполагаемый период использования разрабатываемого ПО пользователем и время на разработку).

Чистый дисконтированный доход рассчитывается по формуле 7.15:

|  |  |
| --- | --- |
| , | (7.15) |

где – расчетный период, лет;

– результат (экономический эффект), полученный в году t, руб.;

– затраты (инвестиции в разработку программного обеспечения) в году t, руб.

Сведем данные расчета показателей экономической эффективности в таблицу 7.3.

Таблица 7.3 – Расчет показателей экономической эффективности разработки и реализации программного продукта

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Показатели | Ед. изм. | Годы | | | |
| 2017 | 2018 | 2019 | 2020 |
| 1 | 2 | 3 | 4 | 5 | 6 |
| Результаты: | | | | | |
| Экономический эффект | руб. | 1913,79 | 2232,76 | 2232,76 | 1594,83 |

*Продолжение таблицы 7.3*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| Дисконтированный результат | руб. | 1913,79 | 1958,56 | 1718,04 | 1076,46 |
| Затраты: | | | | | |
| Затраты на разработку | руб. | 3935,53 |  |  |  |
| Дисконтированные вложения | руб. | 3935,53 |  |  |  |
| Экономический эффект: | | | | | |
| Чистый дисконтированный доход по годам | руб. | -2021,74 | 1958,56 | 1718,04 | 1076,46 |
| Чистый дисконтированный нарастающим итогом | руб. | -2021,74 | -63,18 | 1654,86 | 2731,32 |

Рентабельность инвестиций вычисляется по формуле 7.16:

|  |  |  |
| --- | --- | --- |
|  |  | (7.16) |

где ЧПср – среднегодовая величина чистой прибыли за расчетный период и высчитывается по формуле 7.17.

|  |  |  |
| --- | --- | --- |
|  |  | (7.17) |

Итого рентабельность инвестиций:

Чистый дисконтированный доход больше нуля, то есть проект эффективен, а значит инвестиции в разработку данного ПО экономически целесообразны.

В результате технико-экономического обоснования были определены следующие экономические показатели:

* + - чистая дисконтированная стоимость за четыре года работы программного продукта составит 2731,32 рублей;
    - срок окупаемости проекта равен трём годам;
    - рентабельность инвестиций составила 51%.

Таким образом, исходя из вышеперечисленных показателей можно сделать вывод, что проект является прибыльным и затраты на его разработку целесообразны.

# ЗАКЛЮЧЕНИЕ

выводы, основные преимущества и недостатки разработки (в виде списков)

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Wikipedia [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Цифровая звуковая рабочая станция.
2. Wikipedia [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Плагин>.
3. Wikipedia [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Virtual_Studio_Technology>.
4. Wikipedia [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Октавер>.
5. Лайонс, Р. Цифровая обработка сигналов: Второе издание/ Лайонс Р. – Пер с англ. – М. : ООО «Бином-Пресс», 2006 г. – 656 с.
6. Cooley J. W., Tukey J. W. An algorithm for the machine calculation of complex Fourier series // Mathematics of Computation. 1965. V. 19. No. 90. P. 297-301.
7. DSPLIB [Электронный ресурс]. – Режим доступа: <http://ru.dsplib.org/content/fft_dec_in_time.html>.
8. Wikipedia [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Дилэй>.
9. Audio Artillery [Электронный ресурс]. – Режим доступа: <http://audioartillery.com/projects/tonecore_dsp_dev_kit_guide>.
10. Wikipedia [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Реверберация>.
11. Open Source Acoustic Design [Электронный ресурс] : Algorithmic Reverberation. – Режим доступа: http://arqen.com/wp-content/docs/Hybrid-Convolution-Algorithmic-Reverb.pdf.
12. Christianfloisand wordpress [Электронный ресурс]. – Режим доступа: <https://christianfloisand.wordpress.com/2012/10/18/algorithmic-reverbs-the-moorer-design>/.
13. Wikipedia [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Juce>.
14. Juce [Электронный ресурс]. – Режим доступа: https://www.juce.com/doc/classes.
15. Juce [Электронный ресурс]. – Режим доступа: <https://www.juce.com/doc/tutorial_create_projucer_basic_plugin>.
16. GitHub [Электронный ресурс]. – Режим доступа: <https://github.com/>.

Палицын, В. А. Технико-экономическое обоснование дипломных проектов: методическое пособие в 4 ч. – Минск : БГУИР, 2005 г.

# ПРИЛОЖЕНИЕ А

(обязательное)

Листинг

# ПРИЛОЖЕНИЕ Б

(обязательное)

Спецификация программного дипломного проекта

# ПРИЛОЖЕНИЕ В

(обязательное)

Ведомость документов