Міністерство освіти і науки, молоді та спорту України
Міжнародний економіко-гуманітарний університет імені академіка Степана Дем'янчука
Факультет кібернетики

Кафедра математичного моделювання

ДИСЕРТАЦІЯ МАГІСТРА З ІНФОРМАТИКИ

на тему:

Розробка програмного забезпечення для обробки цифрового звуку

Робота допущена до захисту
Протокол № 6 від " 14" лютого 2012 р.
Завідувач кафедри
математичного моделювання
д. ф.-м. н., професор Джунь Й.В.

_____
(підпис)

Студента магістратури
факультету кібернетики
групи ІН-01М
спеціальності 8.080201
„Інформатика"
Вольського Артемія
Вікторовича

Науковий керівник:Лотюк Ю.Г.
канд.пед.наук,доцент
(підпис)
Науковий консультант:
Літнарович Р.М., к.т.н., доц.

(підпис)

Рівне – 2012

Зміст

Вступ

Цифровий звук (англ. Digital audio) — термін, що уживається на окреслення цифрових технологій роботи зі звуком. У вузькому сенсі слова, цифровий звук являє собою цифровий сигнал, в якому закодовано звук. В більш широкому сенсі поняття цифровий звук охоплює також цифрові технології обробки, зберігання та відтворення звуку. Цифровий звук принципово відрізняється від аналогового. Як правило, цифровий сигнал отримують з аналогового шляхом його конвертації у цифровий. Цей процес називається оцифруванням (оцифровуванням) і здійснюється за допомогою аналогово-цифрового перетворювача (АЦП). В музичній практиці використовують однак і сигнали, що мають винятково цифрове походження, наприклад, згенеровані на цифрових синтезаторах, в цьому випадку аналогово-цифрове перетворення не здійснюється.

Експериментальні цифрові звукозаписи існували з 1960-х. Комерційне продукування цифрових записів класичної та джазової музики починається на початку 1970-х, піонерами були японські компанії Denon, BBC та британський лейбл Decca. Перший 16-бітний PCM-запис у США був зроблений в 1976 році. В більшості випадків мікшування звуку не застосовувалось; цифровий стереозапис виготовлявся та використовувався як незмінний майстер-запис для наступного комерційного використання та позначався як "DDD" . Першим повністю цифровим (DDD) альбомом поп-музики став "Bop Till You Drop" гурту Ry Cooder, записаний в 1978. Цифровий звук виявився корисним для запису, обробки, масового виробництва та поширення аудіо.

Сфера використання цифрового звуку на даний час невпинно розширюється. Насамперед це рекреаційна сфера, але останнім часом із впровадженням WEB 2.0. відбувається інтеграція мультимедіа, і цифрового звуку зокрема в глобальну інформаційну мережу.

Особливістю сучасного IT суспільства є деанонімізація, і соціалізована генерація контенту. Тому, останнім часом можна спостерігати бурхливий ріст медіа-напрямків, пов’язаних із соціальними мережами: рух підкастерів, promodj, тощо. Тому виникає цілком обґрунтована потреба в технологіях обробки цифрового звуку,

які були б доступними широкому загалу аудиторії, не залежно від рівня технічної освіти та спеціальних знань.

Мета даної магістерської роботи полягає в дослідженні цифрового звуку як явища, технологій його обробки, а також в розробці програмного засобу для обробки цифрового звуку.

Актуальність даної роботи визначена невпинним ростом популярності цифрового формату звуку, а отже, потребою в програмних засобах, які дозволяють проводити обробку цифрового звуку.

Новизна даної роботи полягає у комплексному застосуванні алгоритмів оброки цифрового звуку з використанням відповідних методів наближених обчислень разом із ергономічним інтерфейсом, орієнтованим на інтуїтивне сприйняття та когнітивне самонавчання користувача.

Результатом роботи має стати комплексне дослідження цифрового звуку та методів і технологій його обробки. Практичним результатом має стати програмний засіб, який повинен надавати пересічному користувачу можливість редагування та обробки цифрового звуку.

Апробація роботи. Окремі розділи дисертації були докладені і отримали одобрення на наукових конференціях студентів і аспірантів у 2010 і 2011 роках, а також на науковому семінарі кафедри математичного моделювання.

Публікації. Основні положення дисертації опубліковані в монографії автора :
Вольський А.В. Розробка програмного забезпечення для обробки цифрового звуку. Науковий керівник Ю.Г.Лотюк. Науковий консультант Р.М.Літнарович. МЕГУ, Рівне, 2012.- 83 с.

Основні положення дисертації, що виносяться на захист:

- дослідження теоретичних основ цифрового звуку, та основних технологій його обробки;
- розробка програми для обробки цифрового звуку на основі отриманих теоретичних результатів;
- модифікація параметрів звукового сигналу;
- правка фрагментів звукового потоку (семплінг);
- обробка звукового ряду з використанням фільтрації (міксування);
- синтез нових елементів звукового ряду (біт-генератор);

Структура і об'єм роботи:

Магістерська дисертація складається із вступу, двох розділів, розбитих на підрозділи, висновків і списку використаних джерел. Обсяг дисертації 83 сторінки, 11 рисунків, Список використаної літератури із 15 найменувань, в тому числі 4 на іноземній мові.

Розділ І. Теоретичні основи обробки цифрового звуку.

1.1. Цифрове представлення звукових сигналів.

Традиційне аналогове представлення сигналів засноване на подібності (аналогічності) електричних сигналів (змін струму і напруги) представленим ними початковим сигналам (звуковому тиску, температурі, швидкості і т.п.), а також подібності форм електричних сигналів в різних точках підсилювального або передаючого тракту. Форма електричної кривої, що описує (також говорять - несучої) початковий сигнал, максимально наближена до форми кривої цього сигналу.

Таке уявлення найбільш точне, проте щонайменше спотворення форми несучого електричного сигналу неминуче спричинить за собою таке ж спотворення форми і сигналу що переноситься. В термінах теорії інформації, кількість інформації в несучому сигналі в точності рівна кількості інформації в сигналі початковому, і електричне представлення не містить надмірності, яка могла б захистити несучий сигнал від спотворень при зберіганні, передачі і підсиленні.

Цифрове представлення електричних сигналів покликане внести в них надмірність, що оберігає їх від дії паразитних перешкод. Для цього на несучий електричний сигнал накладаються серйозні обмеження - його амплітуда може приймати тільки два граничні значення - 0 і 1.

Вся зона можливих амплітуд в цьому випадку ділиться на три зони: нижня представляє нульові значення, верхня - одиничні, а проміжна є забороненою - всередину неї можуть потрапляти тільки перешкоди. Таким

чином, будь-яка перешкода, амплітуда якої менше половини амплітуди несучого сигналу, не робить впливу на правильність передачі значень 0 і 1. Перешкоди з більшою амплітудою також не роблять впливу, якщо тривалість імпульсу перешкоди відчутно менше тривалості інформаційного імпульсу, а на вході приймача встановлений фільтр імпульсних перешкод.

Сформований таким чином цифровий сигнал може переносити будь-яку корисну інформацію, яка закодована у вигляді послідовності бітів - нулів і одиниць; окремим випадком такої інформації є електричні і звукові сигнали. Тут кількість інформації в несучому цифровому сигналі значно більше, ніж в кодованому початковому, так що несучий сигнал має певну надмірність щодо початкового, і будь-які спотворення форми кривої несучого сигналу, при яких ще зберігається здатність приймача правильно розрізняти нулі і одиниці, не впливають на достовірність передаваної цим сигналом інформації. Проте у разі дії значних перешкод форма сигналу може спотворюватися настільки, що точна передача переносимої інформації стає неможливою - в ній з'являються помилки, які при простому способі кодування приймач не зможе не тільки виправити, але і знайти.

Для ще більшого підвищення стійкості цифрового сигналу до перешкод і спотворень застосовується цифрове надмірне кодування двох типів: перевірочні (EDC - Error Detection Code) і коректуючі (ECC - Error Correction Code) коди.

Цифрове кодування полягає в простому додаванні до початкової інформації додаткових бітів і/або перетворенні початкового бітового ланцюжка в ланцюжок більшої довжини і іншої структури. EDC дозволяє просто знайти факт помилки - спотворення або випадання корисної або появу помилкової цифри, проте інформація, що переноситься, в цьому випадку також спотворюється; ECC дозволяє зразу ж виправляти знайдені помилки, зберігаючи інформацію, що переноситься, незмінною. Для зручності і надійності інформацію розбивають на блоки (кадри), кожний з яких забезпечується власним набором цих кодів.

Кожен вигляд EDC/ECC має свою межу здатності знаходити і виправляти помилки, за яким знову починаються незнайдені помилки і спотворення інформації. Збільшення об'єму EDC/ECC щодо об'єму початкової інформації в загальному випадку підвищує здатність цих кодів до знаходження та коректування сигналу.

Як EDC популярний циклічний надмірний код CRC (Cyclic Redundancy Check), суть якого полягає в складному перемішуванні початкової інформації в блоці і формуванню коротких двійкових слів, розряди яких знаходяться в сильній перехресній залежності від кожного біта блоку.

Зміна навіть одного біта в блоці викликає значну зміну обчисленого по ньому CRC, і вірогідність такого спотворення бітів, при якому CRC не зміниться, надзвичайно мала навіть при коротких (декілька відсотків від довжини блоку) словах CRC. Як ECC використовуються коди Хеммінга (Hamming) і Ріда-Соломона (Reed-Solomon), які також включають в себе і функції EDC.

Інформаційна надмірність несучого цифрового сигналу приводить до значного (на порядок і більше) розширення смуги частот, що вимагається для його успішної передачі, в порівнянні з передачею початкового сигналу в аналоговій формі. Окрім власне інформаційної надмірності, до розширення смуги приводить необхідність збереження досить крутих фронтів цифрових імпульсів.

Окрім цілей захисту від завад, інформація в цифровому сигналі може бути піддана також лінійному або канальному кодуванню, задача якого - оптимізувати електричні параметри сигналу (смугу частот, постійну складову, мінімальну і максимальну кількість нульових/одиничних імпульсів в серії і т.п.) під характеристики реального каналу передачі або запису сигналу.

Одержаний несучий сигнал, у свою чергу, також є звичайним електричним сигналом, і до нього можуть бути застосовні будь-які операції з

такими сигналами - передача по кабелю, підсилення, фільтрування, модуляція, запис на магнітний, оптичний або інший носій і т.п. Єдиним обмеженням є збереження інформаційного вмісту - так, щоб при подальшому аналізі можна було однозначно виділити і декодувати інформацію, а з неї - початковий сигнал.

Початкова форма звукового сигналу - безперервна зміна амплітуди в часі - представляється в цифровій формі за допомогою "перехресної дискретизації" - за часом та рівнем.

Згідно теореми Котельникова, будь-який безперервний процес з обмеженим спектром може бути повністю описаний дискретною послідовністю його миттєвих значень, наступних з частотою, як мінімум удвічі перевищуючої частоту щонайвищої гармоніки процесу; частота *Fd* вибірки миттєвих значень (відліків) називається частотою дискретизації.

З теореми виходить, що сигнал з частотою *Fa* може бути успішно дискретизований за часом на частоті *2Fa* тільки в тому випадку, якщо він є чистою синусоїдою, бо будь-яке відхилення від синусоїдальної форми приводить до виходу спектру за межі частоти *Fa*. Таким чином, для часової дискретизації довільного звукового сигналу (що звичайно має, як відомо, плавно спадаючий спектр), необхідний або вибір частоти дискретизації із запасом, або примусове обмеження спектру вхідного сигналу нижче за половину частоти дискретизації.

Одночасно з часовою дискретизацією виконується амплітудна - вимірювання миттєвих значень амплітуди і їх представлення у вигляді числових величин з певною точністю. Точність вимірювання визначає співвідношення сигнал/шум і динамічний діапазон сигналу (теоретично це - взаємно-зворотні величини, проте будь-який реальний тракт має також і власний рівень шумів і перешкод).

Одержаний потік чисел (серій двійкових цифр), що описує звуковий сигнал, називають імпульсно-кодовою модуляцією або ІКМ (Pulse Code Modulation, PCM), оскільки кожен імпульс дискретизованого за часом сигналу представляється власним цифровим кодом.

Найчастіше застосовують лінійне квантування, коли числове значення відліку пропорційне амплітуді сигналу. Через логарифмічну природу слуху доцільнішим було б логарифмічне квантування, коли числове значення пропорційне величині сигналу в децибелах, проте це зв'язано з труднощами чисто технічного характеру.

Часова дискретизація і амплітудне квантування сигналу неминуче вносять в сигнал шумові спотворення, рівень яких прийнято оцінювати по формулі:

6N + 10lg (Fдискр/2Fмакс) + 3 (дБ),

де константа *3* варіюється для різних типів сигналів: для чистої синусоїди це 1.7 дБ, для звукових сигналів - від -15 до 2 дБ.

Звідси видно, що до зниження шумів в робочій смузі частот *0..Fмакс* приводить не тільки збільшення розрядності відліку, але і підвищення частоти дискретизації щодо *2Fмакс*, оскільки шуми квантування "розмазуються" по всій смузі аж до частоти дискретизації, а звукова інформація займає тільки нижню частину цієї смуги.

У більшості сучасних цифрових звукових систем використовуються стандартні частоти дискретизації 44.1 і 48 кГц, проте частотний діапазон сигналу звичайно обмежується біля 20 кГц для залишення запасу по відношенню до теоретичної межі. Також найбільш поширене 16-розрядне квантування по рівню, що дає граничне співвідношення сигнал/шум близько 98 дБ. У студійній апаратурі використовуються вищі рівні - 18-, 20- і 24-розрядне квантування при частотах дискретизації 56, 96 і 192 кГц. Це робиться для того, щоб зберегти вищі гармоніки звукового сигналу, які безпосередньо не

сприймаються слухом, але впливають на формування загальної звукової картини.

Для оцифровки більш вузькополосних і менш якісних сигналів частота і розрядність дискретизації можуть знижуватися; наприклад, в телефонних лініях застосовується 7- або 8-розрядна оцифровка з частотами 8..12 кГц.

Представлення аналогового сигналу в цифровому вигляді називається також імпульсно-кодовою модуляцією (ІКМ, PCM - Pulse Code Modulation), оскільки сигнал представляється у вигляді серії імпульсів постійної частоти (часова дискретизація), амплітуда яких передається цифровим кодом (амплітудна дискретизація). PCM-потік може бути як паралельним, коли всі біти кожного відліку передаються одночасно по декількох лініях з частотою дискретизації, так і послідовних, коли біти передаються один за одним з вищою частотою по одній лінії.

Сам цифровий звук і речі, що відносяться до нього, прийнято позначати загальним терміном Digital Audio; аналогова і цифрова частини звукової системи позначаються термінами Analog Domain і Digital Domain.

1.2. Перетворення АЦП та ЦАП.

Аналогово-цифровий і цифро-аналоговий перетворювачі є невід’ємними частинам кодування цифрового сигналу. Перший перетворює аналоговий сигнал в цифрове значення амплітуди, другоий виконує зворотне перетворення. У англомовній літературі застосовуються терміни ADC і DAC, а суміщений перетворювач називають codec (coder-decoder).

Принцип роботи АЦП полягає у вимірюванні рівня вхідного сигналу і видачі результату в цифровій формі. В результаті роботи АЦП безперервний аналоговий сигнал перетворюється на імпульсний, з одночасним вимірюванням амплітуди кожного імпульсу. ЦАП набуває на вході цифрове значення

амплітуди і видає на виході імпульси напруги або струму потрібної величини, які розташований за ним інтегратор (аналоговий фільтр) перетворює на безперервний аналоговий сигнал.

Для правильної роботи АЦП вхідний сигнал не повинен змінюватися протягом часу перетворення, для чого на його вході звичайно поміщається схема вибірки-зберігання, що фіксує миттєвий рівень сигналу і зберігає його протягом всього часу перетворення. На виході ЦАП також може встановлюватися подібна схема, що пригнічує вплив перехідних процесів усередині ЦАП на параметри вихідного сигналу.

