НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Радіотехнічний факультет

Кафедра радіоінженерії

До захисту допущено:

Завідувач кафедри

_____ Сергій ЛІТВІНЦЕВ

«16» 06 2026 р.

Дипломна робота

на здобуття ступеня бакалавра за освітньою програмою

«Інформаційна та комунікаційна радіоінженерія»

спеціальності 172 «Електронні комунікації та радіотехніка»

на тему: «Дослідження частотно-часових перестановок аудіосигналів та їх скремблювання на базі плати EZ-Kit Lite»

Виконав (-ла):
студент (-ка) ІІІ курсу, групи РІ-п31
ХРУЦЬКИЙ Арсеній Дмитрович

Керівник:
Старший викладач кафедри радіоінженерії,
ПАВЛОВ Олег Ігорович

Рецензент:
Професор кафедри радіоінженерії, к.т.н., проф.
МОВЧАНЮК Андрій Валерійович _____

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

Київ – 2026 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Радіотехнічний факультет
Кафедра радіоінженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 172 «Електронні комунікації та радіотехніка»

Освітня програма «Інформаційна та комунікаційна радіоінженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій ЛІТВІНЦЕВ
«_20_»____04_____2026 р.

ЗАВДАННЯ
на дипломну роботу студенту
Хруцькому Арсенію Дмитровичу

1. Тема роботи «**Дослідження частотно-часових перестановок аудіосигналів та їх скремблювання на базі плати EZ-Kit Lite**»,

керівник роботи ПАВЛОВ Олег Ігорович,

затверджені наказом по університету від «28»____05___ 2026 р. №1990-с

2. Термін подання студентом роботи _____15.06.2026_____

3. Вихідні дані до роботи: типи скремблювання — часове, частотне, частотно-часове; способи скремблювання: перемішування (перестановки), віддзеркалення, перемішування з віддзеркаленням; кількість часових фрагментів — до 8; кількість частотних підсмуг — до 8; середовище стрес тестування — модель параметричного каналу з кодеком AMR-4750 системи GSM; типи сигналів — РІ, ЛЧМ РІ, реальний мовленнєвий сигнал каналу ТЧ; дослідження розбірливості скрембльованих та дескрембльованих сигналів — системи Speech-to-Text на основ ШІ; мови програмування — Matlab, ANSI C;

платформи для реалізації — ПК з ОС Windows, середовище Matlab, VisualDSP++; відлагоджувальний модуль EZ-KIT Lite ADSP-2181.

4. Зміст роботи: Місце скремблерів в системах зв'язку та Аналіз методів скремблювання; Розгляд операцій скремблювання в часовому та частотному просторах; Розробка моделі на Matlab для дослідження процесів частотно-часових перестановок в сигналах; Розробка моделі на ANS C для дослідження можливості реалізації скремблювання на платформі EZ-KIT Lite ADSP-2181; Дослідження стійкості скрембльованих/дескрембльованих сигналів до розпізнавання голосових фраз в системах Speech-to-Text; Дослідження стійкості скрембльованих/дескрембльованих сигналів до параметричних перетворень в мовленнєвих кодека типу AMR-4750 системи GSM; Розробка прикладів завдань до ЛР та методичних вказівок; Додатки з кодами розроблених програм.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): Презентація з 10-15 слайдів, що ілюструють роботу, в тому числі, стуктуру системи зв'язку та місце в ній пристрою скремблювання, структурнц та функціональну схеми опрерацій скремблювання, осцилограми та спектрограми сигналів, результати експериментів з розпізнавання скрембльованих та дескрембльованих мовленнєвих сигналів, результати стрес-тестування на GSM каналі з кодеком AMR-4750.

6. Дата видачі завдання 20.04.2026р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
	Огляд літератури	20.04 — 26.04	Виконано
	Аналіз методів скремблювання	27.04 — 03.05	Виконано
	Математичні операції при скремблюванні	04.05 — 10.05	Виконано
	Розробка моделі скремблювання на Matlab	11.05 — 17.05	Виконано
	Розробка моделі скремблювання на ANSI C	18.05 — 24.05	Виконано
	Методика тестування, дослідження	25.05 — 31.05	Виконано
	Розробка ЛР та методичних вказівок	01.06 — 07.06	Виконано
	Оформлення ПЗ та презентації	08.06 — 14.06	Виконано

Студент Арсеній ХРУЦЬКИЙ

Керівник Олег ПАВЛОВ

РЕФЕРАТ

Пояснювальна записка дипломної роботи має обсяг 67 сторінок, містить 16 рисунків, 2 таблиці, 20 найменувань у переліку посилань.

Дипломна робота присвячена дослідженню методів захисту мовних сигналів у каналах зв'язку з обмеженою пропускною здатністю шляхом застосування частотно-часових перестановок. Для вирішення проблеми вразливості класичного одномірного скремблювання та уникнення обчислювальної складності цифрових SpeechLike модемів запропоновано використання двовимірного каскадного скремблювання, керованого псевдовипадковими кодами.

Теоретичне моделювання у середовищі MATLAB підтвердило руйнацію початкової структури аудіосигналу та довело математичну оборотність перетворень. Практичну реалізацію системи в режимі реального часу виконано мовою ANSI-C на базі 16-бітного сигнального процесора з фіксованою точкою ADSP-2181 (навчальний модуль EZ-Kit Lite). Під час апаратної адаптації здійснено перехід до покадрової обробки, виконано вирівнювання пам'яті для генераторів адрес та впроваджено алгоритми компенсації динамічного діапазону. Отримані результати підтверджують високу ефективність та достатню швидкодію розробленого програмно-апаратного комплексу. Створені віртуальні макети повністю адаптовані для використання в навчальному процесі під час підготовки фахівців з радіосистем.

Ключові слова: захист інформації, скремблювання, частотно-часові перестановки, аудіосигнал, цифрова обробка сигналів, MATLAB, ADSP-2181, EZ-Kit Lite, вокодер.

ABSTRACT

The diploma thesis consists of 67 pages, 16 figures, 2 tables, and 20 references.

The work is devoted to the study of methods for protecting speech signals in communication channels with limited bandwidth through time-frequency permutations. To address the vulnerability of classical one-dimensional scrambling and the computational complexity of digital SpeechLike modems, the use of two-dimensional cascaded scrambling controlled by pseudorandom codes is proposed.

Theoretical modeling in MATLAB confirmed the destruction of the initial audio signal structure and proved the mathematical reversibility of the transformations. Practical implementation in real-time was performed in ANSI-C using a 16-bit fixed-point digital signal processor ADSP-2181 (EZ-Kit Lite module). During hardware adaptation, a transition to frame-by-frame processing, memory alignment for address generators, and dynamic range compensation algorithms were successfully implemented. The obtained results confirm the high efficiency and sufficient operating speed of the developed hardware-software complex. The created virtual prototypes are fully adapted for educational purposes in training radio systems specialists.

Keywords: information protection, scrambling, time-frequency permutations, audio signal, digital signal processing, MATLAB, ADSP-2181, EZ-Kit Lite, vocoder.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

КМ	імпульсно-кодова модуляція
ЛЧМ	лінійно-частотна модуляція
ТЧ	тональна частота
ШПФ	швидке перетворення Фур’є
2D	двовимірний (від англ. Two-dimensional)
AES	удосконалений стандарт шифрування (від англ. Advanced Encryption Standard)
AMR-NB	адаптивний багатошвидкісний вузькосмуговий кодек (від англ. Adaptive Multi-Rate Narrowband)
ASR	система автоматичного розпізнавання мови (від англ. Automatic Speech Recognition)
DMA	прямий доступ до пам'яті (від англ. Direct Memory Access)
SpeechLike	мовоподібний (про тип модемів)
WRR	відсоток успішно розпізнаних слів (від англ. Word Recognition Rate)

ВСТУП

У сучасних умовах виникають потреби захиститу інформацію, що передається голосовими повідомленнями при використанні звичайних телефонних ліній чи вузькосмугових радіоканалів. Класичне параметричне кодування мовленнєвих сигналів з подальшим криптографічним захистом цифрових даних вимагає занадто широкої смуги частот, яку не забезпечують стандартні канали аналогового зв'язку, а зменшення швидкості потоку даних обмежено суттєвим погіршенням якості існуючих кодеків. Крім того, використання мереж GSM зв’язку чи подібних, при цьому, потребує складного модемного обладнання типу SpeechLike модемів через використання на вході і виході таких систем власних параметричних мовленнєвих кодеків. Через це залишається необхідність шифрувати сигнал одразу в його початковому, аналоговому вигляді, вписуючись у наявні вимоги: сигнал має залишатися мовоподібним сигналом, — займати смугу частот каналу ТЧ та бути придатним до параметричного кодування в системах мобільного зв’язку.

Мета роботи: Проаналізувати методи підвищення рівня конфіденційності мовних каналів телекомунікаційних систем та дослідити алгоритми двовимірного частотно-часового скремблювання, а також можливість створення на їхній основі програмно-апаратного лабораторно-методичного комплексу для впровадження у навчальний процес підготовки фахівців за профільними дисциплінами.

1. МЕТОДИ ЗАХИСТУ ІНФОРМАЦІЇ ЩО МІСТИТЬСЯ В АУДІОСИГНАЛАХ

1.1. Скремблювання як початковий етап криптографічного захисту з використанням передачі трансформованих сигналів через аналогові канали зв'язку.

У цій роботі розглядаються методи захисту мовленнєвих сигналів у каналах з обмеженою пропускною здатністю. Необхідність аналогового захисту зумовлена технічними характеристиками стандартних ліній зв'язку. Наприклад, під час перетворення мовленнєвого сигналу за допомогою стандартної ІКМ генерується цифровий потік зі швидкістю 64 кбіт/с. Для передачі такого обсягу даних без втрат необхідні цифрові або широкосмугові лінії зв'язку. Проте класичні аналогові телефонні лінії або вузькосмугові радіоканали мають обмеження смуги частот і не здатні пропустити цей високошвидкісний цифровий потік без застосування складного модемного обладнання. Звідси виникла технічна потреба захищати сигнал безпосередньо в його початковій, безперервній формі, не виходячи за межі доступного спектра та зберігаючи його мовоподібні властивості, до яких оптимізовані параметричні мовленнєві кодеки мобільних мереж.

У загальній системі зв’язку розроблюваний скремблер слід розглядати як термінальний пристрій перетворення мовного сигналу, який встановлюється між джерелом мови та каналом зв’язку (рис. 1.1). На передавальній стороні він виконує оборотне перетворення форми аудіосигналу за секретним правилом, а на приймальній стороні дескремблер, використовуючи синхронізований ключ, відновлює початкову часово-частотну структуру мовлення.

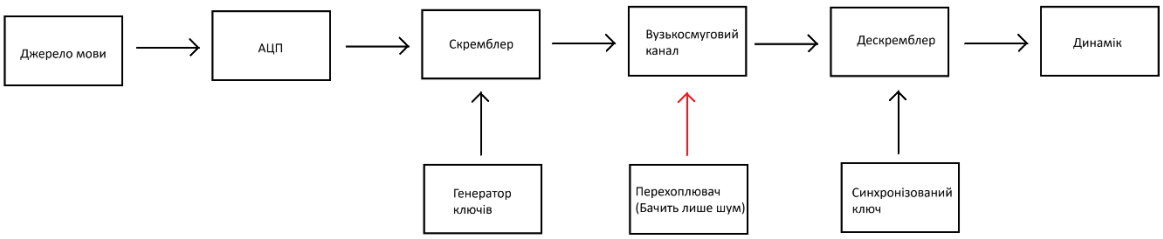

Рисунок 1.1 – Місце двовимірного скремблера у системі захищеного мовного зв'язку.

На відміну від цифрового шифрування бітового потоку, такий підхід не вимагає передавання цифрової криптограми через канал, що оптимізований виключно для людської мови. Головна вимога до такого вузла полягає у тому, щоб після перетворення сигнал залишався у базовій смузі тонального каналу (0,3–3,4 кГц) та не потребував розширення загальної смуги пропускання.

Під час побудови захищених вузлів зв'язку важливо розрізняти математичне шифрування та процеси скремблювання [1]. Класичне шифрування оперує виключно цифровими бітовими масивами. Натомість скремблювання змінює структуру безперервного аналогового сигналу. Суть цього процесу полягає у зміні внутрішньої структури безперервного мовного сигналу таким чином, щоб зробити його нерозбірливим для перехоплення сторонніми слухачами чи стандартними приймачами. Ключовою умовою тут є те, що після всіх маніпуляцій спектр перетвореного сигналу має чітко вкладатися в межі вузької смуги пропускання наявного аналогового каналу [2].

Прихованість такого сигналу досягається через цілеспрямовану зміну амплітудних, частотних або часових характеристик мови, які згодом відновлюються на приймальній стороні.

Базовим методом є частотна інверсія спектра. Під час такої обробки вхідний сигнал перемножується з гармонійним коливанням опорної частоти від локального генератора (гетеродина). Внаслідок цього формуються верхня та нижня бічні смуги частот. За допомогою фільтра низьких частот виділяється лише нижня бічна смуга, яка фактично є дзеркальним відображенням

вихідного спектра. Це означає, що низькочастотні компоненти голосу перетворюються на високочастотні, а високі, навпаки, переходять у низькочастотну область.

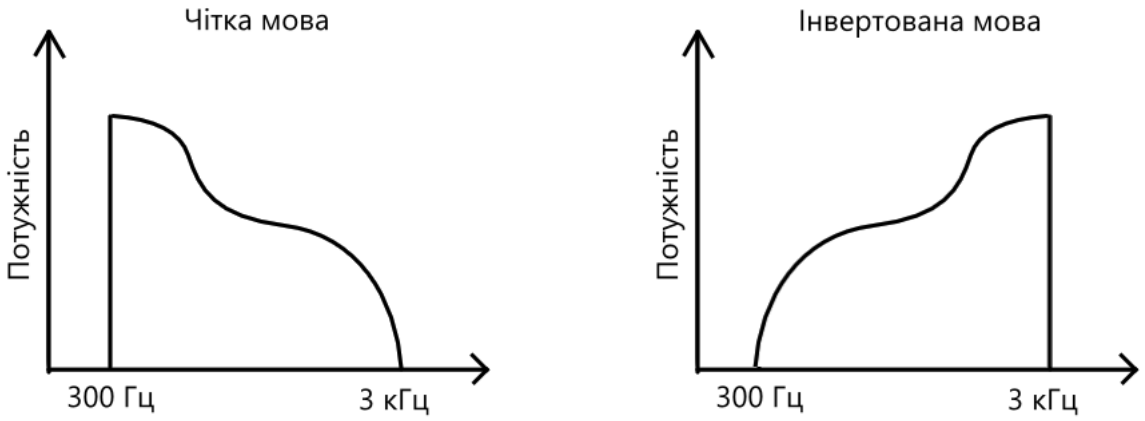

Рисунок 1.2 – Принцип частотної інверсії спектра

Більш складним підходом є смугове скремблювання. У цьому випадку загальна смуга частот мовного каналу розбивається на кілька вужчих підсмуг за допомогою набору смугових фільтрів. Далі ці виділені частотні діапазони перемішуються між собою згідно із заданим алгоритмом: наприклад, друга підсмуга може транслюватися на місці п'ятої, а перша — на місці третьої. Додатково спектр окремих підсмуг може бути інвертований. Після виконання цих перестановок усі компоненти лінійно додаються, утворюючи новий сумарний сигнал, частотна структура якого суттєво відрізняється від оригіналу.