При часовій дискретизації спектр одержаного імпульсного сигналу в своїй нижній частині *0..Fa* повторює спектр початкового сигналу, а вище містить ряд віддзеркалень (aliases, дзеркальних спектрів), які розташовані навколо частоти дискретизації *Fd* і її гармонік (бічні смуги). При цьому перше віддзеркалення спектру від частоти *Fd* у разі *Fd = 2Fa* розташовується безпосередньо за смугою початкового сигналу, і вимагає для його придушення аналогового фільтру (anti-alias filter) з високою крутизною зрізу. У АЦП цей фільтр встановлюється на вході, щоб виключити перекриття спектрів і їх інтерференцію, а в ЦАП - на виході, щоб подавити у вихідному сигналі надтональні перешкоди, внесені часовою дискретизацією.

В основному застосовується три конструкції АЦП:

- паралельні - вхідний сигнал одночасно порівнюється з еталонними рівнями набором схем порівняння (компараторів), які формують на виході двійкове значення. У такому АЦП кількість компараторів рівна (2 в ступені N) - 1, де N - розрядність цифрового коду (для восьмирозрядного - 255), що не дозволяє нарощувати розрядність понад 10-12.
- послідовного наближення - перетворювач за допомогою допоміжного ЦАП генерує еталонний сигнал, порівнюваний з

вхідним. Еталонний сигнал послідовно змінюється за принципом половинного розподілу (дихотомії), який використовується в багатьох методах пошуку прикладної математики, що сходиться. Це дозволяє завершити перетворення за кількість тактів, рівну розрядності слова, незалежно від величини вхідного сигналу.

- з вимірюванням часових інтервалів - широка група АЦП, що використовує для вимірювання вхідного сигналу різні принципи перетворення рівнів в пропорційні часові інтервали, тривалість яких вимірюється за допомогою тактового генератора високої частоти.

Серед АЦП з вимірюванням часових інтервалів переважають наступні три типи:

- послідовного рахунку, або одноразової інтеграції (single-slope) - в кожному такті перетворення запускається генератор лінійно зростаючої напруги, яка порівнюється з вхідним. Звичайно таку напругу одержують на допоміжному ЦАП, подібно АЦП послідовного наближення.
- подвійної інтеграції (dual-slope) - в кожному такті перетворення вхідний сигнал заряджає конденсатор, який потім розряджається на джерело опорної напруги з вимірюванням тривалості розряду.
- стежачі - варіант АЦП послідовного рахунку, при якому генератор еталонної напруги не перезапускається в кожному такті, а змінює його від попереднього значення до поточного.

Найпопулярнішим варіантом стежачого АЦП є sigma-delta, що працює на частоті *Fs*, яка значно (у 64 і більше разів) перевищує частоту дискретизації *Fd* вихідного цифрового сигналу. Компаратор такого АЦП видає значення зниженої розрядності (зазвичай однобітові - 0/1), сума яких на інтервалі дискретизації *Fd* пропорційна величині відліку. Послідовність малорозрядних значень піддається цифровій фільтрації і пониженню частоти проходження

(decimation), внаслідок чого виходить серія відліків із заданою розрядністю і частотою дискретизації *Fd*.

Для поліпшення співвідношення сигнал/шум і зниження впливу помилок квантування, яке у разі однобітового перетворювача виходить досить високим, застосовується метод формування шуму (noise shaping) через схеми зворотного зв'язку помилкового і цифрового фільтрування. В результаті застосування цього методу форма спектру шуму міняється так, що основна шумова енергія витісняється в область вище за половину частоти *Fs*, незначна частина залишається в нижній половині, і практично весь шум видаляється із смуги початкового аналогового сигналу.

ЦАП в основному будуються за трьома принципами:

- зважуючі - з підсумовуванням зважених струмів або напруг, коли кожен розряд вхідного слова вносить відповідний своїй двійковій вазі внесок в загальну величину одержуваного аналогового сигналу; такі ЦАП називають також паралельними або багаторозрядними (multibit).
- sigma-delta, з попередньою цифровою передискретизацією і видачею малорозрядних (звичайно однобітових) значень на схему формування еталонного заряду, які з такою ж високою частотою додаються до вихідного сигналу. Такі ЦАП носять також назву bitstream.
- з широтно-імпульсною модуляцією (ШІМ, Pulse Width Modulation, PWM), коли на схему вибірки-зберігання аналогового сигналу видаються імпульси постійної амплітуди і змінної тривалості, управляючи дозуванням вихідного заряду. На цьому принципі працюють перетворювачі MASH (Multi-stAge Noise Shaping - багатостадійне формування шуму) фірми Matsushita. Свою назву ці ЦАП одержали унаслідок застосування в них декількох послідовних формувачів шуму.

При використанні передискретизації в десятки разів (зазвичай - 64х..512х) стає можливим зменшити розрядність ЦАП без відчутної втрати якості сигналу; ЦАП з меншим числом розрядів володіють також кращою лінійністю.

Форма вихідного сигналу таких ЦАП є корисним сигналом, обрамленим значною кількістю високочастотного шуму, який, проте, ефективно пригнічується аналоговим фільтром навіть середньої якості.

ЦАП є "прямими" пристроями, в яких перетворення виконується простіше і швидше, ніж в АЦП, які в більшості своїй - послідовні і повільніші пристрої.

Передискретизація (oversampling) це дискретизація сигналу з частотою, що перевищує основну частоту дискретизації. Передискретизація може бути аналоговою, коли з підвищеною частотою робляться вибірки початкового сигналу, або цифровою, коли між вже існуючими цифровими відліками вставляються додаткові, розраховані шляхом інтерполяції. Інший спосіб отримання значень проміжних відліків полягає у вставці нулів, після чого вся послідовність піддається цифровій фільтрації. У АЦП використовується аналогова передискретизація, в ЦАП - цифрова.

Передискретизація використовується для спрощення конструкцій АЦП і ЦАП. За умов задачі на вході АЦП і виході ЦАП повинен бути встановлений аналоговий фільтр з АЧХ, лінійної в робочому діапазоні і круто спадаючої за його межами. Реалізація такого аналогового фільтру вельми складна; в той же час при підвищенні частоти дискретизації віддзеркалення спектру, що вносяться нею, пропорційно відсовуються від основного сигналу, і аналоговий фільтр може мати набагато меншу крутизну зрізу.

Інша перевага передискретизації полягає у тому, що помилки амплітудного квантування (шум дроблення), розподілені по всьому спектру квантованого сигналу, при підвищенні частоти дискретизації розподіляється по

ширшій смузі частот, так що на частку основного звукового сигналу доводиться менша кількість шуму. Кожне подвоєння частоти знижує рівень шуму квантування на 3 дБ; оскільки один двійковий розряд еквівалентний 6 дБ шуму, кожне збільшення частоти в чотири рази дозволяє зменшити розрядність перетворювача на одиницю.

Передискретизація разом із збільшенням розрядності відліку, інтерполяцією відліків з підвищеною точністю і висновком їх на ЦАП належній розрядності дозволяє поліпшити якість відновлення звукового сигналу. З цієї причини навіть в 16-розрядних системах нерідко застосовуються 18- і 20-розрядні ЦАП з передискретизацією.

АЦП і ЦАП з передискретизацією за рахунок значного зменшення часу перетворення можуть обходитися без схеми вибірки-зберігання.

1.3. Переваги та недіолки цифрового звуку.

Цифрове представлення звуку цінне перш за все можливістю нескінченного зберігання і тиражування без втрати якості, проте перетворення з аналогової форми в цифрову і назад все ж таки неминуче приводить до часткової його втрати. Найнеприємніші на слух спотворення, що вносяться на етапі оцифровки - гранулярний шум, що виникає при квантуванні сигналу по рівню через округлення амплітуди до найближчого дискретного значення. На відміну від простого широкосмугового шуму, що вноситься помилками квантування, гранулярний шум є гармонійними спотвореннями сигналу, найпомітнішими у верхній частині спектру.

Потужність гранулярного шуму обернено пропорційна кількості ступенів квантування, проте через логарифмічну характеристику слуху при лінійному квантуванні (постійна величина ступеня) на тихі звуки доводиться менше ступенів квантування, ніж на гучні, і в результаті основна густина нелінійних спотворень доводиться на область тихих звуків. Це приводить до обмеження динамічного діапазону, який в ідеалі (без урахування гармонійних спотворень)

був би рівний співвідношенню сигнал/шум, проте необхідність обмеження цих спотворень знижує динамічний діапазон для 16-розрядного кодування до 50-60 дБ.

Положення могло б врятувати логарифмічне квантування, проте його реалізація у реальному часі вельми складна і дорога.

Спотворення, що вносяться гранулярним шумом, можна зменшити шляхом додавання до сигналу звичного білого шуму (випадкового або псевдовипадкового сигналу), амплітудою в половину молодшого значущого розряду; така операція називається згладжуванням (dithering). Це приводить до незначного збільшення рівня шуму, зате ослабляє кореляцію помилок квантування з високочастотними компонентами сигналу і покращує суб'єктивне сприйняття. Згладжування застосовується також перед округленням відліків при зменшенні їх розрядності. По суті, dithering і noise shaping є окремими випадками однієї технології - з тією різницею, що в першому випадку використовується білий шум з рівномірним спектром, а в другому - шум із спеціально "формованим" спектром.

При відновленні звуку з цифрової форми в аналогову виникає проблема згладжування ступінчастої форми сигналу і придушення гармонік, що вносяться частотою дискретизації. Через неідеальність АЧХ фільтрів може відбуватися або недостатнє придушення цих перешкод, або надмірне ослаблення корисних високочастотних складових. Погано пригнічені гармоніки частоти дискретизації спотворюють форму аналогового сигналу (особливо у області високих частот), що створює враження "шорсткого", "брудного" звуку.

1.4. Технології та методи обробки цифрового звуку.

Цифровий звук обробляється за допомогою математичних операцій, вживаних до окремих відліків сигналу, або до груп відліків різної довжини. Виконувані математичні операції можуть або імітувати роботу традиційних аналогових засобів обробки (мікшування двох сигналів - складання,

посилення/ослаблення сигналу - множення на константу, модуляція - множення на функцію і т.п.), або використовувати альтернативні методи - наприклад, розкладання сигналу в спектр (ряд Фурьє), корекція окремих частотних складових, потім зворотна "збірка" сигналу із спектру.

Обробка цифрових сигналів підрозділяється на лінійну (у реальному часі, над "живим" сигналом) і нелінійну - над заздалегідь записаним сигналом. Лінійна обробка вимагає достатньої швидкодії обчислювальної системи (процесора); у ряді випадків неможливе поєднання необхідної швидкодії і якості, і тоді використовується спрощена обробка із зниженою якістю. Нелінійна обробка ніяк не обмежена в часі, тому для неї можуть бути використані обчислювальні засоби будь-якої потужності, а час обробки, особливо з високою якістю, може досягати декількох хвилин і навіть годин.

Для обробки застосовуються як універсальні процесори загального призначення - Intel 8035, 8051, 80x86, Motorola 68xxx, SPARC - так і спеціалізовані цифрові сигнальні процесори (Digital Signal Processor, DSP) Texas Instruments TMS xxx, Motorola 56xxx, Analog Devices ADSP-xxxx і ін.

Різниця між універсальним процесором і DSP полягає у тому, що перший орієнтований на широкий клас задач - наукових, економічних, логічних, ігрових і т.п., і містить великий набір команд загального призначення, в якому переважають звичні математичні і логічні операції.

DSP спеціально орієнтовані на обробку сигналів і містять набори специфічних операцій - складання з обмеженням, перемножування векторів, обчислення математичного ряду і т.п. Реалізація навіть нескладної обробки звуку на універсальному процесорі вимагає значної швидкодії і далеко не завжди можлива у реальному часі, тоді як навіть прості DSP нерідко справляються у реальному часі з відносно складною обробкою, а потужні DSP здатні виконувати якісну спектральну обробку відразу декількох сигналів.

Через свою спеціалізацію DSP рідко застосовуються самостійно - найчастіше пристрій обробки має універсальний процесор середньої потужності для управління всім пристроєм, прийому/передачі інформації, взаємодії з користувачем, і один або декілька DSP - власне для обробки звукового сигналу. Наприклад, для реалізації надійної і швидкої обробки сигналів в комп'ютерних системах застосовують спеціалізовані плати з DSP, через які пропускається оброблюваний сигнал, тоді як центральному процесорі комп'ютера залишаються лише функції управління і передачі.

Розглянемо основні методи обробки цифрового звуку.

1. Монтаж. Полягає у вирізуванні із запису одних ділянок, вставці інших, їх заміні, розмноженні і т.п. Називається також редагуванням. Всі сучасні звуко- і відеозаписи в тій чи іншій мірі піддаються монтажу.
2. Амплітудні перетворення. Виконуються за допомогою різних дій над амплітудою сигналу, які кінець кінцем зводяться до множення значень семплів на постійний коефіцієнт (посилення/ослаблення) або функцію-модулятор, що змінюється в часі (амплітудна модуляція). Окремим випадком амплітудної модуляції є формування огинаючої для додання стаціонарному звучанню розвитку в часі. Амплітудні перетворення виконуються послідовно з окремими семплами, тому вони прості в реалізації і не вимагають великого об'єму обчислень.
3. Частотні (спектральні) перетворення. Виконуються над частотними складовими звуку. Якщо використовувати спектральне розкладання - форму представлення звуку, в якій по горизонталі відлічуються частоти, а по вертикалі - інтенсивності цих частот, то багато частотних перетворень стають схожими на амплітудні перетворенням над спектром. Наприклад, фільтрація - посилення або ослаблення певних смуг частот - зводиться до накладення на

спектр відповідної амплітудної огинаючої. Проте частотну модуляцію таким чином представити не можна - вона виглядає, як зсув всього спектру або його окремих ділянок в часі по певному закону. Для реалізації частотних перетворень звичайно застосовується спектральне розкладання по методу Фурьє, яке вимагає значних ресурсів. Проте є алгоритм швидкого перетворення Фурьє (БПФ, FFT), який робиться в цілочисельній арифметиці і дозволяє вже на молодших моделях процесорів розвертати у реальному часі спектр сигналу середньої якості.

4. Фазові перетворення. Зводяться в основному до постійного зрушення фази сигналу або її модуляції деякою функцією або іншим сигналом. Завдяки тому, що слуховий апарат людини використовує фазу для визначення напряму на джерело звуку, фазові перетворення стереозвуку дозволяють одержати ефект звуку, що обертається, хору і йому подібні.
5. Часові перетворення. Полягають в додаванні до основного сигналу його копій, зсунутих в часі на різні величини. При невеликих зрушеннях (порядку менше 20 мс) це дає ефект розмноження джерела звуку (ефект хору), при великих - ефект луни.
6. Формантні перетворення. Є окремим випадком частотних і оперують з формантами - характерними смугами частот, що зустрічаються в звуках, вимовних людиною. Кожному звуку відповідає своє співвідношення амплітуд і частот декількох формант, яке визначає тембр і розбірливість голосу. Змінюючи параметри формант, можна підкреслювати або затушовувати окремі звуки, міняти одну голосну на іншу, зсовувати регістр голосу і т.п.

Розглянемо основні ефекти, які можуть використовуватись при роботі із цифровим звуком.

- вібрато - амплітудна або частотна модуляція сигналу з невеликою частотою (до 10 Гц). Амплітудне вібрато також носить назву тремоло; на слух воно сприймається, як завмирання або тремтіння звуку, а частотне - як "завивання" або "плавання" звуку (типова несправність механізму магнітофона).
- динамічна фільтрація (wah-wah - "вау-вау") - реалізується зміною частоти зрізу або смуги пропускання фільтру з невеликою частотою. На слух сприймається, як обертання або затуляння/відкриття джерела звуку - збільшення високочастотних складових асоціюється з джерелом, обернутим на слухача, а їх зменшення - з відхиленням від цього напряму.
- фленжер (flange - облямівка, гребінь). Назва походить від способу реалізації цього ефекту в аналогових пристроях - за допомогою так званих гребінчастих фільтрів. Полягає в додаванні до початкового сигналу його копій, зсунутих в часі на невеликі величини (до 20 мс) з можливою частотною модуляцією копій або величин їх тимчасових зрушень і зворотним зв'язком (сумарний сигнал знову копіюється, зсовується і т.п.). На слух це відчувається як "дроблення", "розмазання" звуку, виникнення биття - різницевих частот, характерних для гри в унісон або хорового співу, тому фленжери з певними параметрами застосовуються для отримання хорового ефекту (chorus). Міняючи параметри фленжера, можна в значній мірі змінювати первинний тембр звуку.
- реверберація (reverberation - повторення, віддзеркалення). Виходить шляхом додавання до початкового сигналу затухаючої серії його зсунутих в часі копій. Це імітує загасання звуку в приміщенні, коли за рахунок багатократних віддзеркалень від стін, стелі і інших поверхонь звук отримує повноту і гулкість, а після припинення звучання джерела затухає не відразу, а поступово. При цьому час між послідовними відгомонами (приблизно до 50 мс) асоціюється з

величиною приміщення, а їх інтенсивність - з його гулкістю. По суті, ревербератор є окремим випадком фленжера із збільшеною затримкою між відлуннями основного сигналу, проте особливості слухового сприйняття якісно розрізняють ці два види обробки.

- луна (echo). Реверберація з ще більш збільшеним часом затримки - вищі 50 мс. При цьому слух перестає суб'єктивно сприймати віддзеркалення, як призвуки основного сигналу, і починає сприймати їх як повторення. Луна звичайно реалізується так само, як і природне - із загасанням копій, що повторюються.
- дістошн (distortion - спотворення) - навмисне спотворення форми звуку, що додає йому різкий, скрегочучий відтінок. Наибольшее застосування одержав як гітарний ефект (класична гітара heavy metal). Виходить перепідсиленням початкового сигналу до появи обмежень в підсилювачі (зрізу верхівок імпульсів) і навіть його самозбудження. Завдяки цьому початковий сигнал стає схожий на прямокутний, чому в ньому з'являється велика кількість нових частотних складових, що різко розширюють спектр. Цей ефект застосовується в різних варіаціях (fuzz, overdrive і т.п.), що розрізняються способом обмеження сигналу (звичне або згладжене, весь спектр або смуга частот, весь амплітудний діапазон або його частина і т.п.), співвідношенням початкового і спотвореного сигналів у вихідному, частотними характеристиками підсилювачів (наявність/відсутність фільтрів на виході).
- компресія - стиснення динамічного діапазону сигналу, коли слабкі звуки посилюються сильніше, а сильні - слабкіші. На слух сприймається як зменшення різниці між тихим і гучним звучанням початкового сигналу. Використовується для подальшої обробки методами, чутливими до зміни амплітуди сигналу. У звукозаписі використовується для зниження відносного рівня шуму і запобігання перевантаженням. Як гітарна приставка дозволяє

значно (на десятки секунд) продовжити звучання струни без загасання гучності.

- фейзер (phase - фаза) - змішування початкового сигналу з його копіями, зсунутими по фазі. По суті справи, це окремий випадок фленжера, але з набагато простішою аналоговою реалізацією (цифрова реалізація однакова). Зміна фазових зрушень результуючих сигналів приводить до придушення окремих гармонік або частотних областей, як в багатосмуговому фільтрі.
- вокодер (voice coder - кодувальник голосу) - синтез мови на основі довільного вхідного сигналу з багатим спектром. Мовний синтез реалізується за допомогою формантних перетворень: виділення з сигналу з достатнім спектром потрібного набору формант з потрібними співвідношеннями додає сигналу властивості відповідного голосного звуку. Спочатку вокодери використовувалися для передачі кодованої мови: шляхом аналізу початкового мовного сигналу з нього виділялася інформація про зміну положень формант (перехід від звуку до звуку), яка кодувалася і передавалася по лінії зв'язку, а на приймальному кінці блок керованих фільтрів і підсилювачів синтезував мову наново. Подаючи на блок мовного синтезу звучання, наприклад, електрогітари і вимовляючи слова в мікрофон блоку аналізу, можна одержати ефект "розмовляючої гітари"; при подачі звучання з синтезатора виходить відомий "голос робота", а подача сигналу, близького по спектру до коливань голосових зв'язок, але відмінного по частоті, міняє регістр голосу - чоловічий на жіночий або дитячий, і навпаки.

Ще однією особливістю, яку слід враховувати при роботі із цифровим звуком є jitter - тремтіння (швидкі коливання) фази синхросигналів в цифрових системах, що призводить до нерівномірності в часі моментів спрацьовування тактованих цими сигналами цифрових пристроїв.

Самі по собі цифрові пристрої нечутливі до таких коливань, поки вони не досягають значної величини в порівнянні із загальною тривалістю імпульсів, проте в "прикордонних" пристроях, що знаходяться на стику цифрової і аналогової частин схеми - АЦП і ЦАП - джіттер призводить до нерівномірності моментів спрацьовування компараторів АЦП або ключів ЦАП, що в свою чергу призводить до порушення правильності форми аналогового сигналу.

Для високочастотних компонент сигналу тремтіння фази призводить до "розмивання" звуку - порушенню суб'єктивної просторової локалізації джерел, оскільки слухове сприйняття локалізації базується в основному на фазових, а не на амплітудних співвідношеннях стереоканалів.

Джіттер може виникати через будь-яку нестабільність напруг і струмів у області ЦАП/АЦП. Наприклад, коливання живлячих напруг змінюють частоту опорного генератора, електромагнітні наведення спотворюють форму цифрових сигналів. Навіть якщо ці спотворення не змінюють інформаційного вмісту сигналу - укладеної в ньому бітової послідовності, вони можуть порушити рівномірність вхідного звукового сигналу в АЦП або видачу вихідного сигналу з ЦАП і привести до спотворень форми, особливо помітною у області високих частот.

Величина джіттера позначає максимальне абсолютне відхилення моменту переходу тактового сигналу з одного стану в інший від розрахункового значення, і вимірюється в секундах. Для систем середньої якості допустима величина джіттера складає порядка 100 пікосекунд, для систем класу Hi-Fi її прагнуть гранично мінімізувати.

Для боротьби з джіттером використовується тактуюче АЦП і ЦАП високостабільними генераторами, а для придушення нерівномірності цифрового потоку, що поступає на ЦАП - проміжними буферами типа FIFO (черга). Для зменшення впливу перешкод застосовуються звичні методи - екранування, розв'язки, виключення "земляних петель", роздільні джерела

живлення, живлення критичних схем від акумулятора і т.п. Добрі результати дають зовнішні модулі ЦАП, в яких реалізовані описані методи, - наприклад, Audio Alchemy DAC-in-the-Box і інші.

Необхідно розрізняти "прикордонний" джіттер, діючий на межах аналогової і цифрової частини схеми - у області АЦП або ЦАП, і "внутрішній", що виникає в будь-яких інших ділянках чисто цифрової схеми.

Вплив на звуковий сигнал має тільки "прикордонний" джіттер, бо тільки він безпосередньо пов'язаний з перетворенням аналогового звукового сигналу. Весь "внутрішній" джіттер при грамотній побудові схеми повинен повністю пригнічуватися в інтерфейсних ланцюгах, проте некоректна реалізація може пропускати його і безпосередньо на ЦАП/АЦП.

Виникаючий в ланцюгах формування, обробки, передачі, запису і читання цифрових сигналів "внутрішній" джіттер цілком може розповсюджуватися по системі, виходити за її межі і переноситися між системами через цифрові інтерфейси передачі або цифрові ж носії інформації. При цьому величина джіттера може як ослаблятися, так і посилюватися. При використанні інтерфейсів передачі з "вбудованим" (embedded) синхросигналом, а також при читанні з будь-якого носія, приймальна сторона вимушена синхронізуватися з передавачем шляхом використання систем фазового автопідстроювання частоти (ФАПЧ, Phase Locked Loop - PLL), яка вносить додаткові тремтіння, будучи не в змозі миттєво відстежувати зміни фази і частоти сигналу, що приймається.

Один з можливих способів ослаблення джіттера при передачі - використання синхронних інтерфейсів з окремим тактовим сигналом (Word Clock), а ще краще - асинхронних двонаправлених з можливістю узгодження темпу передачі, на зразок RS-232. В цьому випадку сторони можуть не боятись можливого опустіння або переповнювання буфера на приймальному кінці, передача може виконуватися блоками з вищою швидкістю, чим йде потік звуку,

а приймальна сторона може використовувати повністю незалежний стабільний генератор для витягання відліків з буфера. Проте все це має сенс тільки у тому випадку, коли приймач працює безпосередньо на ЦАП - при записі на носій нерівномірності такої величини впливу на якість звуку не дають.

Таким чином, в коректно реалізованій системі всі види джіттера, що виникають в чисто цифрових блоках і між ними, є "внутрішніми" і повинні бути пригнічені до передачі цифрового сигналу на ЦАП для крайового перетворення. Це може бути зроблено за допомогою проміжного буфера, схеми ФАПЧ з плавною зміною частоти генератора (повільна зміна в невеликих межах, на відміну від тремтіння, практично не відчувається на слух), або яким-небудь іншим методом.

Для слухової оцінки звукового сигналу його необхідно відтворити або одночасно на двох різних системах, або послідовно - на одній.

Навіть якщо в обох випадках сам цифровий сигнал буде однаковим, набір супутніх умов - апарат, носій, його мікроструктура, первинні сигнали при зчитуванні інформації, особливості роботи декодерів, спектр аналогових шумів і перешкод - майже завжди буде різний. Всі ці побічні процеси можуть створювати паразитні наведення, що спотворюють форму цифрового сигналу, породжують джіттер, і впливають на ланцюги живлення та інші аналогові компоненти системи. У правильно сконструйованих і ретельно виконаних апаратах всі ці впливи повинні бути пригнічені до рівня, недоступного сприйняттю, проте для більшості побутових і особливо бюджетних апаратів це не так.

Можуть бути і прозаїчніші причини для виникнення різниці - такі, як нестійке зчитування цифрового носія, при якому декодер не в змозі однозначно відновити закодований звуковий сигнал і вимушений вдаватися до його інтерполяції, яка погіршує якість звучання.

Така ж інтерполяція або гасіння відбувається у разі помилкового їх прийому по цифрових міжсистемних інтерфейсах, що може бути викликане поганою якістю або надмірною довжиною кабелю, дією на нього сильних перешкод, несправністю приймача або передавача, поганою їх сумісністю і т.п. Тому питання про порівняння звучання повинне розглядатися тільки після того, як доведена ідентичність цифрових потоків, що поступають на крайовий ЦАП. Під ЦАП тут повинен розумітися саме неподільний, "самий останній" перетворювач, а не довільний складний пристрій, що одержує на вході цифровий сигнал і видає на виході аналоговий.

1.5. Особливості зберігання якості звуку при проведенні цифрових перетворень.

Оскільки будь-який цифровий сигнал представляється реальною електричною кривою напруги або струму - його форма так чи інакше спотворюється при будь-якій передачі, а "заморожений" для зберігання сигнал (сигналограмма) схильний до деградації через звичні фізичні причини. Всі ці дії на форму несучого сигналу є перешкодами, які до певної величини не змінюють інформаційного змісту сигналу, як окремі спотворення і випадання букв в словах звичайно не заважають правильному розумінню цих слів, причому надмірність інформації, як і збільшення довжини слів, підвищує вірогідність успішного розпізнавання.

Іншими словами, сам несучий сигнал може спотворюватися, проте інформація, що ним переноситься - закодований звуковий сигнал - в абсолютній більшості випадків залишається незмінною.

Для того, щоб якість несучого сигналу не погіршувалася, будь-яка передача корисної звукової інформації - копіювання, запис на носій читання з нього - обов'язково повинна включати операцію відновлення форми несучого сигналу, а в ідеалі - і первинного цифрового виду сигналу інформаційного, і лише після цього наново сформований несучий сигнал може бути переданий

наступному споживачу. У разі прямого копіювання без відновлення якість цифрового сигналу погіршується, хоча він як і раніше повністю містить всю інформацію. Проте після багатократного послідовного копіювання або тривалого зберігання якість погіршується настільки, що починають виникати непоправні помилки, що необоротно спотворюють інформацію. Тому копіювання і передачу цифрових сигналів необхідно вести тільки в цифрових пристроях, а при зберіганні на носіях - своєчасно "освіжати" не чекаючи необоротної деградації. Правильно передана або оновлена цифрова сигналограмма якості не втрачає і може копіюватися і існувати вічно в абсолютно незмінному вигляді.

Проте, не слід забувати, що коректуюча здатність будь-якого коду кінцева, а реальні носії далекі від ідеальних, тому виникнення непоправних помилок - на така вже рідкісна річ, особливо при неакуратному поводженні з носієм. При читанні з нових і правильно збережених носіїв в якісних і надійних апаратах таких помилок практично не виникає, проте при старінні, забрудненні і пошкодженні носіїв і зчитуючих систем їх стає більше. Одиночна невиправлена помилка майже завжди непомітна на слух завдяки інтерполяції, проте вона приводить до спотворення форми початкового звукового сигналу, а накопичення таких помилок з часом починає відчуватися і на слух. Окрему проблему складає складність реєстрації невиправлених помилок, а також перевірки ідентичності оригінала і копії.

Перш за все, необхідно розрізняти "спотворюючі" і "неспотворюючі" види обробки. До перших відносяться операції, що змінюють форму і структуру сигналу - змішування, посилення, фільтрація, модуляція і т.п., до других - операції монтажу (вирізка, вклейка, накладення) і перенесення (копіювання).

Якість сигналу може страждати тільки при "спотворюючій" обробці, причому будь-який - і аналоговий, і цифровий. У першому випадку це відбувається в результаті внесення шумів, гармонійних, інтермодуляційних і інших спотворень у вузлах аналогового тракту, в другому - завдяки кінцевій

точності квантування сигналу і математичних обчислень. Всі цифрові обчислення виконуються в деякій розрядній сітці фіксованої довжини - 16, 20, 24, 32, 64, 80 і більш битий; збільшення розрядності сітки підвищує точність обчислень і зменшує помилки округлення, проте в загальному випадку не може виключити їх повністю. Кінцева точність квантування первинного аналогового сигналу призводить до того, що навіть при абсолютно точній обробці одержаного цифрового сигналу квантоване значення кожного відліку все одно відрізняється від свого ідеального значення. Для мінімізації спотворень при обробці в студіях вважають за краще обробляти і зберігати сигналограмми на майстер-носіях з підвищеним дозволом (20, 24 або 32 розряди), навіть якщо результат тиражуватиметься на носії з меншим дозволом.

Окрім власне помилок обчислень і округлення, на точність сильно впливає вибір представлення числових відліків сигналу при обробці.

Традиційне уявлення PCM з так званою фіксованою крапкою (fixed point), коли відліки представляються цілими числами, найбільш зручне і забезпечує мінімум накладних витрат, проте точність обчислень залежить від масштабу операцій - наприклад, при множенні утворюються числа удвічі більшої розрядності, які потім доводиться приводити назад до розрядності початкових відліків, а це може привести до переповнювання розрядної сітки.

Компромісним варіантом служить проміжне збільшення розрядності відліків (наприклад, 16->32), що знижує вірогідність переповнювання, проте вимагає більшої обчислювальної потужності, об'єму пам'яті і вносить додаткові спотворення при зворотному пониженні розрядності. Крім того, зниженню погрішності сприяє правильний вибір послідовності комутативних операцій, групування дистрибутивних операцій, облік особливостей роботи конкретного процесора і т.п.

Іншим способом збільшення точності є перетворення відліків у форму з плаваючою крапкою (floating point) з розділенням на значущу частину -

мантису і показник величини - порядок. У цій формі всі операції зберігають розрядність значущої частини, і множення не приводить до переповнювання розрядної сітки. Проте, як саме перетворення між формами з фіксованою і плаваючою крапкою, так і обчислення в цій формі вимагають на порядки більшої швидкодії процесора, що сильно утрудняє їх використовування у реальному часі.

Не дивлячись на те, що якість сигналу неминуча, хоча і незначно, погіршується при будь-якій "спотворюючій" цифровій обробці, деякі операції за певних умов є повністю і однозначно оборотними.