На відміну від частотних маніпуляцій, часові перестановки фрагментів оперують сигналом уздовж осі часу. Безперервний аналоговий мовний сигнал спочатку сегментується на короткі часові відрізки [1] [3]. Вони накопичуються в буферній пам'яті апаратного забезпечення, після чого зчитуються та передаються в лінію зв'язку у зміненому, псевдовипадковому порядку. На приймальній стороні здійснюється зворотна буферизація та відновлення правильної послідовності фрагментів.

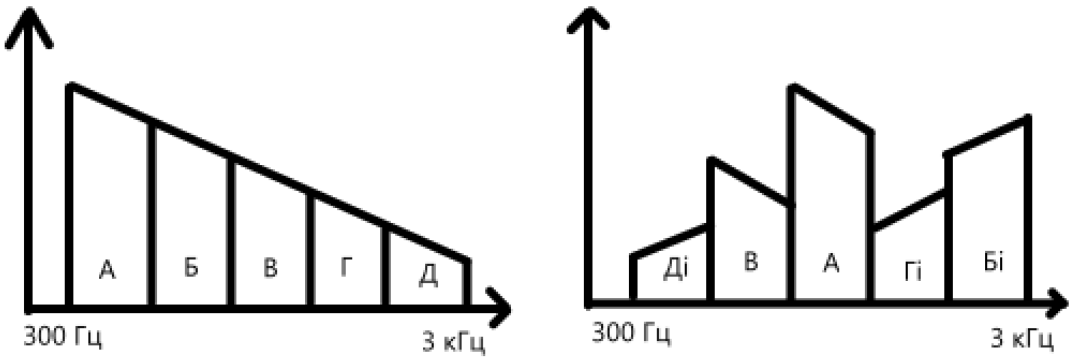

Рисунок 1.3 – Принцип часового скремблювання.

Аналіз характеристик розглянутих методів свідчить про їхню головну технічну перевагу — збереження вихідної смуги пропускання. Після виконання частотних або часових перетворень спектр скрембльованого сигналу гарантовано залишається у межах стандартного телефонного або вузькосмугового радіоканалу, який зазвичай становить 0,3–3,4 кГц. Це означає, що впровадження такого захисту дозволяє повноцінно використовувати наявну аналогову інфраструктуру зв'язку без необхідності її дорогої апаратної модернізації чи розширення пропускної здатності комунікаційних ліній.

Незважаючи на перевагу у вигляді збереження вихідної вузької смуги пропускання, одномірне скремблювання має низьку криптографічну стійкість за сучасними стандартами. Завдяки стрімкому розвитку обчислювальної техніки та доступності алгоритмів ШПФ, зловмисник здатний у режимі реального часу побудувати спектрограму перехопленого сигналу [4]. Це дозволяє візуально або за допомогою автоматизованих засобів аналізу виявити структурні закономірності та реконструювати правило перемішування частот чи послідовність часових сегментів з мінімальними обчислювальними затратами. Отже, використання одномірних перетворень є недостатнім для захисту інформації від цілеспрямованого криптоаналізу.

В спеціалізованій літературі широко обговорюються шляхи вдосконалення методів перетворення аналогових сигналів для усунення

недоліків одномірного скремблювання. Значна кількість досліджень присвячена застосуванню математичного апарату хаотичних відображень для керування перестановками. Іншим напрямком є використання алгоритмів сліпого розділення джерел, зокрема методів аналізу головних та незалежних компонент [5]. Ці підходи ускладнюють завдання для криптоаналітиків і дозволяють зберігати вихідну смугу пропускання. Водночас їх впровадження вимагає складних математичних обчислень, що не завжди підходить для портативного обладнання. На тлі цих методів перехід до класичного двовимірного частотно-часового скремблювання виступає раціональним рішенням. Воно поєднує достатній рівень приховування інформації та можливість стабільної роботи на базі поширених сигнальних мікропроцесорів.

1.2 Криптографічний захист у вокодерних системах з використанням передачі шифрованих даних через цифрові канали зв'язку

Обмежена криптографічна стійкість аналогових методів скремблювання спонукала до пошуку альтернативних інженерних рішень. Передача нестисненого оцифрованого голосу вимагає широкосмугових каналів, але оскільки такий потік неможливо передати через стандартний вузькосмуговий канал, логічним кроком стало застосування вокодерних систем. На відміну від ІКМ, вокодери не передають саму форму хвилі, а здійснюють параметризацію мови. Це забезпечується шляхом математичного моделювання голосового тракту людини: на етапі аналізу генерується та передається лише набір специфічних коефіцієнтів, серед яких параметри лінійного передбачення, частота основного тону та тип сигналу збудження [2][6].

Рисунок 1.4 – Зовнішній вигляд вокодерного модема

Вокодери здійснюють параметричне стиснення мови до низькошвидкісних потоків (2,4–9,6 кбіт/с). Таке глибоке кодування є невід'ємним етапом підготовки мовного сигналу до передачі в умовах обмеження смуги пропускання. Сформований потік даних здатен пройти через стандартні телефонні лінії або радіоканали з використанням відповідного модемного обладнання.

Переведення мови у формат компактного цифрового масиву відкриває нові шляхи для забезпечення конфіденційності переговорів. Отримана послідовність бітів уже не має прямого фізичного зв'язку з початковим безперервним коливанням, що дозволяє відмовитися від параметричного перемішування аналогових сигналів. Натомість організацією захищеного каналу стає застосування математично стійких криптографічних алгоритмів для цифрового шифрування згенерованих вокодерних параметрів [7].

У результаті зловмисник, перехопивши передані дані в каналі зв'язку, отримує масив, який статистично нагадує білий шум. Без знання секретного ключа обчислити згенеровану псевдовипадкову послідовність та відновити інформаційні параметри вокодера вкрай важко. Завдяки цьому мовний сигнал виявляється захищеним надійним математичним апаратом, стійким до спектральних чи кореляційних методів аналізу, які зазвичай застосовуються для злому аналогового скремблювання.

Попри високу надійність математичного алгоритму, пряме шифрування вокодерних даних висуває технічні вимоги до характеристик каналу зв'язку.

Головне обмеження полягає у необхідності підтримання безперервної та точної тактової і циклової синхронізації між обладнанням передавача та приймача. Оскільки цифровий потік вокодера є глибоко стисненим, кожен переданий біт несе значне інформаційне навантаження. За наявності перешкод у лінії втрата або інверсія навіть одного двійкового символу викликає ефект поширення помилки під час роботи криптографічного алгоритму. Це призводить до повного руйнування внутрішньої структури мовного кадру, через що розшифрування наступного блоку інформації стає неможливим.

На практиці передача зашифрованого цифрового потоку через реальні канали з високим рівнем завад створює суттєву проблему. В аналогових системах скремблювання погіршення якості лінії супроводжується плавним зниженням розбірливості голосу. Натомість цифрові системи за наявності помилок реагують різким збоєм: під час надходження пошкодженого зашифрованого масиву синтезатор вокодера отримує некоректні параметри збудження та резонансів. У результаті приймальна апаратура або повністю блокує вихідний аудіосигнал, або генерує цифрові шумові артефакти.