Наприклад, посилення сигналу по амплітуді в три рази полягає в множенні кожного відліку на три; якщо ця операція виконувалася з фіксованою крапкою і при цьому не виникло переповнювання, за допомогою розподілу на три потім можна буде повернути всі відліки в початковий стан, тим самим повністю відновивши первинний стан сигналу. І в той же час після множення кожен відлік виявиться збільшеним точно в три рази, тому помилка щодо початкового аналогового сигналу, внесена при квантуванні, також збільшиться в середньому в три рази, тим самим погіршивши загальну якість сигналу.

Сказане вище демонструє, що погіршення якості при "спотворюючій" цифровій обробці зовсім не обов'язково накопичується з часом, хоча в більшості реальних застосувань відбувається саме так. Крім того, це не означає, що будь-яка операція цифрового посилення завжди буде однозначно оборотною - це залежить від багатьох особливостей застосування операції. Проте, грамотно і якісно реалізована цифрова обробка може давати істотно менший рівень спотворень, ніж така ж аналогова, хіба що це будуть спотворення різних видів.

Тільки у тому випадку, коли в процесі перетворення застосовуються "спотворюючі" операції - зміна розрядності відліку, частоти дискретизації, фільтрування, стиснення з втратами і т.п. Просте збільшення розрядності

відліку із збереженням частоти дискретизації буде неспотворюючим, проте таке ж збільшення, зв'язане із застосуванням згладжуючої функції - вже немає. Зменшення розрядності відліку завжди є спотворюючою операцією, крім випадку, коли перетворювані відліки були одержані таким же простим збільшенням розрядності - рівної або меншої.

Багато форматів відрізняються один від одного тільки порядком бітів в слові, відліків лівого і правого каналів в потоці і службовою інформацією - заголовками, контрольними сумами, завадозахиснимиі кодами і т.п. Точний спосіб перевірки неспотворюваності сигналу полягає в перетворенні декількох різних потоків (файлів) формату F1 у формат F2, а потім назад в F1. Якщо інформаційна частина кожного потоку (файлу) при цьому буде ідентична початковою - даний вид перетворення можна вважати неспотворюючим.

Під інформаційною частиною потоку (файлу) розуміється власне набір даних, що описують звуковий сигнал; решта частини вважається службовою і на форму сигналу в загальному випадку не впливає. Наприклад, якщо в службовій частині файлу або потоку передбачене поле для часу його створення (передачі), то навіть у разі повного збігу інформаційних частин двох різних файлів або потоків їх службові частини виявляться різними, і це бути зафіксовано логічним аналізатором у разі потоку або програмою побайтного порівняння - у разі файлу.

Окрім цього, часове зрушення одного сигналу щодо іншого, що виникає при вирівнюванні цифрового потоку по межах слів або блоків і полягає в додаванні нульових відліків в початок і/або в кінець файлу або потоку, також приводить до їх уявного цифрового неспівпадання. У таких ситуаціях для перевірки ідентичності цифрових сигналів необхідно користуватися спеціальною апаратурою або програмою.

Для "перегонки" звуку між спеціалізованими системами, що мають сумісні цифрові інтерфейси, достатньо з'єднати їх цифровим кабелем і

переписати звук з однієї системи на іншу; у ряді поєднань пристроїв при цьому можливе погіршення якості сигналу через зменшення розрядності відліку, передіскретізациі або стиснення звуку. Наприклад, при копіюванні звуку між однаковими системами через інтерфейс S/PDIF стислий звуковий потік на передаючій стороні піддається відновленню, а на приймальні - повторному стисненню. Унаслідок несиметричності алгоритму ATRAC в звук при повторному стисненні будуть внесені додаткові спотворення.

Для перетворення комп'ютерного файлу в інший формат використовуються програми-конвертори: WAV2AIFF/AIFF2WAV, Convert, AWave і інші - на IBM PC, SoundExtractor, SampleEditor, BST - на Apple Mac.

Обмін звуковою інформацією між комп'ютерною і спеціалізованою системою нерідко можливий декількома способами: Пряме перенесення по цифровому інтерфейсу, якщо у обох систем є сумісні цифрові інтерфейси. При цьому на комп'ютерній системі використовується програма запису/відтворення, що формує або відтворююча стандартний для даної системи звуковий файл.

Розділ ІІ. Розробка програмного забезпечення для обробки цифрового звуку.

2.1. Аналіз та постановка задачі.

Проведемо аналіз предметноїгалузі та постановку задачі по розробці програмного рішення для обробки цифрового звуку.

Перш за все на етапі загальносистемного проектування необхідно визначити концепцію майбутньої системи, яка визначатиме подальний хід проектування вимог до функціональних характеристик.

Проектування розпочинається з визначення цільової аудиторії, на яку має бути розрахований програмний засіб, що розробляється. Враховуючи наявні технічні можливості, та тенденції розвитку ІТ-комьюніті, слід звернути увагу на цільову аудиторію підкастерів (авторів аудіоблогів) а також на аудиторію осіб, що починають займатись обробкою і створенням електронної музики, але не мають досвіду та необхідних знань для використання професійного програмного забезпечення, яке є досить складним і вимагає спеціальних професійних знань та умінь.

Виходячи із даної цільової аудиторії можна виділити ряд рис, яким має відповідати розроблювана програма. Насамперед програма повинна забезпечувати можливість швидкого редагування цифрового звуку, та обробку його за допомогою певного набору ефектів. Разом з цим програма має бути досить простою у використанні для того, щоб її функціоналом могла користуватись людина, яка не володіє спеціальними знаннями в сфері обробки звуку.

Отже, виділимо окремі задачі, які необхідно реалізувати в процесі розробки:

1. Програма повинна забезпечувати можливість обробки як існуючого звукового потоку, так і потокового сигналу із цифрового чи аналогового носія, а також можливість комбінувати ці джерела звуку. Слід забезпечити можливість кодування та деконування у найбільш широко вживані стандарти стиснення аудіоданих.
2. Необхідно надати функціонал керування частотними характеристиками звукового потоку (ширина полоси, кількість каналів, бітрейт, тощо).
3. Слід забезпечити можливість для роботи з окремими фрагментами звукового ряду, тобто створити функціонал для роботи із семплами.
4. Потрібно надати користувачу можливість додавати до звукового ряду певні ефекти.

Програма повинна мати зручний та інтуїтивно зрозумілий користувацький інтерфейс. Всі елементи керування повинні бути стандартизовані та згруповані відповідно до правил ергономіки.

Програма повинна відповідати сучасним вимогам до програмного забезпечення, та працювати у всіх сучасних операційних системах сімейства Microsoft Windows.

2.2. Аналіз ринку програмного забезпечення для обробки цифрового звуку.

На ринку програмного забезпечення існує досить велика кількість різних програмних засобів для роботи із цифровим звуком. Розглянемо найбільш відомі з програм цього типу, та визначимо їх переваги та недоліки.

Першою програмою, яку слід розглянути є Adobe Audition (рис. 2.1.)

Adobe Audition (Cool Edit Pro) - професійний інструмент для роботи з аудіо-файлами, призначений для фахівців у області обробки аудіо і

відеопродукції. Adobe Audition пропонує необмежені можливості мікшування, редагування, створення майстер-копій і обробки звукових спецефектів. Продукт суміщає гнучкість технологічного процесу з граничною простотою у використанні і дозволяє створювати різноманітну аудіо-продукцію високої якості. Adobe Audition - це повноцінна студія звукозапису, оснащена гнучкими і простими у використовуванні інструментами.

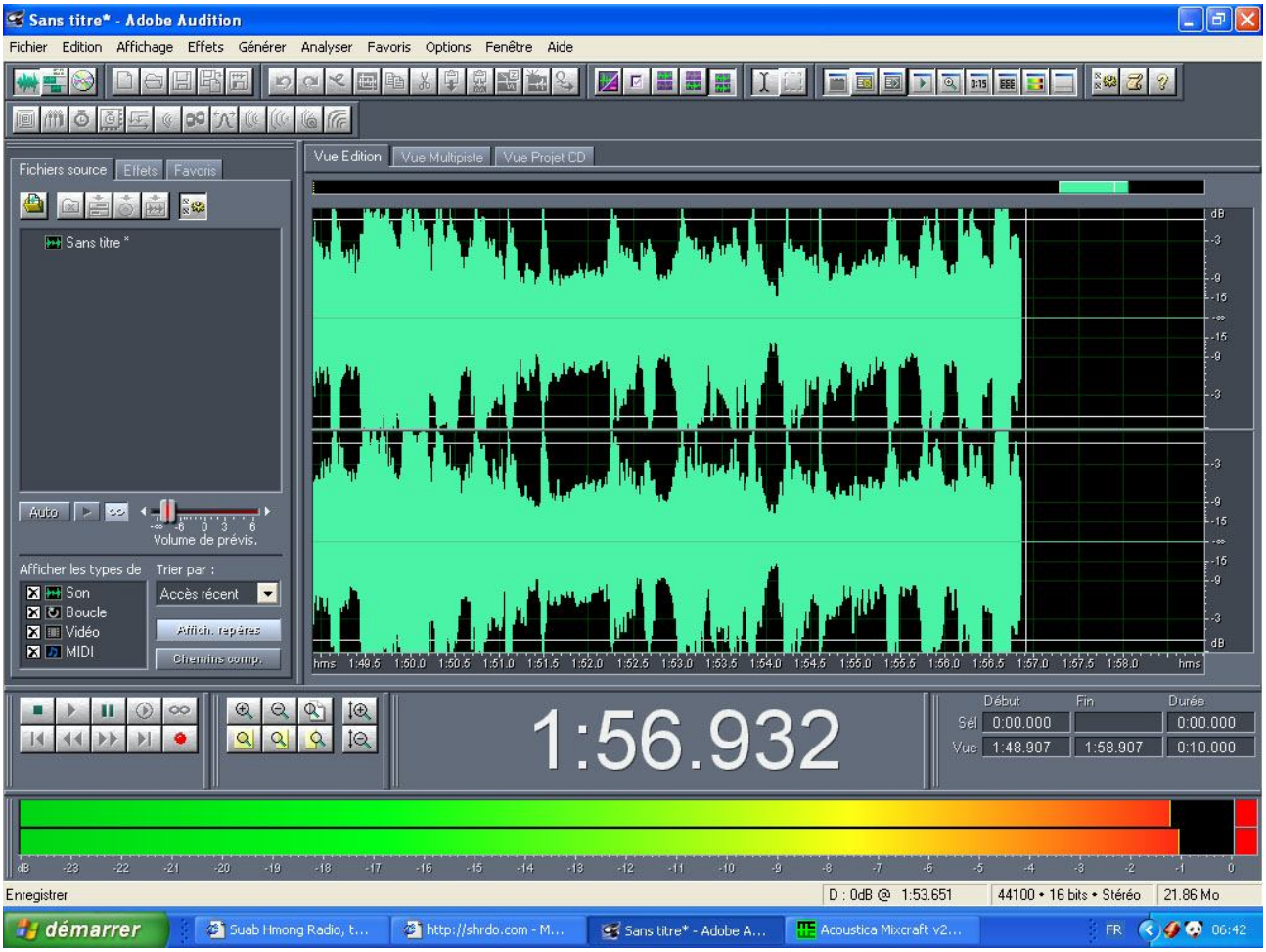

Рис. 2.1. Інтерфейс програми Adobe Audition.

Основною перевагою даного програмного засобу є його надзвичайно висока функціональність. Основний недолік – користувач повинен мати знання із принципів будови і процесінгу цифрового звуку. Таким чином вхідний поріг використання обмежується рівнем професійних знань.

Наступною програмою, яку ми розглянемо, є звуковий редактор Audicity. Даний програмний продукт поширюється за ліцензцією open-source, тобто на відміну від свого попередника є безкоштовним.

Програма має досить широкий функціонал, що є її основною перевагою.

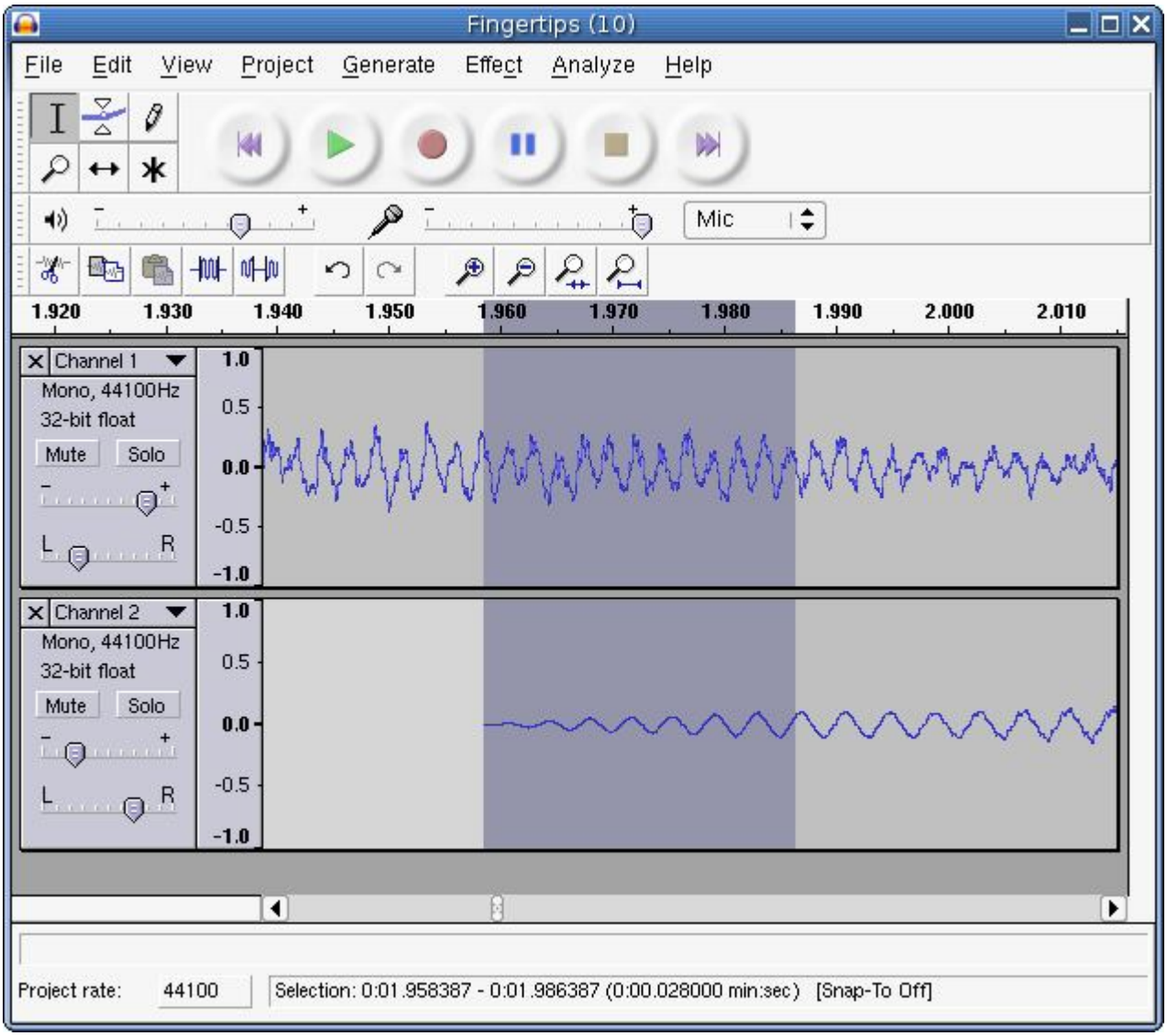

Рис. 2.2. Аудіо-редактор Audicity.

Основним недоліком даного програмного продукту є те, що він має не дружній до користувача інтерфейс, через що користуватись даною програмою для нового користувача є досить складно.

Наступною програмою, яку ми розглянемо, є Audio Editor Pro. Даний аудіоредактор дозволяє записувати і редагувати mp3, wma, wav файли із застосуванням різноманітних цифрових спецефектів і фільтрів (20 аудіоефектів і 6 фільтрів), а також конвертувати файли з одного формату в інший і переписувати CD на жорсткий диск. Підтримуються mp3 VBR кодек, Windows Media 9, ID3-тэги, а також завантаження даних з CDDB.