1.3 Комбіновані методи захисту з використанням мовоподібних модемів

Окрім високої чутливості до завад і ризику втрати синхронізації, передача зашифрованих цифрових даних стикається з ще однією суттєвою проблемою — несумісністю криптографічних масивів зі стандартними алгоритмами стиснення мови. Сучасні телекомунікаційні мережі, зокрема стільникові стандарти зв'язку або вузли комерційної IP-телефонії, використовують власні транзитні вокодери для мінімізації трафіку. Конфлікт виникає тоді, коли через такий стандартний голосовий канал, оптимізований під людську мову, намагаються передати потік зашифрованих бітів за допомогою класичних методів модуляції [18].

Спроба пропустити такий сигнал через вузькосмуговий канал призводить до того, що алгоритми стиснення мережі незворотно руйнують

передані дані [7]. Внаслідок втрати інформаційної цілісності відновлення початкової бітової послідовності на приймальній стороні стає неможливим.

Вирішенням проблеми несумісності криптографічних масивів зі стандартними алгоритмами стиснення мови стала розробка технології мовоподібних модемів. Концепція цього методу полягає у тому, що перед відправленням у транзитну телекомунікаційну мережу зашифрований цифровий потік піддається специфічній попередній модуляції. Генерується спеціальний аналоговий сигнал, який за своїми акустичними характеристиками імітує звучання людської мови, ефективно маскуючи та приховуючи всередині цифрову інформацію.

Процес маскування реалізується шляхом генерації параметрів, притаманних природному голосовому тракту людини. Під час перетворення двійкових даних модем штучно формує спектральні максимуми (форманти), імітує наявність частоти основного тону та відтворює характерні для мови огинаючі амплітуди. Завдяки цьому транзитний вокодер комерційної мережі фіксує на вході сигнал, який повністю відповідає його очікуваній математичній моделі. Алгоритми стиснення коректно розпізнають закладені акустичні ознаки, виконують стандартну параметризацію та передають мовоподібний сигнал через вузькосмуговий канал зв'язку з мінімальними спотвореннями прихованого цифрового навантаження [8][9].

На приймальній стороні здійснюється зворотний процес обробки. Отриманий із каналу зв'язку аналоговий мовоподібний сигнал надходить на демодулятор модема, де відбувається розпізнавання штучних формант і витягнення первинного зашифрованого бітового потоку. Лише після демодуляції ці цифрові дані передаються на криптографічний модуль для остаточного математичного розшифрування і подальшого синтезу оригінального голосу. Забезпечення такого багатоступеневого ланцюга перетворень вимагає узгодженої роботи різних алгоритмів цифрової обробки.

Загальна архітектура комбінованого захисту, яка включає послідовні етапи обробки (шифрування, SpeechLike модуляцію, проходження через

транзитний вокодер мережі, демодуляцію та фінальне розшифрування), показує успішне вирішення проблеми сумісності з комерційними каналами зв'язку.

Проте багатоступенева архітектура перетворень супроводжується багаторазовою буферизацією даних, що генерує значні часові затримки, які є критичними для голосового спілкування в режимі реального часу. Алгоритмічна імітація акустичних ознак людської мови вимагає використання потужних апаратних обчислювальних ресурсів. Це суттєво підвищує загальну вартість комунікаційного обладнання, його енергоспоживання та габарити, що робить гібридні модеми неоптимальним рішенням для портативних вузлів зв'язку.

З огляду на ці технічні обмеження, високу вартість та апаратну складність повністю цифрових або гібридних систем, актуальним і практичним завданням залишається вдосконалення методів аналогового скремблювання. Теоретичний аналіз наведених методів показує, що перехід від вразливих одномірних перетворень до комбінованих двовимірних методів частотно-часових перестановок сегментів аналогового сигналу дозволяє ефективно обійти ключові недоліки попередніх поколінь захисту. Такий підхід забезпечує достатній рівень стійкості до сучасних методів спектрального комп'ютерного аналізу, зберігаючи при цьому вихідну вузьку смугу пропускання та вимагаючи значно меншої обчислювальної складності порівняно зі SpeechLike модемами [4].

З метою систематизації розглянутих підходів у таблиці 1.1 наведено порівняння сучасних інженерних методів захисту мовного сигналу в телекомунікаційних системах за їхніми ключовими характеристиками.

Таблиця 1.1 – Порівняння методів захисту мовного сигналу з інженерної точки зору

Метод	**Смуга каналу**	**Затримка**	**Обчислювальна складність**	**Стійкість до криптоаналізу**
Частотна інверсія	0,3–3,4 кГц	Мала	Мала	Низька
Смугові перестановки	0,3–3,4 кГц	Мала/Середня	Середня	Середня
Часові перестановки	Не розширює смугу	Кадрова	Мала/Середня	Середня
2D (Час + Частота)	Не розширює смугу	Кадрова	Середня	Висока
SpeechLike модем	GSM Voice	Значна	Висока	Дуже висока

Аналіз таблиці показує, що жоден з методів не є абсолютно універсальним. Цифрове шифрування забезпечує найвищу математичну криптостійкість, проте їх реалізація є алгоритмічно складною, а затримка кодування часто є критичною для дуплексного розмовного зв’язку. Одновимірні аналогові скремблери простіші та добре узгоджуються з вузькосмуговими каналами, але мають низьку стійкість до спектрального аналізу. З цієї причини в роботі обґрунтовано компромісний перехід до двовимірного частотно-часового скремблювання.

Висновки до Розділу 1

Було проведено комплексний аналіз методів захисту мовних сигналів у каналах зв’язку з обмеженою пропускною здатністю. Встановлено, що класичне одномірне аналогове скремблювання ефективно зберігає вихідну смугу пропускання, проте є об'єктивно вразливим до сучасних методів спектрального криптоаналізу. Використання цифрового шифрування у вокодерних системах забезпечує високу математичну криптостійкість, але висуває критичні вимоги до синхронізації та якості каналу.

Застосування SpeechLike модемів вирішує інженерну проблему несумісності цифрових криптограм із транзитними вокодерами комерційних мереж, однак вносить суттєві часові затримки та вимагає значних обчислювальних ресурсів апаратури.

На основі цього аналізу зроблено висновок, що оптимальним компромісним рішенням є перехід від вразливих одномірних перетворень до комбінованих методів двовимірних частотно-часових перестановок. Такий підхід дозволяє суттєво підвищити рівень захищеності сигналу від перехоплення, зберігаючи вузьку смугу пропускання та вимагаючи значно меншої обчислювальної складності порівняно зі SpeechLike модемами.

2. ДОСЛІДЖЕННЯ ЧАСТОТНО-ЧАСОВИХ ПЕРЕСТАНОВОК АУДІОСИГНАЛІВ

2.1 Розгляд підходів до вибору фрагментів сигналу та їх часових перестановок

Математична модель прямого часового скремблювання аудіосигналу передбачає представлення вхідного дискретизованого сигналу у вигляді одновимірного масиву (вектора-стовпця) відліків:

$$x[n], \quad n = 0,1, \dots, N-1,$$

де N — загальна кількість дискретних точок сигналу.