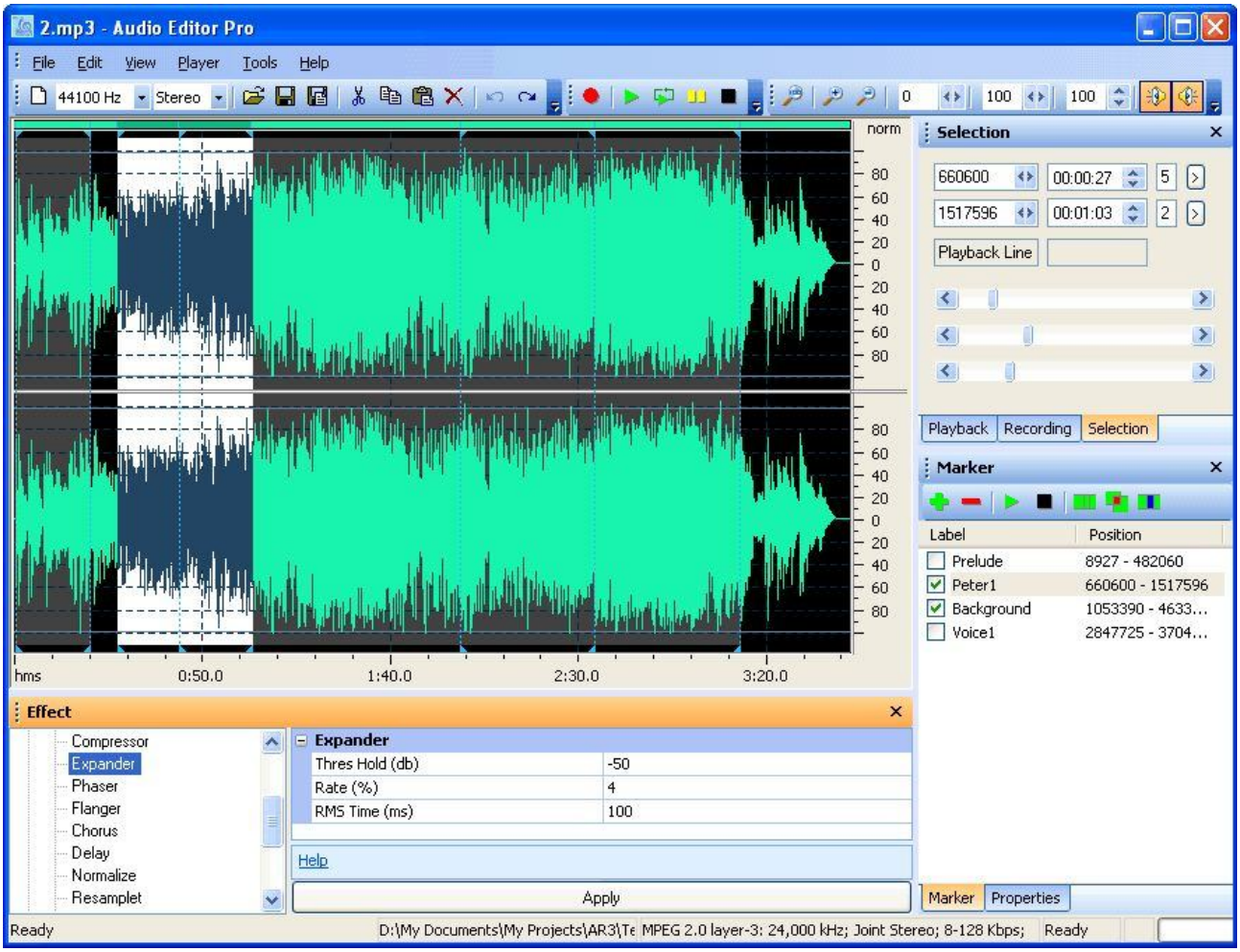

Рис. 2.3. Інтерфейс Audio Editor Pro.

Дана програма є відносно простою у використанні, і разом з тим дружньою до користувача, однак має суттєвий недолік – для використання необхідно придбати ліцензію, що робить дану програму не зовсім доступною для пересічного користувача.

При розробці програми необхідно скористатись досвідом розглянутих вище програмних засобів, і розробити з однієї сторони максимально спрощений і зрозумілий для кожного користувача програмний продукт. З іншої сторони, не зважаючи на простоту необхідно забезпечити виконання всього необхідного функціоналу при роботі із цифровим звуком. Тому слід одночасно враховувати обидва ці напрямки розробки.

2.3. Обгрунтування засобів реалізації.

В якості засобу реалізації оберемо систему інтегрованої розробки додатків Borland Delphi 7. Проаналізуємо особливості даного середовища.

Delphi — це інтегроване середовище швидкої розробки програмного забезпечення для роботи під Microsoft Windows. Воно підтримує розробку Windows-застосунків на мові програмування Delphi, яка є наступницею мови Object Pascal.

Delphi 2007, одинадцята версія, входить до складу CodeGear RAD Studio 2007, яка також підтримує розробку на C++ для 32-бітної Microsoft Windows, а також на Delphi і C# для платформи Microsoft .NET. У Delphi 2009, що випущена у серпні 2008-ого, бібліотеки VCL та IDE повністю переведені на Юнікод, також з'явилися нові можливості компілятора (узагальнення (англ. Generics) і анонімні методи), в IDE було додано менеджер ресурсів та повністю перебудовано менеджер проектів. Найновіша, 15 версія, має назву Delphi XE і входить до Embarcadero RAD Studio XE.

Delphi в основному використовується для розробки настільних застосунків та корпоративних СКБД, проте цей інструмент можна використовувати для розробки будь-якого загального програмного забезпечення. Не залишена осторонь і можливість побудови Веб-застосунків, так потрібних у сучасному інформаційному світі.

Delphi поширюється у кількох редакціях з різними можливостями і цінами: Personal (на даний час недоступний), Professional, Enterprise (раніше Client/Server) та Architect.

Borland Kylix — це еквівалент до Delphi для платформи Linux. Проте, розробка наступних версій була припинена компанією Borland. Проте 16 травня 2009 року на конференції Delphi Live 2009 було оголошено про роботу над проектом Delphi «X», що полягає на введенні кросплатформенної підтримки для розробки на Mac OS та Linux.

Turbo Pascal та Borland Pascal були дешевими 16-бітними компіляторами. За роки свого існування, вони пройшли через багато релізів, і в основному використовувалися для створення програм, що виводили інформацію у текстовому режимі. Коли використання графічного інтерфейсу користувача стало необхідним у Microsoft Windows 3.1, було представлено Delphi, розроблене на основі Borland Pascal. Delphi була першою так званою системою швидкої розробки, випущеною у 1995-ому році для 16-бітної Windows 3.1.

Delphi 2, представлена роком пізніше, підтримувала 32-бітне Windows-середовище, а версія, що використовувала мову C++, під назвою C++ Builder побачила світ іще кількома роками пізніше.

Головним архітектором Delphi на той час був Андерс Гейлсберг, який розробив Turbo Pascal. Він перейшов у Microsoft у 1996 для розробки мови C#.

У 2001-ому році була представлена версія для операційної системи Linux під назвою Kylix. Проте дуже швидко вона була розкритикована за низьку якість і велику кількість помилок. Через це, враховуючи низький рівень продажу, вона була занедбана після третьої версії.

Була зроблена спроба зробити підтримку і Linux, і Windows для багатоплатформної розробки, внаслідок чого у Delphi 6 була включена бібліотека CLX, багатоплатформна версія бібліотеки VCL. Технологія CLX теж зазнала поразки і після падіння Kylix теж була закинута.

Ще починаючи з першої версії 1.0 розробка програм для баз даних стала однією з сильних сторін Delphi. Бібліотека візуальних компонент (англ. Visual Component Library, VCL) містила велику бібліотеку компонент для доступу та маніпулювання з базами даних. Borland Database Engine була оригінальною технологією зв'язку з базами даних, і була єдиним рішенням у ранніх версіях Delphi. Навіть зараз, коли вона рідко використовується, остання версія IDE все ще поставляється з BDE, необов'язковим для встановлення.

Delphi 7, випущена у серпні 2002, стала стандартом де-факто для багатьох Delphi-розробників, і навіть зараз вона активно використовується. В Delphi 7 додано підтримку для тем Windows XP і покращено можливості для побудови Web-застосунків. Також це була остання версія Delphi, яка могла використовуватися без активації. Вона мала лише необов'язкову реєстрацію, яку можна було просто проігнорувати. Delphi 7 є найбільш оціненою IDE, створеною Borland завдяки своїй стабільності, швидкості і низькими вимогами до апаратного забезпечення. Незважаючи на це у цій версії Delphi, як і у всіх інших, була велика кількість відомих помилок, так і ніколи не виправлених Borland. Завдання виправлення цих помилок компанія залишила на спільноту Delphi (дивіться «проект відродження Delphi»).

Delphi 8, представлений у грудні 2003-ого, був лише .NET-релізом, що дозволяв розробникам компілювати вихідні коди Object Pascal у .NET CIL. Також він дуже відрізнявся від попередників зовнішнім виглядом IDE, в якому вперше застосовано багатовіконний стиль, багато в чому схожий на середовище Microsoft's Visual Studio.NET. Хоча можливість перемикання у класичне (англ. Classic Undocked) розміщення вікон все ще залишилася.

Наступна версія Delphi 2005 (Delphi 9) включала в собі можливість розробки як для платформи Microsoft .NET, так і під «рідну» Win32. Також ця версія дозволяла маніпулювати даними з баз даних ще у режимі дизайнера. Відзначалося і вдосконалене IDE, а з мовних можливостей був новий вираз for … in (аналог foreach у C#'і). Проте, середовище було знову розкритиковане за

виявлені помилки — обидві версії Delphi 8 і Delphi 2005 мали проблеми із стабільністю, які були лише частково виправлені у сервісних пакетах.

Наприкінці 2005-ого було випущено Borland Developer Studio 2006, яка включала C#, Delphi.NET, Delphi Win32 та C++ у єдиному IDE. Ця версія була набагато стабільніша за Delphi 8 чи Delphi 2005, і ще більше поліпшена з випуском сервісних пакетів та оновлень.

8-ого лютого 2006-ого року Borland оголосила про пошук покупця для її IDE і лінії продуктів для баз даних, що включало Delphi. Це рішення компанія пояснила намаганням сконцентруватися на своїй лінії продуктів ALM. Ця новина була сприйнята з неоднозначною реакцією розробників, які все ще не полишили Delphi.

Шостого вересня 2006-ого Developer Tools Group (робоча назва ще не відділеної групи розробників) компанії Borland Software Corporation випустили одномовні версії Borland Developer Studio, повернувшись до популярного імені Turbo. Набір «Turbo» продуктів включав Turbo Delphi for Win32, Turbo Delphi for .NET, Turbo C++, та Turbo C#. Кожна версія доступна в двох редакціях: Explorer — версія для вільного скачування; та Professional — дешева (US$899 для нових користувачів, US$399 для оновлення) версія, яка відкривала доступ до тисяч компонент сторонніх виробників. На відміну від ранніх Personal версій Delphi, нові Explorer-редакції могли використовуватися для комерційної розробки програмного забезпечення.

14-ого листопада 2006 Borland оголосила про відділення групи розробників у незалежну дочірню компанію CodeGear.

Delphi 2007 — перша версія випущена CodeGear 16-ого березня 2007. Win32-версія була представлена першою, пізніше була випущена .NET-версія Delphi 2007 як частина продукту CodeGear RAD Studio 2007. Нові можливості включали підтримку MS Build та вдосконалення Visual Component Library для Windows Vista. CodeGear також представила DBX4 як нову версію dbExpress.

Вперше Delphi можна було завантажити з інтернету і активувати ліцензійним ключем. Локалізовані версії Delphi 2007 були одночасно представлені на англійській, французькій, німецькій та японській мовах. RAD Studio 2007, яка включала розробку на Delphi.NET та C++, була випущена 5-ого вересня 2007-ого.

У Delphi 2009 (кодова назва Tiburón) додано багато нових можливостей, зокрема узагальнення (англ. generics), анонімні методи (для Win32 та .NET), повністю перероблено VCL та RTL для повної підтримки Юнікоду.

Borland продала CodeGear компанії Embarcadero Technologies в 2008. Embarcadero зберегла відділ CodeGear, створений Borland, для ідентифікації куплених продуктів, свої ж розробки Embarcadero вирішила розповсюджувати під іменем DatabaseGear. 25 серпня 2009 року було випущено 14-ту (13-ту версію розробники пропустили) версію — Delphi 2010. 30 серпня 2010 випущено 15ту версію — Delphi XE.

Вибір саме цієї системи пояснюється тим, що вона є простою у використанні, має звичний та інтуїтивно зрозумілий синтаксис, і разом з тим дозволяє використовувати потужні програмні компоненти: від вбудованих до загальносистемних.

2.4. Практична реалізація системи для обробки цифрового звуку.

Розглянемо практичну реалізацію програмного засобу для обробки цифрового звуку.

Програма X3M EJay реалізована у вигляді віконного додатку для операційної системи Microsoft Windows. Всього у програмі присутні три вікна: головне вікно, що містить весь основний функціонал, вікно довідкової системи, що містить дані про роботу з програмою, та інформаційне вікно про програму.

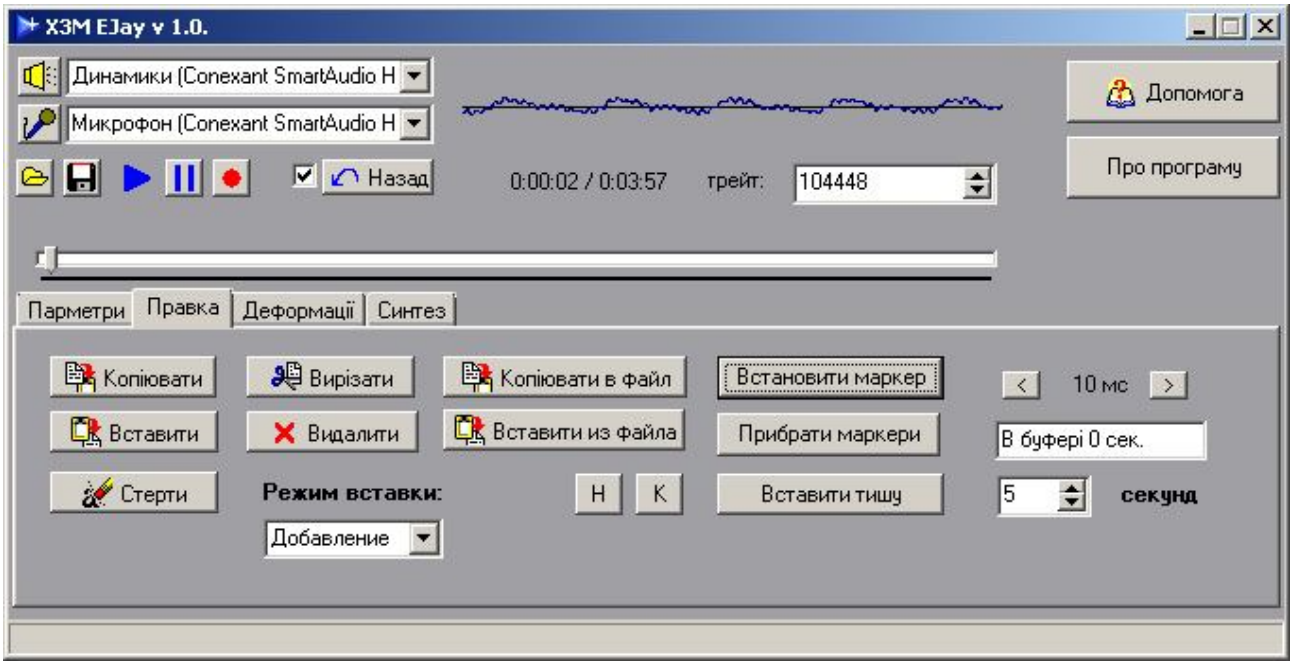

Рис. 2.4. Інтерфейс програми X3M EJay.