Довжина сигналу N не завжди є кратною заданій кількості часових фрагментів обробки M, тому виконується процедура доповнення нульовими відліками Довжина одного фрагмента L визначається як:

$$L = \left\lceil \frac{N}{M} \right\rceil,$$

а необхідна кількість нульових відліків для вирівнювання розмірності становить $N_{pad} = (L \cdot M) - N$. Модифікований сигнал $\tilde{x}[n]$ у результаті матиме фіксовану довжину $N' = L \cdot M$.

Процес сегментації та формування двовимірної структури даних (матриці кадрів) описується як відображення одновимірного масиву в матрицю X розмірністю $L \times M$, де кожен m-й стовпець відповідає окремому часовому інтервалу [1]:

$$X = [x_0, x_1, \dots, x_{M-1}],$$

елементи якої визначаються через індекси вихідного сигналу як $X_{n,m} = \tilde{x}[n + m \cdot L]$, де $n = 0, 1, \dots, L - 1$ є локальним індексом всередині кадру, а $m = 0, 1, \dots, M - 1$ — номером часового фрагмента.

Операція часового перемішування кадрів полягає в заміні початкового порядку стовпців матриці X на новий порядок, що задається псевдовипадковим вектором-ключем $P = [p_0, p_1, \dots, p_{M-1}]$ [4]. Результатом цього етапу є проміжна матриця Y:

$$Y = [x_{p_0}, x_{p_1}, \dots, x_{p_{M-1}}].$$

Для розширення ключового простору та ліквідації залишкової розбірливості мови застосовується внутрішня поблокова інверсія (реверс часу) [3]. Ця операція керується бінарним вектором інверсії $R = [r_0, r_1, \dots, r_{M-1}]$, де $r_m \in \{0,1\}$. Якщо $r_m = 1$, то відліки всередині m-го стовпця переставляються у зворотному порядку. Остаточний вигляд елементів матриці скремблюваного сигналу Z задається виразом:

$$Z_{n,m} = Y_{L-1-n,m}, \text{якщо } r_m = 1,$$
$$Z_{n,m} = Y_{n,m}, \text{якщо } r_m = 0,$$

Формування вихідного одновимірного скремблюваного сигналу s[k] здійснюється шляхом послідовної векторизації (розгортання) матриці Z по стовпцях:

$$s[n + m \cdot L] = Z_{n,m}.$$

Процес дескремблювання (відновлення) на приймальній стороні є симетричним. Завдяки властивості самозворотності операції реверсу часу, зняття інверсії виконується повторним застосуванням того самого бінарного вектора R:

$$Y_{n,m} = Z_{L-1-n,m}, \text{якщо } r_m = 1,$$
$$Y_{n,m} = Z_{n,m}, \text{якщо } r_m = 0,$$

Після цього відновлення первинного порядку кадрів здійснюється за допомогою інверсного ключа перестановок P^{-1}, який задовольняє умову $P^{-1}\big(P(m)\big) = m$:

$$X = \left[y_{p_0^{-1}}, y_{p_1^{-1}}, \dots, y_{p_{M-1}^{-1}}\right].$$

Отриманий в результаті розгортання матриці X сигнал повністю ідентичний вихідному масиву $\tilde{x}[n]$, що теоретично гарантує нульову втрату інформації в ідеальному лінійному каналі зв'язку.

Програмна реалізація доповнення масиву нульовими відліками у середовищі MATLAB має вигляд:

```
if isrow(audio_in)
```

```
    audio_in = audio_in'; % Переводимо в вектор-стовбець
end

total_samples = length(audio_in);
samples_per_chunk = ceil(total_samples / num_chunks);
pad_length = (samples_per_chunk * num_chunks) - total_samples;

% Доповнюємо нулями якщо сигнал не ділиться націло
if pad_length > 0
    audio_in = [audio_in; zeros(pad_length, 1)];
end
```

Далі одновимірний масив трансформується у двовимірну матрицю chunk_matrix, де кожен стовпець відповідає окремому часовому кадру. Перестановка цих фрагментів реалізується звичайним матричним індексуванням згідно з вектором-ключем perm_vector:

```
chunk_matrix = reshape(audio_in, samples_per_chunk, num_chunks);
scrambled_matrix = chunk_matrix(:, perm_vector);
```

Наступним кроком застосовується внутрішня часова інверсія. Якщо відповідний елемент керуючого вектора reflect_vector дорівнює одиниці, стовпець матриці перевертається за допомогою вбудованої функції flipud. Після виконання всіх маніпуляцій сформована матриця розгортається назад в одновимірний масив scrambled_audio:

```
% Інверсія усередині вже переставлених фрагментів
for i = 1:num_chunks
    if reflect_vector(i) == 1
        scrambled_matrix(:, i) = flipud(scrambled_matrix(:, i));
    end
end
scrambled_audio = scrambled_matrix(:);
```

Завершальним етапом роботи функції time_scramble.m є переведення оброблених дискретних даних у стандартний одновимірний формат.

Отриманий кінцевий вектор-стовпець scrambled_audio є готовим скрембльованим аудіосигналом, придатним для подальшого цифрового збереження або передачі у канал зв'язку.

Для візуального контролю перетворень згенеровано тестовий лінійно-частотно-модульований (ЛЧМ) сигнал із граничною частотою 3000 Гц. Його лінійна структура дозволяє відстежити розриви, зсуви та перестановки як у часовій, так і в частотній областях.

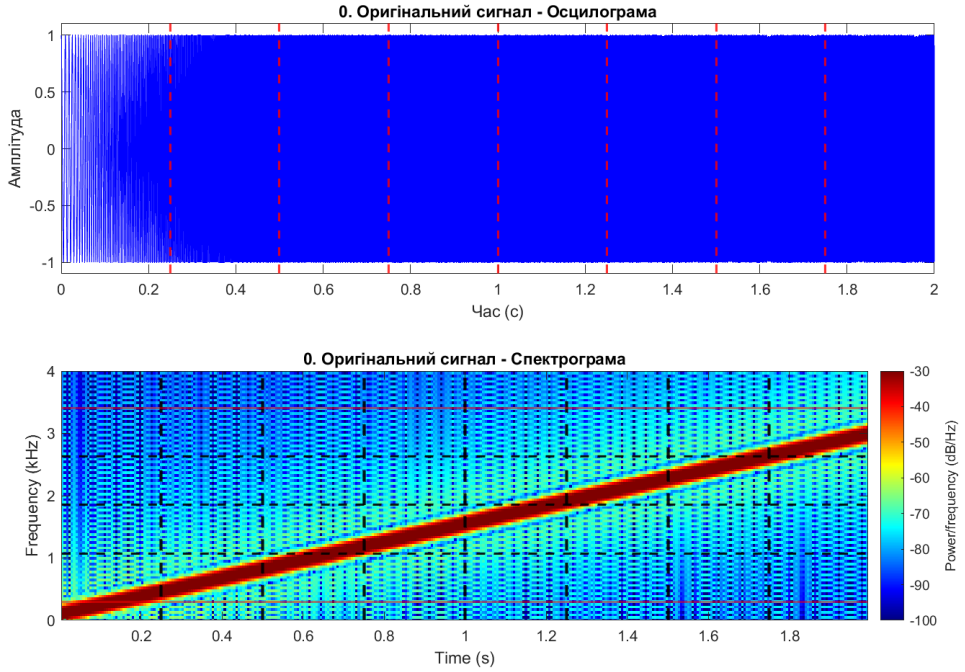

Рисунок 2.1 – Осцилограма та спектрограма оригінального тестового ЛЧМ-сигналу

Ще більш інформативною є побудована спектрограма скрембльованого сигналу. На графіку видно, як безперервна діагональна лінія оригінального ЛЧМ-сигналу розбилася на окремі чіткі прямокутні блоки. Ці спектральні блоки хаотично перемішані вздовж осі часу відповідно до застосованого ключа. Окремо відзначається, що деякі з цих блоків змінили свій нахил на зворотний, демонструючи локальне спадання частоти замість її зростання. Цей ефект є прямим візуальним підтвердженням коректної роботи алгоритму часової інверсії.

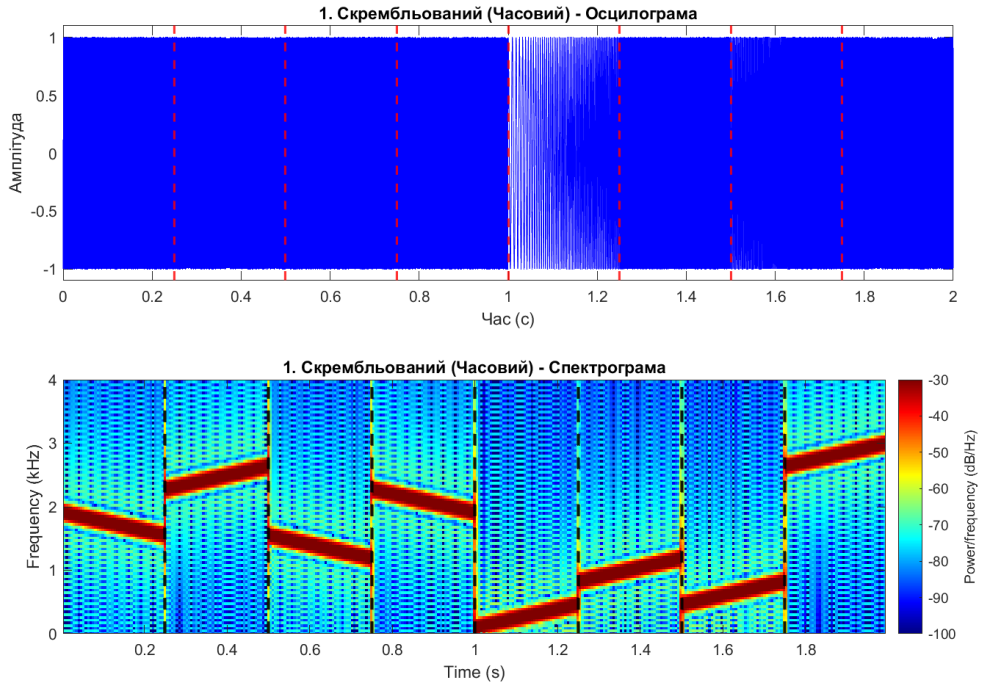

Рисунок 2.2 – Осцилограма та спектрограма сигналу після часового скремблювання

Процес декодування виконується за допомогою функції time_descramble.m. Хоча кожна окрема операція є симетричною та оборотною сама по собі, через їхнє послідовне застосування весь процес дескрембліювання також розгортається у зворотному порядку: спочатку знімається внутрішня часова інверсія, потім виконується зворотна перестановка.

Для контролю якості відновленого сигналу було повторно створено спектрограму:

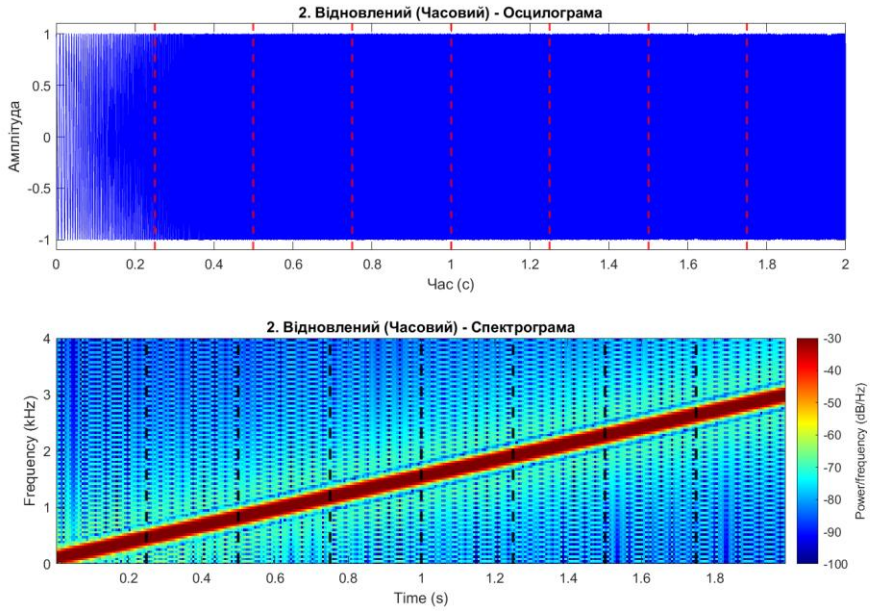

Рисунок 2.3 – Осцилограма та спектрограма відновленого сигналу після часового дескремблювання

Практичне моделювання підтверджує, що сформований вихідний вектор повністю збігається з оригінальним аудіосигналом, забезпечуючи нульову втрату інформації. Таким чином, проведений аналіз підтверджує ефективність часових маніпуляцій для маскування інформації, проте для досягнення більшого рівня захищеності доцільно розглянути аналогічний підхід, але вже стосовно спектральних компонентів сигналу.

2.2 Розгляд підходів до вибору фрагментів спектру сигналу та їх частотних перестановок

Математична модель частотного скремблювання передбачає перехід від часового представлення сигналу $\tilde{x}[n]$ до його частотно-часового спектрального опису. Для цього застосовується віконне короткочасне перетворення Фур'є, яке в дискретній формі аналітично описується виразом [1][10]:

$$X(m,k) = \sum_{n=0}^{N_w-1} \tilde{x}[n + m \cdot H] \cdot w[n] e^{-j\frac{2\pi}{N_w}kn},$$

де m — часовий індекс аналізованого вікна (кадру); k — дискретний частотний індекс біна ($k = 0,1, \dots, N_w - 1$); N_w — розмір вікна обчислення ШПФ (у розробленій моделі $N_w = 1024$ відліки) ; H — крок зсуву вікна вздовж осі часу (крок децимації) [13] [19].

Для забезпечення стабільної амплітудної характеристики під час зворотного відновлення сигналу довжина перекриття кадрів встановлюється рівною 50% від розміру вікна (H = 512 відліків). Це гарантує виконання умови постійного додавання з перекриттям (COLA):

$$\sum_m w[n - m \cdot H] = \text{const},$$

що усуває виникнення амплітудної модуляції чи пульсацій гучності на стиках суміжних вікон. Як вагове вікно аналізу w[n] застосовано симетричне вікно Хеммінга, що описується функцією:

$$w[n] = 0{,}54 - 0{,}46\cos\left(\frac{2\pi n}{N_w - 1}\right), \quad 0 \le n \le N_w - 1.$$

З метою імітації умов передачі інформації стандартним телефонним каналом зв'язку в алгоритмі задаються обмеження спектра — від $f_{min} = 300$ до $f_{max} = 3400$. Переклад фізичних частот у граничні індекси бінів ШПФ (k_{min} k_{max} здійснюється через співвідношення частоти дискретизації f_s та розміру спектрального вікна:

$$k_{min} = \max\left(2, \left\lfloor \frac{f_{min} \cdot N_w}{f_s} \right\rfloor + 1\right),$$

$$k_{max} = \min\left(\frac{N_w}{2} - 1, \left\lfloor \frac{f_{max} \cdot N_w}{f_s} \right\rfloor + 1\right).$$