Користувацький інтерфейс програми розділено на декілька логічних зон. До першої зони відносяться елементи інтерфейсу, що відповідають за завантаження та керування звуковим потоком (рис. 2.5.)

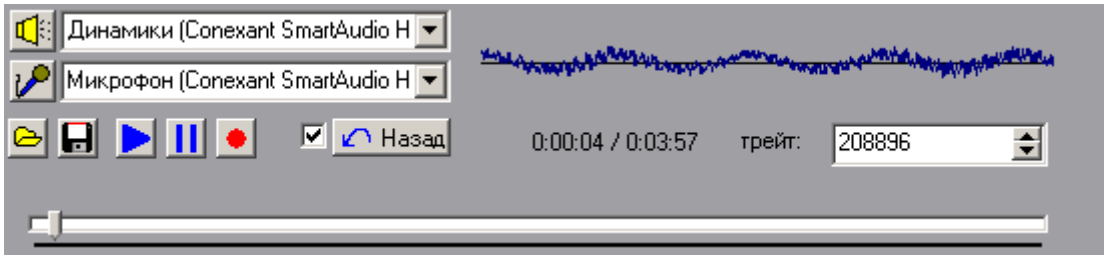

Рис. 2.5. Елементи керування вхідним та вихідним потоками.

В лівому верхньому кутку знаходяться селектори вхідного та вихідного потоків. Вхідним потоком по замовчуванню є роз'єм мікрофона звукової карти, вихідним потоком – вихідний роз'єм звукової карти.

Нижче розташовані елементи керування завантаженням звукового ряду з існуючого треку та зберігання поточних змін.

При завантаженні готового треку відбувається виклик стандартного діалогу відкриття файлу, після чого шлях до файлу передаєттся на відповідний обробник, і далі наступний код опрацьовує файл треку:

```
if (Status<>'starting')and(Status<>'waiting') then Exit;

if OpenDialog.Execute then FileName := OpenDialog.FileName else Exit;

Status := 'opening';

AudioData.Data.Clear;

if GetFileAttributes(PChar(FileName)) and FILE_ATTRIBUTE_READONLY
= FILE_ATTRIBUTE_READONLY then

  SetFileAttributes(PChar(FileName),  GetFileAttributes(PChar(FileName))
xor FILE_ATTRIBUTE_READONLY);

Ext := ExtractFileExt(FileName);

for i := 1 to Length(Ext) do Ext[i] := UpCase(Ext[i]);

if Ext = '.WAV' then

begin

  PCM := TPCMFile.Open(FileName);

  PCM.ReadAudioData(AudioData);

  PCM.Destroy;

end;

if Ext = '.MP3' then

begin
```

```
  MP3 := TMP3File.Open(FileName);
  MP3.ReadAudioData(AudioData);
  MP3.Destroy;
end;
if Ext = '.EM1' then
begin
  EM1 := TEM1File.Open(FileName);
  EM1.ReadAudioData(AudioData);
  EM1.Destroy;
end;
Str(AudioData.nChannels, S);
nChannelsText.Caption := S + ' channels';
Str(AudioData.nBitsPerSample, S);
nBitsPerSampleText.Caption := S + ' bits';
Str(AudioData.nSamplesPerSec, S);
nSamplesPerSecText.Caption := S + ' Hz';
AudioPosition := 0;
AudioData.Calculate_nBlockAlign;
SetAudioPosition;
DeleteMarkers;
Status := 'waiting';
```

Як бачимо, залежно від того, якого типу алгоритм кодування вихідного файлу, зчитування даних відбувається по різному.

Поряд розташовані елементи керування медіаплеєром – запуск відтворення, пауза, та старт запису. Також поряд з даними елементами розташовано перемикач режиму «відкат на попередню операцію». По замовчуванню даний режим увімкнено, але він вимагає потужного процесора а також додаткового місця на диску. Тому для збільшення ефективності роботи програми даний режим може бути вимкнений.

В центральній частині розташовано візуалізатор звукового ряду, який у вигляді осцилограми відтворює наближену картину частотних характеристик треку. Під візуалізатором розміщено лічильник вказівника медіаплеєра, а також індикатор бітрейта, який відображає поточний бітрейт в режимі реального часу. Функціонал цього індикатора працює як із статичним так і з динамічним бітрейтом.

В нижній частині розташовано регулятор вказівника треку, який дозволяє проводити переміщення по звукоряду із врахуванням даних лічильника.

Поряд із зоною керування треком знаходяться кнопки виклику текстової довідки та інформаційного вікна про програму.

Всі елементи для обробки звукового ряду знаходяться в окремій зоні, що розміщена в нижній частині вікна.

Всі можливі операції над даними було розподілено на чотири основні групи, і відповідні елементи керування згруповано у вигляді вкладок. Розглянемо елементи керування кожної із вкладок.

Вкладка «Параметри» дозволяє редагувати параметри, що стосуються звукоряду в цілому. Окремі функціональні елементи тут також згруповано у вигляді окремих вкладок. Так, керування частотними характеристиками відбувається через вкладку «Аудіодані» (рис.2.6.):

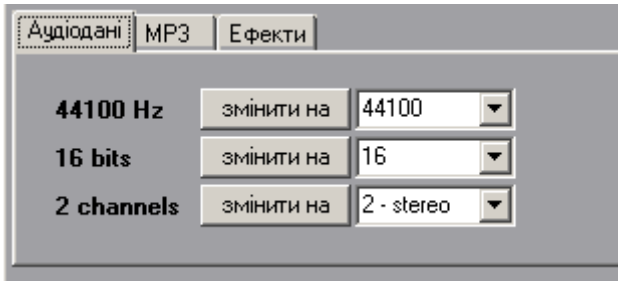

Рис. 2.6. Керування частотними характеристиками.

Як бачимо з рис. 2.6., для треку можна задати розмір частотної полоси, розрядність, та кількість каналів. По замовчуванню встановлюються загальносистемні параметри.

Для стандарту MP3 є можливість задати пропускну здатність, або бітрейт. Програма дозволяє працювати як із статичним бітрейтом, так і з динамічним, коли його розмір змінюється під час відтворення аудіо потоку. Також на цій вкладці можна задавати режим відтворення стерео-звучання.

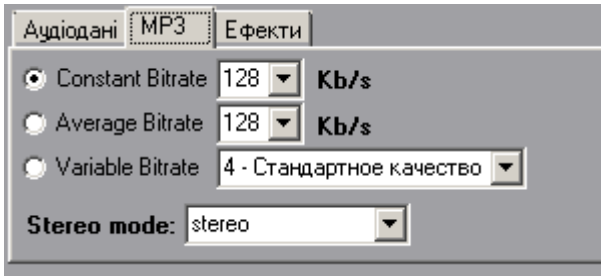

Рис. 2.7. Налаштування кодування MP3.

Для треку в цілому можна задати ефекти. Це можна зробити за допомогою вкладки «Ефекти» (рис.2.8.)

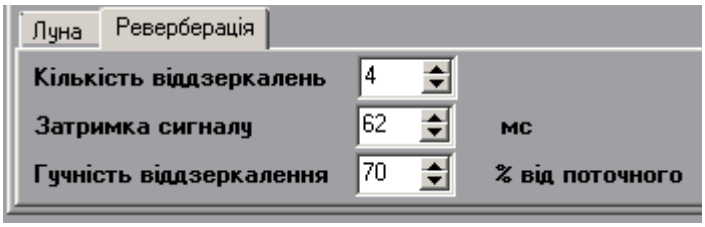

Рис. 2.8. Керування загальними ефектами треку.

Всього доступними є два типи ефектів: луна та реверберація. Налаштування кожного з них є абсолютно однаковими. Для того чи іншого ефекту необхідно задати кількість віддзеркалювань, затримку сигналу, та гучність віддзеркалювання.

Режим правки трека дозволяє проводити створення та зведення сем плів (рис. 2.9.)

Функціонал програми дозволяє виділяти необхідні фрагменти оригінального звукового ряду, і далі, в залежності від режиму вставки, заміняти, вставляти, або змішувати семпли. Крім того є можливість користуватись маркерами, та проводити експорт сем плів в окремі файли з подальшою можливість провести зворотну операцію – експорт семпла з файла в трек.

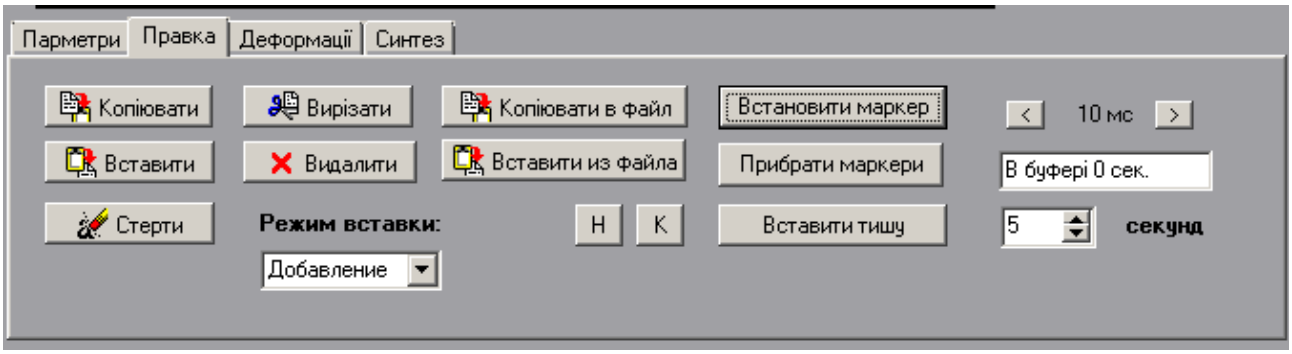

Рис. 2.9. Режим правки.

Для прикладу, розглянемо код копіювання фрагмента трека в семпл:

```
if Status<>'waiting' then Exit;
Status := 'editing';
with AudioData do
  AudioSize := Data.Size div nBlockAlign;
with Selection do
begin
  if not StartExists or not FinishExists then
  begin
    DeleteMarkers;
    Start := 0;
    Finish := AudioSize-1;
  end;
  CopyAudio(AudioData, AudioClipBoard, Start, Finish);
end;
Str(AudioClipBoard.Data.Size div AudioClipBoard.nBlockAlign div
AudioClipBoard.nSamplesPerSec, S);
Memo1.Text := В буфері:' + S + ' сек.';
Status := 'waiting';
```

Як бачимо, із загального звукового потоку виділяється фрагмент, визначений змінними *Start, Finish*, і за допомогою функції *CopyAudio* відправляється в буфер.

Основним засобом застосування фільтрів є вкладка «Деформації» (рис. 2.10):

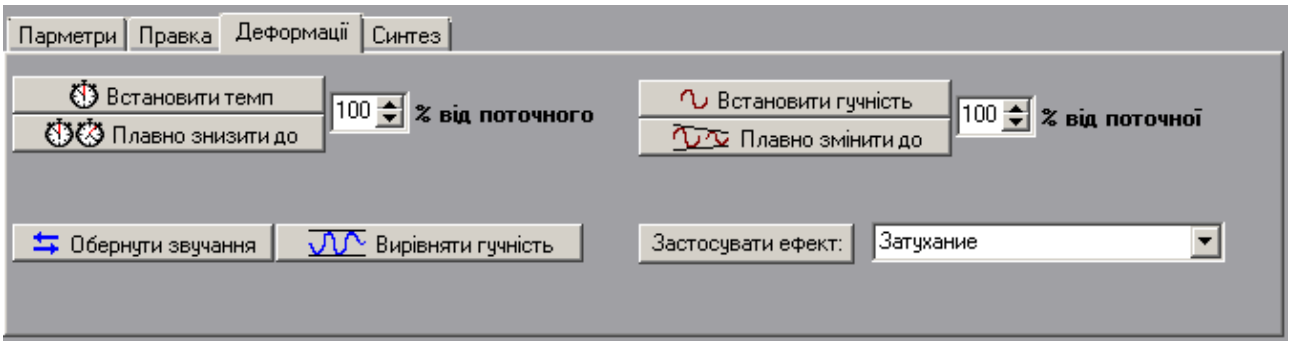

Рис. 2.10. Режим накладання фільтрів.

Накладання фільтрів дозволяє надати треку закінченості, а також створити оптимальні звукові переходи між окремими семплами. У випадку створення підкастів використання фільтрів дозволяє створити правильно оформлені початок і закінчення підкаста.

Всього доступно два основних ефекти (темп та звучання), кожен з яких може бути застосований як дискретно, так і з плавним авто-решулюванням. Доступний також режим авто-регулювання гучності, та режим інверсії звучання. Також можна застосувати один із додаткових ефектів (луна, реверберація, тощо не до трека взагалі, а до окремого семпла).

Для прикладу розглянемо код автоматичного вирівнювання гучності:

```
 AudioSize: Cardinal;
begin
 if Status<>'waiting' then Exit;
 Status := 'deformation';
 SaveUndoInfo;
 with AudioData do
  AudioSize := Data.Size div nBlockAlign;
```

```
with Selection do
begin
  if not StartExists or not FinishExists then
  begin
    DeleteMarkers;
    Start := 0;
    Finish := AudioSize-1;
  end;
  Normalize(AudioData, Start, Finish-Start+1);
end;
Status := 'waiting';
```

Як бачимо із лістинга коду, нормалізація відбувається викликом відповідної функції для звукового потоку із вказаним початком та кінцем періоду. Всі вихідні коди функцій подано у розділі додатків, в додатку 1.

Наступним елементом функціоналу є вкладка «Синтез» (рис. 2.11.):

Рис. 2.11. Режим синтезу звуку.

Режим синтезу звуку в даній програмі реалізовано у вигляді дискретного біт-генератора, який дозволяє генерувати біт заданої частоти.

Генерація біту відбувається за допомогою наступного програмного коду:

```
if Status<>'waiting' then Exit;
Status := 'deformation';
SaveUndoInfo;
with AudioData do
  AudioSize := Data.Size div nBlockAlign;
with Selection do
begin
  if not StartExists or not FinishExists then
  begin
    DeleteMarkers;
    Start := 0;
    Finish := AudioSize-1;
  end;
  AddBrainWave(AudioData, Start, Finish-Start+1, BWFreqEdit1.Value, BWFreqEdit2.Value);
end;
Status := 'waiting';
```

Біт генерується за допомогою функції *AddBrainWave* в якості аргументів якої виступають звуковий ряд, проміжок семпла, а також частотні характеристики біта.

Таким чином було розглянуто основні функціональні можливості програми X3M EJay. Програма дозволяє виконувати задачі, які були поставлені, отже можна вважати, що мета даної роботи була цілком досягнута.

Перспектива розвитку даної програми полягає в розширенні функціональних характеристик, і можливій інтеграції із більш потужними аудіо-редакторами по обміну даними.

Висновки

Дана магістерська робота присвячена технологіям обробки цифрового звуку.

В результаті виконання даної роботи було досліджено теоретичні основи цифрового звуку, та основні технології його обробки. На основі отриманих теоретичних результатів було розроблено програму для обробки цифрового звуку.

Розроблена програма дозволяє виконувати над цифровим звуком всі базові операції, потреба в яких може виникнути в пересічного користувача. Зокрема основу для обробки може становити як готовий звуковий ряд, завантажений із файла в одному із трьох форматів кодування (wav, mp3, em1). При цьому декодування і подальше кодування із стисненням при зберіганні відбуваються автоматично.

Крім того передбачено можливість накладання на звукоряд фрагментів із інших готових звукових рядів, або потокового звуку із зовнішнього джерела, під яким розуміється мікрофон, або інший аналоговий пристрій-генератор звуку.

До основних функціональних характеристик розробленої програми можна віднести наступні:

- модифікація параметрів звукового сигналу;
- правка фрагментів звукового потоку (семплінг);
- обробка звукового ряду з використанням фільтрації (міксування);
- синтез нових елементів звукового ряду (біт-генератор).

Завдяки функціоналу модифікації параметрів звукового сигналу можна змінювати особливості кодування (частотні характеристики, бітрейт, тощо), а

також накладати деякі базові ефекти: віддзеркалення та затримку звуку, луну тощо.