Компоненти спектра $X(m, k)$, які лежать поза розрахованим робочим діапазоном $k \notin [k_{min}, k_{max}]$ занулюються для фільтрації позасмугових шумів. Загальна кількість корисних бінів $N_{\text{usable}} = k_{max} - k_{min} + 1$ ділиться на задану кількість частотних підсмуг B. Кількість спектральних відліків на одну підсмугу N_{band} обчислюється з округленням вниз для запобігання взаємному накладанню сусідніх компонентів:

$$N_{\text{band}} = \left\lfloor \frac{N_{\text{usable}}}{B} \right\rfloor.$$

Пряме частотне скремблювання полягає в перенесенні виділених спектральних блоків (підсмуг) на нові частотні позиції згідно з псевдовипадковим вектором-ключем $P_f = [p_0, p_1, \dots, p_{B-1}]$, а також у виконанні спектральної інверсії під керуванням бінарного вектора $R_f = [r_0, r_1, \dots, r_{B-1}]$, де $r_b \in \{0,1\}$ [4]. Математичне правило формування скрембльованої спектральної матриці $X_{\text{scr}}(m, k')$ для b-ї підсмуги, що переноситься на цільову позицію $b' = p_b$, визначається як:

$$X_{\text{scr}}(m, k_{min} + b' \cdot N_{\text{band}} + j) = \text{X}(\text{m}, \text{k}_{\text{min}} + \text{b} \cdot \text{N}_{\text{band}} + \text{N}_{\text{band}} - 1 - \text{j})$$

$$, \text{якщо } r_b = 1,$$

$$X_{\text{scr}}(m, k_{min} + b' \cdot N_{\text{band}} + j) = \text{X}(\text{m}, \text{k}_{\text{min}} + \text{b} \cdot \text{N}_{\text{band}} + \text{j})$$

$$, \text{якщо } r_b = 0,$$

де $j = 0,1, \dots, N_{\text{band}} - 1$ — локальний частотний індекс всередині підсмуги.

Таке матричне перенесення спектральних бінів є еквівалентним застосуванню фільтрів із прямокутною характеристикою. Внаслідок цього на межах суміжних переставлених підсмуг виникають різкі фазові розриви, що руйнує безперервність фази оригінального мовного сигналу. Це викликає ефект спектрального просочування, який у часовій області трансформується в ефект дзвону [1].

Повернення скрембльованих даних із частотної у часову область для формування вихідного аудіосигналу $s_{\text{scr}}[n]$ здійснюється за допомогою зворотного короткочасного перетворення Фур'є з обов'язковим виділенням дійсної частини отриманого масиву:

$$s_{\text{scr}}[n] = \text{Re}\left\{ \frac{1}{N_w} \sum_{m} \sum_{k=0}^{N_w-1} X_{\text{scr}}(m, k) \cdot e^{j\frac{2\pi}{N_w} k(n - m \cdot H)} \right\}.$$

Програмна реалізація прямого частотного скремблювання виконується за допомогою функції stft. Наступний фрагмент коду демонструє розрахунок спектра та занулення позасмугових компонентів:

```
window_size = 1024;
overlap = 512;
orig_length = length(audio_in);

[S, f, t] = stft(audio_in, fs, 'Window', hamming(window_size), ...
                 'OverlapLength', overlap, 'FFTLength', window_size, ...
                 'FrequencyRange', 'onesided');

S_scrambled = zeros(size(S));
N_bins = size(S, 1);
```

Наступним кроком виділена корисна смуга ділиться на задану кількість підсмуг num_bands. Для чіткого стикування спектральних блоків без їхнього взаємного накладання кількість бінів на одну підсмугу обчислюється з округленням вниз за допомогою функції floor.

```
bin_min = max(2, round(f_min * window_size / fs) + 1);
bin_max = min(N_bins - 1, round(f_max * window_size / fs) + 1);
usable_bins = bin_max - bin_min + 1;
bins_per_band = floor(usable_bins / num_bands);
bin_max = bin_min + bins_per_band * num_bands - 1;
```

Зміна структури спектра реалізується в циклі: кожен виділений частотний блок копіюється на нову позицію згідно з вектором-ключем perm_vector, а за наявності одиничного маркера у векторі reflect_vector — додатково інвертується вздовж осі частот за допомогою функції flipud.

```
for i = 1:num_bands
    % Визначаємо індекси вихідної смуги
    source_start = bin_min + (i - 1) * bins_per_band;
    source_end = source_start + bins_per_band - 1;
```

```
        band_block = S(source_start:source_end, :);

        % Спектральна інверсія
        if reflect_vector(i) == 1
            band_block = flipud(band_block);
        end

        % Перестановка в новую позицию
        target_idx = perm_vector(i);
        target_start = bin_min + (target_idx - 1) * bins_per_band;
        target_end = target_start + bins_per_band - 1;

        S_scrambled(target_start:target_end, :) = band_block;
    end
```

Розглядаючи поведінку корисного сигналу всередині робочої смуги, можна спостерігати деформацію структури сигналу. На початковій спектрограмі оригінальний chirp-сигнал виглядає як безперервна діагональна лінія, що відображає лінійне зростання частоти з часом. Натомість після частотного скремблювання ця суцільна лінія розривається на окремі горизонтальні сегменти, розташовані в межах виділених підсмуг. Зазначені частотні сегменти зміщені вздовж осі ординат, що є прямим результатом роботи вектора перестановки підсмуг. Крім того, на спектрограмі помітно, що певні фрагменти мають зворотний нахил: у цих підсмугах частота не зростає, а спадає, що візуально підтверджує спрацювання алгоритму спектральної інверсії.

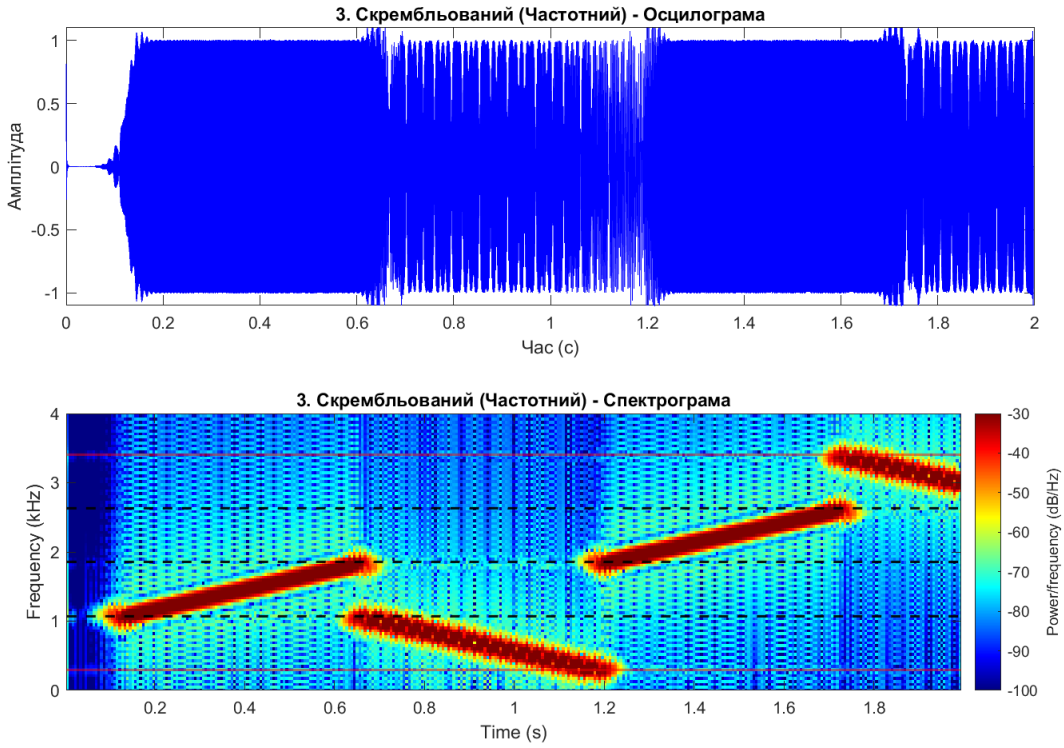

Рисунок 2.4 – Осцилограма та спектрограма сигналу після частотного скремблювання

Помітні перешкоди та залишкові шуми в інвертованих сегментах зумовлені виникненням різких фазових розривів на межах стикування спектральних блоків. Оскільки програмна інверсія та перестановка масивів фактично діють як прямокутний фільтр, безперервність фази оригінального мовного сигналу під час таких стрибкоподібних змін руйнується.