Функціонал правки фрагментів звукового потоку дозволяє виділяти із загальної композиції окремі елементи (семпли), і оперувати ними, модифікуючи загальну композицію.

Функціонал звукових деформацій (накладання фільтрів) дозволяє керувати рядом різних звукових харкетеристик, зокрема контролювати тепм, гучність, а також застосовувати ефекти (плавну зміну темпу і гучності, вирівнювання гучності, інверсію сигналу, тощо).

В програму також вбудований біт-генератор, який щоправда має базовий функціонал, однак дозволяє генерувати ритм-біт із заданими частотними характеристиками.

Розроблена програма орієнтована на використання в підкастах для запису та обробки звуку, а також для навчання основам обробки цифрового звуку при створенні електронної музики.

Отже, в підсумку, можна сказати, що в результаті виконання даної роботи було досягнено всіх поставлених цілей, а саме досліджено поняття цифрового звуку та технологій його обробки, і на основі отриманих теоретичних результатів розроблено програму для роботи із цифровим звуком.

Список використаної літератури

1. Чеппел Д. Создаём свою компьютерную студию звукозаписи. Триумф, 2007.
2. Червяков А. Звукозапись и обработка звука в программе Audacity.
3. Козюренко Ю.И. Звукозапись с микрофона. Altex, 2008.
4. Севашко А.В. Звукорежиссура и запись фонограмм. Профессиональное руководство. Altex, 2008.
5. Крапивенко А.В., «Технологии мультимедиа и восприятие ощущений», Москва М:. БИНОМ. Лаборатория знаний, 2009;
6. Крапивенко А.В., «Методы и средства обработки аудио- и видеоданных», Москва М:. «Вузовская книга», 2010;
7. Сергиенко А.Б. Цифровая обработка сигналов. – М.: Наука, 1999. 303 с
8. Ю.А.Ковалгин Цифровое кодирование звуковых сигналов. – М.: Питер, 2004. 245 с
9. Алдошина И.А., Вологдин Э.И. Электроакустика и звуковое вещание. Учебное пособие для вузов. – М.: Питер, 2005. 409 с
10. Р. Петелин, Ю.Петелин. Музыкальный компьютер. Секреты мастерства СПб: БХВ-Санкт-Петербург, Арлит, 2003. - 686 с.
11. Питер Кирн. Цифровой звук. ISBN 978-5-8459-1324-1
12. Borwick, John, ed., 1994: Sound Recording Practice (Oxford: Oxford University Press)
13. Ifeachor, Emmanuel C., and Jervis, Barrie W., 2002: Digital Signal Processing: A Practical Approach (Harlow, England: Pearson Education Limited)
14. Rabiner, Lawrence R., and Gold, Bernard, 1975: Theory and Application of Digital Signal Processing (New Jersey: Prentice-Hall, Inc.)
15. Watkinson, John, 1994: The Art of Digital Audio (Oxford: Focal Press)

Додатки

Додаток 1.

Програмна реалізація ефектів для обробки цифрового звуку.

```
unit AudioFormat;
interface
uses
  SysUtils, FileUtils;
type
  TAudioFile = class(TFile)
  end;
type
  TAudioData = class(TObject)
  public
    nChannels: Word;
    nSamplesPerSec: LongWord;
    nBitsPerSample: Word;
    nBlockAlign: Word;
    Data: TFile;
    constructor Create;
    destructor Destroy;
    procedure Calculate_nBlockAlign;
    procedure ReadSample(Number, Channel: LongInt; var Value: Integer);
    procedure WriteSample(Number, Channel: LongInt; Value: Integer);
    function Extremum(Number, Channel: LongInt): Boolean;
```

```
  private
    Name: String;
  end;
```

procedure CopyAudio(var AudioSource, AudioGeter: TAudioData; Start, Finish: Cardinal);

procedure DeleteAudio(var AudioData: TAudioData; Start, Finish: Cardinal);

procedure InsertAudio(var AudioSource, AudioGeter: TAudioData; Start: Cardinal);

procedure OverwriteAudio(var AudioSource, AudioGeter: TAudioData; Start: Cardinal);

procedure MixAudio(var AudioSource, AudioGeter: TAudioData; Start: Cardinal);

procedure ReverseAudio(var AudioData: TAudioData; Start, Count: Cardinal);

procedure AddBrainWave(var AudioData: TAudioData; Start, Count, Freq1, Freq2: Integer);

procedure SetSpeedOfAudio(var AudioData: TAudioData; Start, Count: Cardinal; Speed: Real);

function ChangeSpeedOfAudio(var AudioData: TAudioData; Start, Count: Cardinal; Speed: Real): Cardinal;

procedure SetnSamplesPerSec(var AudioData: TAudioData; Value: Cardinal);

procedure SetnBitsPerSample(var AudioData: TAudioData; Value: Cardinal);

procedure SetnChannels(var AudioData: TAudioData; Value: Cardinal);

procedure SetVolumeOfAudio(var AudioData: TAudioData; Start, Count: Cardinal; Volume: Real);

procedure Echo(var AudioData: TAudioData; Start, Count, Number, Delay: Cardinal; Volume: Real);

procedure Reverberation(var AudioData: TAudioData; Start, Count, Number, Delay: Cardinal; Volume: Real);

```
    procedure ChangeVolumeOfAudio(var AudioData: TAudioData; Start, Count: Cardinal; Volume: Real);

    procedure ReChangeVolumeOfAudio(var AudioData: TAudioData; Start, Count: Cardinal; Volume: Real);

    procedure Normalize(var AudioData: TAudioData; Start, Count: Cardinal);

    implementation

    constructor TAudioData.Create;

    var

      TempDir, FileName: String;

      i: Word;

    begin

      inherited Create;

      TempDir := GetEnvironmentVariable('TEMP')+'\';

      i := 0;

      FileName := TempDir + '\' + '0.TAD';

      while FileExists(FileName) do

      begin

        Inc(i);

        Str(i, FileName);

        FileName := TempDir + '\' + FileName + '.TAD';

      end;

      Name := FileName;

      Data := TFile.Create(FileName);

    end;

    procedure TAudioData.Calculate_nBlockAlign;

    begin
```

```
  nBlockAlign := nBitsPerSample div 8;
  if nBitsPerSample mod 8 <> 0 then  Inc(nBlockAlign);
  nBlockAlign := nBlockAlign*nChannels;
end;
procedure TAudioData.ReadSample(Number, Channel: LongInt; var Value: Integer);
var
  i: Byte;
  Mult, AbsValue: LongWord;
begin
  Calculate_nBlockAlign;
  Data.Position := Number*nBlockAlign + Channel*(nBlockAlign div nChannels);
  AbsValue := 0;
  Data.Read(AbsValue, nBlockAlign div nChannels);
  Mult := 1;
  for i := 1 to nBlockAlign div nChannels do
    Mult := Mult*256;
  if nBitsPerSample>8 then
    if AbsValue >= Trunc(Mult/2) then Value := AbsValue - Mult else Value := AbsValue
  else Value := AbsValue-128;
end;
procedure TAudioData.WriteSample(Number, Channel: LongInt; Value: Integer);
var
  K: Byte;
```

```
  N: Cardinal;
begin
  Calculate_nBlockAlign;
  Data.Position := Number*nBlockAlign + Channel*(nBlockAlign div nChannels);
  if Data.Position>Data.Size then
  begin
    K := 0;
    N := Data.Position + nBlockAlign div nChannels;
    Data.Position := Data.Size;
    while Data.Position<=N do Data.Write(K, 1);
    Data.Position := Number*nBlockAlign + Channel*(nBlockAlign div nChannels);
  end;
  if nBitsPerSample<=8 then Value := Value+128;
  Data.Write(Value, nBlockAlign div nChannels);
end;
function TAudioData.Extremum(Number, Channel: LongInt): Boolean;
var
  Smp1, Smp, Smp2: Integer;
begin
  if (Number = 0) or (Number + 1 = Data.Size div nBlockAlign) then
  begin
    Extremum := True;
    Exit;
  end;
```

```
    ReadSample(Number-1, Channel, Smp1);
    ReadSample(Number, Channel, Smp);
    ReadSample(Number+1, Channel, Smp2);
    if (Smp1<Smp)and(Smp>Smp2) or (Smp1>Smp)and(Smp<Smp2) then
      Extremum := True
    else
      Extremum := False;
  end;
  destructor TAudioData.Destroy;
  begin
    Data.Destroy;
    DeleteFile(Name);
    inherited Destroy;
  end;
  procedure CopyAudio(var AudioSource, AudioGeter: TAudioData; Start,
Finish: Cardinal);
  var
    i: Cardinal;
    Buf: array[0..63] of Byte;
  begin
    AudioGeter.Data.Clear;
    AudioGeter.nChannels := AudioSource.nChannels;
    AudioGeter.nSamplesPerSec := AudioSource.nSamplesPerSec;
    AudioGeter.nBitsPerSample := AudioSource.nBitsPerSample;
    AudioGeter.Calculate_nBlockAlign;
    AudioSource.Data.Position := Start*AudioSource.nBlockAlign;
```

```
    for i := 1 to Abs(Finish-Start) do
    begin
      AudioSource.Data.Read(Buf, AudioSource.nBlockAlign);
      AudioGeter.Data.Write(Buf, AudioSource.nBlockAlign);
    end;
    AudioGeter.nChannels := AudioSource.nChannels;
    AudioGeter.nSamplesPerSec := AudioSource.nSamplesPerSec;
    AudioGeter.nBitsPerSample := AudioSource.nBitsPerSample;
  end;
  procedure InsertAudio(var AudioSource, AudioGeter: TAudioData; Start:
Cardinal);
  var
    i: Cardinal;
    Buf: Byte;
  begin
    if AudioSource.Data.Size = 0 then Exit;
    AudioGeter.Data.Insert(Start*AudioGeter.nBlockAlign,
AudioSource.Data.Size);
    AudioSource.Data.Position := 0;
    AudioGeter.Data.Position := Start*AudioGeter.nBlockAlign;
    for i := 1 to AudioSource.Data.Size do
    begin
      AudioSource.Data.Read(Buf, 1);
      AudioGeter.Data.Write(Buf, 1);
    end;
  end;
```

```
procedure DeleteAudio(var AudioData: TAudioData; Start, Finish: Cardinal);
begin
  AudioData.Data.Delete(Start*AudioData.nBlockAlign, Abs(Finish-Start)*AudioData.nBlockAlign);
end;
procedure OverwriteAudio(var AudioSource, AudioGeter: TAudioData; Start: Cardinal);
var
  i: Cardinal;
  Buf: Byte;
begin
  if AudioSource.Data.Size = 0 then Exit;
  AudioSource.Data.Position := 0;
  AudioGeter.Data.Position := Start*AudioGeter.nBlockAlign;
  for i := 1 to AudioSource.Data.Size do
  begin
    AudioSource.Data.Read(Buf, 1);
    AudioGeter.Data.Write(Buf, 1);
  end;
end;
procedure MixAudio(var AudioSource, AudioGeter: TAudioData; Start: Cardinal);
var
  i, Channel, AudioSize: Cardinal;
  Value, MaxValue: Int64;
  Samp1, Samp2: Integer;
begin
```

```
    if AudioSource.Data.Size = 0 then Exit;
    AudioSize := AudioGeter.Data.Size div AudioGeter.nBlockAlign;
    MaxValue := 1; for i := 1 to AudioGeter.nBitsPerSample-1 do MaxValue := MaxValue*2;
    for i := 0 to AudioSource.Data.Size div AudioGeter.nBlockAlign - 1 do
     for Channel := 0 to AudioGeter.nChannels-1 do
     begin
      AudioSource.ReadSample(i, Channel, Samp1);
      if Start+i<AudioSize then
       AudioGeter.ReadSample(Start+i, Channel, Samp2)
      else
       Samp2 := 0;
      Value := Samp1 + Samp2;
      if (Value < -MaxValue)or(Value >= MaxValue) then
       Value := Trunc((Value)/2);
      AudioGeter.WriteSample(Start+i, Channel, Value);
     end;
  end;
  procedure ReverseAudio(var AudioData: TAudioData; Start, Count: Cardinal);
  var
   i, AbsStart, AbsFinish, AbsCount: Cardinal;
   BufferStart: Cardinal;
   Buf: Int64;
   TempAudio: TAudioData;
  begin
   TempAudio := TAudioData.Create;
```

```
AbsStart := Start*AudioData.nBlockAlign;
AbsCount := Count*AudioData.nBlockAlign;
AbsFinish := AbsStart+AbsCount;
i := AbsFinish;
repeat
  if i-AbsStart>=MaxSizeOfBuffer then
    BufferStart := i - MaxSizeOfBuffer
  else
    BufferStart := AbsStart;
  AudioData.Data.Position := BufferStart;
  AudioData.Data.Read(Buf, 1);
  while i>BufferStart do
  begin
    i := i - AudioData.nBlockAlign;
    AudioData.Data.Position := i;
    AudioData.Data.Read(Buf, AudioData.nBlockAlign);
    TempAudio.Data.Write(Buf, AudioData.nBlockAlign);
  end;
until i=AbsStart;
AudioData.Data.Position := AbsStart;
TempAudio.Data.Position := 0;
for i := 1 to Count do
begin
  TempAudio.Data.Read(Buf, AudioData.nBlockAlign);
  AudioData.Data.Write(Buf, AudioData.nBlockAlign);
end;
```

```
  TempAudio.Destroy;
end;
procedure AddBrainWave(var AudioData: TAudioData; Start, Count, Freq1, Freq2: Integer);
var
  i, MaxAmplitude: Cardinal;
  T, TL, TR: Real;
  Freq: Integer;
  SampL, SampR: Integer;
begin
  if AudioData.nChannels = 1 then Exit;
  MaxAmplitude := 1;
  for i := 1 to AudioData.nBitsPerSample-1 do
    MaxAmplitude := MaxAmplitude*2;
  for i := Start to Start+Count-1 do
  begin
    Freq := Freq1 + Round((i-Start)*(Freq2-Freq1)/Count);
    T := 2*Pi/(AudioData.nSamplesPerSec/(Freq/100));
    TL := 2*Pi/(AudioData.nSamplesPerSec/(50+50*Freq/100));
    TR := 2*Pi/(AudioData.nSamplesPerSec/(50+50*Freq/100+Freq/100));
    AudioData.ReadSample(i, 0, SampL);
    AudioData.ReadSample(i, 1, SampR);
    SampL := Trunc(0.6*SampL+0.4*MaxAmplitude*Sin(i*TL));
    SampR := Trunc(0.6*SampR+0.4*MaxAmplitude*Sin(i*TR));
    AudioData.WriteSample(i, 0, SampL);
    AudioData.WriteSample(i, 1, SampR);
```

```
  end;
end;
procedure Normalize(var AudioData: TAudioData; Start, Count: Cardinal);
var
  i, MaxAmplitude, MaxVolume: Cardinal;
  Volume: Integer;
  K: Real;
  Channel: Word;
begin
  MaxAmplitude := 1;
  for i := 1 to AudioData.nBitsPerSample-1 do
    MaxAmplitude := MaxAmplitude*2;
  for Channel := 0 to AudioData.nChannels-1 do
  begin
    MaxVolume := 0;
    for i := Start to Start+Count-1 do
    begin
      AudioData.ReadSample(i, Channel, Volume);
      if Abs(Volume) > MaxVolume then MaxVolume := Abs(Volume);
    end;
    K := MaxAmplitude/MaxVolume;
    for i := Start to Start+Count-1 do
    begin
      AudioData.ReadSample(i, Channel, Volume);
      Volume := Round(Volume*K);
      AudioData.WriteSample(i, Channel, Volume);
```

```
    end;
  end;
end;
procedure SetSpeedOfAudio(var AudioData: TAudioData; Start, Count: Cardinal; Speed: Real);
var
  i, j, k, n, NewCount: Cardinal;
  Channel: Byte;
  Smp1, Smp2: Integer;
  Interval: Real;
  TempAudio: TAudioData;
  Buf: Int64;
begin
  if (Speed = 1) or (Speed = 0) then Exit;
  TempAudio := TAudioData.Create;
  TempAudio.nChannels := AudioData.nChannels;
  TempAudio.nSamplesPerSec := AudioData.nSamplesPerSec;
  TempAudio.nBitsPerSample := AudioData.nBitsPerSample;
  TempAudio.nBlockAlign := AudioData.nBlockAlign;
  NewCount := Round(Count/Speed);
  if Speed > 1 then
  begin
    i := NewCount;
    Interval := Speed;
    AudioData.Data.Position := Start*AudioData.nBlockAlign;
    while i<>0 do
```

```
        begin
          AudioData.Data.Read(Buf, AudioData.nBlockAlign);
          TempAudio.Data.Write(Buf, AudioData.nBlockAlign);
          AudioData.Data.Position := AudioData.Data.Position -
AudioData.nBlockAlign + Trunc(Interval)*AudioData.nBlockAlign;
          Interval := Interval-Trunc(Interval)+Speed;
          Dec(i);
        end;
      end
      else
      begin
        Speed := 1/Speed;
        for Channel := 0 to AudioData.nChannels-1 do
        begin
          i := 0;
          j := 0;
          Interval := Speed;
          while i<>Count do
          begin
            AudioData.ReadSample(Start+i, Channel, Smp1);
            if i+1<>Count then
              AudioData.ReadSample(Start+i+1, Channel, Smp2)
            else
              Smp2 := Smp1;
            k := Trunc(Interval);
            for n := 0 to k-1 do
```

```
            TempAudio.WriteSample(j+n, Channel, Round(Smp1+(Smp2-
Smp1)/k*n));
            Interval := Interval-Trunc(Interval)+Speed;
            Inc(i);
            Inc(j, k);
          end;
        end;
      end;
      DeleteAudio(AudioData, Start, Start+Count-1);
      InsertAudio(TempAudio, AudioData, Start);
      TempAudio.Destroy;
    end;
    function ChangeSpeedOfAudio(var AudioData: TAudioData; Start, Count:
Cardinal; Speed: Real): Cardinal;
    var
      i, j, k, n: Cardinal;
      Channel: Byte;
      Smp1, Smp2: Integer;
      Interval, FinalSpeed: Real;
      TempAudio: TAudioData;
      Buf: Int64;
    begin
      if (Speed = 1) or (Speed = 0) then Exit;
      TempAudio := TAudioData.Create;
      TempAudio.nChannels := AudioData.nChannels;
      TempAudio.nSamplesPerSec := AudioData.nSamplesPerSec;
```

```
TempAudio.nBitsPerSample := AudioData.nBitsPerSample;
TempAudio.nBlockAlign := AudioData.nBlockAlign;
FinalSpeed := Speed;
if Speed > 1 then
begin
  Speed := 1;
  Interval := Speed;
  AudioData.Data.Position  := Start*AudioData.nBlockAlign;
  while AudioData.Data.Position div AudioData.nBlockAlign < Start+Count do
  begin
    AudioData.Data.Read(Buf, AudioData.nBlockAlign);
    TempAudio.Data.Write(Buf, AudioData.nBlockAlign);
    AudioData.Data.Position := AudioData.Data.Position -
AudioData.nBlockAlign + Trunc(Interval)*AudioData.nBlockAlign;
    Interval := Interval-Trunc(Interval)+Speed;
    Speed := Speed+Trunc(Interval)*(FinalSpeed-1)/Count;
  end;
end
else
begin
  FinalSpeed := 1/FinalSpeed;
  for Channel := 0 to AudioData.nChannels-1 do
  begin
    i := 0;
    j := 0;
    Speed := 1;
```

```
        Interval := Speed;
        while i<>Count do
        begin
          AudioData.ReadSample(Start+i, Channel, Smp1);
          if i+1<>Count then
            AudioData.ReadSample(Start+i+1, Channel, Smp2)
          else
            Smp2 := Smp1;
          k := Trunc(Interval);
          for n := 0 to k-1 do
            TempAudio.WriteSample(j+n, Channel, Round(Smp1+(Smp2-
Smp1)/k*n));
          Interval := Interval-Trunc(Interval)+Speed;
          Inc(i);
          Inc(j, k);
          Speed := Speed+(FinalSpeed-1)/Count;
        end;
      end;
    end;
    DeleteAudio(AudioData, Start, Start+Count-1);
    InsertAudio(TempAudio, AudioData, Start);
    ChangeSpeedOfAudio := TempAudio.Data.Size div TempAudio.nBlockAlign;
    TempAudio.Destroy;
  end;
  procedure SetnSamplesPerSec(var AudioData: TAudioData; Value: Cardinal);
  var
```

```
  AudioSize: Cardinal;
begin
  if AudioData.nSamplesPerSec = Value then Exit;
  with AudioData do
    AudioSize := Data.Size div nBlockAlign;
  if AudioSize <> 0 then SetSpeedOfAudio(AudioData, 0, AudioSize,
AudioData.nSamplesPerSec/Value);
  AudioData.nSamplesPerSec := Value;
end;

procedure SetnBitsPerSample(var AudioData: TAudioData; Value: Cardinal);
var
  AudioSize, Max1, Max2, i: Cardinal;
  Channel: Word;
  Smp: Integer;
  Mult: Real;
  TempAudio: TAudioData;
begin
  if AudioData.nBitsPerSample = Value then Exit;
  with AudioData do
    AudioSize := Data.Size div nBlockAlign;
  TempAudio := TAudioData.Create;
  TempAudio.nChannels := AudioData.nChannels;
  TempAudio.nSamplesPerSec := AudioData.nSamplesPerSec;
  TempAudio.nBitsPerSample := Value;
  TempAudio.Calculate_nBlockAlign;
```

```
    Max1 := 1; for i := 1 to AudioData.nBlockAlign div AudioData.nChannels
do Max1 := Max1*256;

    Max2 := 1; for i := 1 to TempAudio.nBlockAlign div TempAudio.nChannels
do Max2 := Max2*256;

    Mult := Max2/Max1;

    if AudioSize<>0 then

    begin

      for Channel := 0 to AudioData.nChannels-1 do

        for i := 0 to AudioSize-1 do

          begin

            AudioData.ReadSample(i, Channel, Smp);

            Smp := Trunc(Smp*Mult);

            TempAudio.WriteSample(i, Channel, Smp);

          end;

      AudioData.Data.Clear;

      OverwriteAudio(TempAudio, AudioData, 0);

    end;

    TempAudio.Destroy;

    AudioData.nBitsPerSample := Value;

    AudioData.Calculate_nBlockAlign;

  end;

  procedure SetnChannels(var AudioData: TAudioData; Value: Cardinal);

  var

    AudioSize: Cardinal;

    TempAudio: TAudioData;

    i: Integer;
```

```
  Channel: Cardinal;
  Smp: Integer;
begin
  if AudioData.nChannels = Value then Exit;
  with AudioData do
    AudioSize := Data.Size div nBlockAlign;
  TempAudio := TAudioData.Create;
  TempAudio.nChannels := Value;
  TempAudio.nSamplesPerSec := AudioData.nSamplesPerSec;
  TempAudio.nBitsPerSample := AudioData.nBitsPerSample;
  TempAudio.Calculate_nBlockAlign;
  for i := 0 to AudioSize-1 do
    for Channel := 0 to Value-1 do
    begin
      if Channel < AudioData.nChannels then
        AudioData.ReadSample(i, Channel, Smp)
      else
        AudioData.ReadSample(i, AudioData.nChannels-1, Smp);
      TempAudio.WriteSample(i, Channel, Smp);
    end;
  AudioData.Data.Clear;
  AudioData.nChannels := Value;
  AudioData.Calculate_nBlockAlign;
  OverWriteAudio(TempAudio, AudioData, 0);
  TempAudio.Destroy;
end;
```

```
procedure SetVolumeOfAudio(var AudioData: TAudioData; Start, Count: Cardinal; Volume: Real);
var
  MaxValue: Cardinal;
  Value: Integer;
  i: Cardinal;
  Channel: Word;
begin
  MaxValue := 1;
  for i := 1 to AudioData.nBlockAlign div AudioData.nChannels do
    MaxValue := MaxValue*256;
  MaxValue := MaxValue div 2 - 1;
  for Channel := 0 to AudioData.nChannels-1 do
    for i := Start to Start+Count-1 do
    begin
      AudioData.ReadSample(i, Channel, Value);
      //Value := Trunc(Value*Exp(Volume/20));
      Value := Trunc(Value*Volume);
      if Abs(Value)>MaxValue then
        if Value<0 then Value := -MaxValue
        else Value := MaxValue;
      AudioData.WriteSample(i, Channel, Value);
    end;
end;
procedure Echo(var AudioData: TAudioData; Start, Count, Number, Delay: Cardinal; Volume: Real);
```

```
var
  TempAudio: TAudioData;
  i, j, DelaySmp: Cardinal;
  SummSmp: Int64;
  Mult: Real;
  Smp: Integer;
  Channel: Word;
  MaxValue: Cardinal;
begin
  for i := 1 to AudioData.nBlockAlign div AudioData.nChannels do
    MaxValue := MaxValue*256;
  MaxValue := MaxValue div 2 - 1;
  TempAudio := TAudioData.Create;
  TempAudio.nChannels := AudioData.nChannels;
  TempAudio.nSamplesPerSec := AudioData.nSamplesPerSec;
  TempAudio.nBitsPerSample := AudioData.nBitsPerSample;
  TempAudio.Calculate_nBlockAlign;
  DelaySmp := Round(Delay*AudioData.nSamplesPerSec/1000);
  for Channel := 0 to AudioData.nChannels-1 do
    for i := Start to Start+Count-1 do
    begin
      AudioData.ReadSample(i, Channel, Smp);
      SummSmp := Smp;
      Mult := Volume;
      for j := 1 to Number do
      begin
```

```
          if i-Start < DelaySmp*j then
            Smp := 0
          else
            AudioData.ReadSample(i-DelaySmp*j, Channel, Smp);
          SummSmp := SummSmp + Round(Mult*Smp);
          Mult := Mult*Volume;
        end;
        Smp := Round(SummSmp/(Number+1));
        if Abs(Smp)>MaxValue then
          if Smp<0 then Smp := -MaxValue
          else Smp := MaxValue;
        TempAudio.WriteSample(i-Start, Channel, Smp);
      end;
    OverwriteAudio(TempAudio, AudioData, Start);
    TempAudio.Destroy;
    Normalize(AudioData, Start, Count);
  end;
  procedure Reverberation(var AudioData: TAudioData; Start, Count, Number,
Delay: Cardinal; Volume: Real);
  var
    TempAudio: TAudioData;
    i, j, k, DelaySmp: Cardinal;
    SummSmp: Int64;
    SmpBuf: array[0..64] of Int64;
    Mult: Real;
    Smp: Integer;
```

```
  Channel: Word;
  MaxValue: Cardinal;
begin
  for i := 1 to AudioData.nBlockAlign div AudioData.nChannels do
    MaxValue := MaxValue*256;
  MaxValue := MaxValue div 2 - 1;
  TempAudio := TAudioData.Create;
  TempAudio.nChannels := AudioData.nChannels;
  TempAudio.nSamplesPerSec := AudioData.nSamplesPerSec;
  TempAudio.nBitsPerSample := AudioData.nBitsPerSample;
  TempAudio.Calculate_nBlockAlign;
  DelaySmp := Round(Delay*AudioData.nSamplesPerSec/1000);
  for Channel := 0 to AudioData.nChannels-1 do
    for i := Start to Start+Count-1 do
    begin
      for j := Number downto 0 do
      begin
        if i-Start < DelaySmp*j then
          Smp := 0
        else
          AudioData.ReadSample(i-DelaySmp*j, Channel, Smp);
        SmpBuf[j] := Smp;
      end;
      Mult := Volume;
      for j := 1 to Number do
      begin
```

```
          for k := 1 to Number do
            SmpBuf[k-1] := SmpBuf[k-1] + Round(SmpBuf[k]*Mult);
          Mult := Mult*Volume;
        end;
        Smp := Round(SmpBuf[0]/(Number+1));
        if Abs(Smp)>MaxValue then
          if Smp<0 then Smp := -MaxValue
          else Smp := MaxValue;
        TempAudio.WriteSample(i-Start, Channel, Smp);
      end;
    OverwriteAudio(TempAudio, AudioData, Start);
    TempAudio.Destroy;
    Normalize(AudioData, Start, Count);
  end;
  procedure ChangeVolumeOfAudio(var AudioData: TAudioData; Start, Count:
Cardinal; Volume: Real);
  var
    MaxValue: Cardinal;
    Value: Integer;
    i: Cardinal;
    FinalVolume: Real;
    Channel: Word;
  begin
    MaxValue := 1;
    for i := 1 to AudioData.nBlockAlign div AudioData.nChannels do
      MaxValue := MaxValue*256;
```

```
  MaxValue := MaxValue div 2 - 1;
  FinalVolume := Volume;
  for Channel := 0 to AudioData.nChannels-1 do
  begin
   Volume := 1;
   for i := Start to Start+Count-1 do
   begin
    AudioData.ReadSample(i, Channel, Value);
    //Value := Trunc(Value*Exp(Volume/20));
    Value := Trunc(Value*Volume);
    if Abs(Value)>MaxValue then
     if Value<0 then Value := -MaxValue
     else Value := MaxValue;
    AudioData.WriteSample(i, Channel, Value);
    Volume := Volume + (FinalVolume-1)/Count;
   end;
  end;
 end;
 procedure ReChangeVolumeOfAudio(var AudioData: TAudioData; Start, Count: Cardinal; Volume: Real);
 var
  MaxValue: Cardinal;
  Value: Integer;
  i: Cardinal;
  FinalVolume: Real;
  Channel: Word;
```

```
begin
  MaxValue := 1;
  for i := 1 to AudioData.nBlockAlign div AudioData.nChannels do
    MaxValue := MaxValue*256;
  MaxValue := MaxValue div 2 - 1;
  FinalVolume := Volume;
  for Channel := 0 to AudioData.nChannels-1 do
  begin
    Volume := 0;
    for i := Start to Start+Count-1 do
    begin
      AudioData.ReadSample(i, Channel, Value);
      //Value := Trunc(Value*Exp(Volume/20));
      Value := Trunc(Value*Volume);
      if Abs(Value)>MaxValue then
        if Value<0 then Value := -MaxValue
        else Value := MaxValue;
      AudioData.WriteSample(i, Channel, Value);
      Volume := Volume + FinalVolume/Count;
    end;
  end;
end;
end.
```

Приватний вищий навчальний заклад "Міжнародний економіко-гуманітарний університет ,імені академіка Степана Дем'янчука"

Адреса: Україна, 33000, м. Рівне,

вул. академіка Степана Дем'янчука, 4

тел./факс: (0362) 23-01-86

р/р 26007301588483 в Промінвестбанку

м. Рівне, МФО 333335, код ЗКПО 24171048

The Higher Educational Establishment of private form of property "International University of Economics & Humanities named after academician Stepan Demianchuk»

Address: Ukraine, 33000, Rivne

Academician Stepana Demianchuka street, 4

Tel/fax (0362) 23-01-86

26007301588483 in Prominvestbank

Rivne, MFO 333335, kode 24171048

E-mail: mail@regi.rovno.ua

Довідка про впровадження програмного продукту

Декану факультету кібернетики

Міжнародного економіко-гуманітарного

університету імені академіка Степана Дем'янчука

проф. Янчуку Петру Степановичу

ПВНЗ "МЕГУ",

радіостудія «Університетська хвиля».

Гарматюк Валерій Миколайович

Програмний продукт X3MEJ призначений для обробки звуку, автором якого, є студент Міжнародного економіко-гуманітарного університету імені академіка Степана Дем'янчука, Вольський Артемій Вікторович, було представлено на перевірку його роботи та функціональність.

Робота даного програмного продукту була проаналізована провідними спеціалістами радіо " Університетська хвиля " у відповідному для цього середовищі. Після детального аналізу та тестів X3MEJ отримала оцінку – "відмінно". Від 21.12.2011 року за певною домовленістю із автором програми, програма використовується на радіо. Збереження всіх авторських прав радіо "Університетська хвиля" гарантує.

Радіостудія «Університетська хвиля».

Гарматюк Валерій Миколайович