Для відновлення початкового спектра на приймальній стороні алгоритм передбачає виконання процедури дескремблювання. Процедура дескремблювання забезпечує зворотне відновлення порядку спектральних компонентів.

```
for current_idx = 1:num_bands
    original_idx = inv_perm_vector(current_idx);

    % Беремо з поточної позиції (яку ми приймаємо з каналу)
    curr_start = bin_min + (current_idx - 1) * bins_per_band;
    curr_end = curr_start + bins_per_band - 1;
```

```
            % Кладемо в оригінальну позицію (де вона має бути)
            orig_start = bin_min + (original_idx - 1) * bins_per_band;
            orig_end = orig_start + bins_per_band - 1;

            S_restored_order(orig_start:orig_end, :) = S(curr_start:curr_end, :);
        end

        S_descrambled = S_restored_order;

        % Зняття частотної інверсії (повторний переворот)
        for i = 1:num_bands
            if reflect_vector(i) == 1
                idx_start = bin_min + (i - 1) * bins_per_band;
                idx_end = idx_start + bins_per_band - 1;
                S_descrambled(idx_start:idx_end, :) =
flipud(S_descrambled(idx_start:idx_end, :));
            end
        end
```

Завершальним етапом алгоритму дескремблювання є застосування зворотного короткочасного перетворення Фур'є для повернення відліків із частотної в часову область. Після цього виконується виділення виключно дійсної частини відновленого сигналу для усунення можливих уявних артефактів машинних обчислень. Далі алгоритм здійснює динамічну фіксацію довжини масиву відліків: залежно від результату зворотного перетворення, масив або доповнюється нулями, або обрізається до оригінального розміру вхідного аудіосигналу.

```
        % Зворотне перетворення
        scrambled_audio = istft(S_scrambled, fs, 'Window', hamming(window_size),
...
                                'OverlapLength', overlap, 'FFTLength',
window_size, ...
                                'FrequencyRange', 'onesided');

        scrambled_audio = real(scrambled_audio);
```

```
            % Динамічна фіксація довжини
            if length(scrambled_audio) < orig_length
                scrambled_audio = [scrambled_audio; zeros(orig_length -
length(scrambled_audio), 1)];
            elseif length(scrambled_audio) > orig_length
                scrambled_audio = scrambled_audio(1:orig_length);
            end
```

З метою верифікації відновленого сигналу була створена спектрограма:

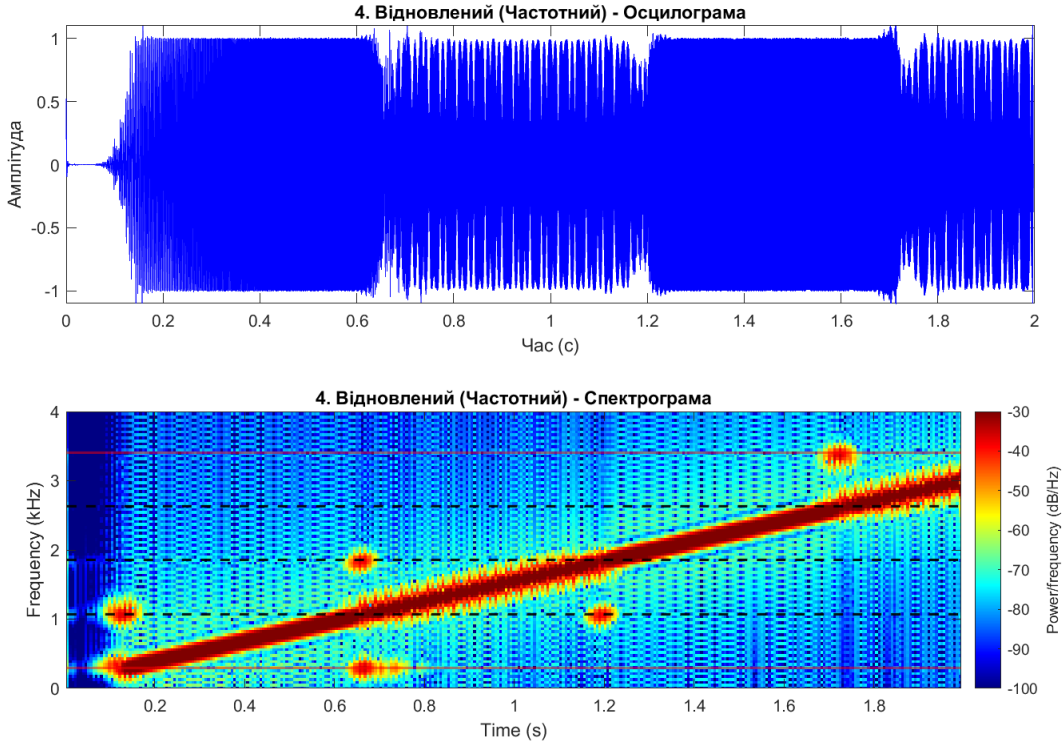

Рисунок 2.5 – Осцилограма та спектрограма відновленого сигналу після частотного дескремблювання

Візуальний аналіз спектрограми відновленого сигналу підтверджує, що макроструктура вихідної інформації відновлюється повністю. Водночас на графіку чітко фіксуються залишкові шуми у вигляді розмитих горизонтальних смуг на стиках частотних підсмуг. Унаслідок цього відновлений голос набуває специфічного металевого або роботизованого відтінку. Проте необхідно зазначити, що для систем аналогового скремблювання подібні акустичні

спотворення є технічною нормою, яка суттєво не перешкоджає загальній розбірливості мови.

Висновки до Розділу 2

Було реалізовано та досліджено математичні моделі алгоритмів скремблювання аудіосигналів у програмному середовищі MATLAB. Розроблено модульну архітектуру скриптів для виконання як одномірних, так і двовимірних каскадних перестановок з використанням псевдовипадкових ключів та бінарних векторів інверсії фрагментів.

Візуальний аналіз побудованих спектрограм тестових ЛЧМ-сигналів підтвердив, що двовимірне частотно-часове скремблювання руйнує структуру повідомлення. Процес дескремблювання довів повну оборотність алгоритмів: макроструктура сигналу відновлюється, а неминучі залишкові спотворення не є критичними і не перешкоджають загальній розбірливості мови. Створений комплекс алгоритмів має практичну цінність для подальшого навчання і використання як віртуального лабораторного макета у навчальному процесі.

3. ВИКОРИСТАННЯ ЧАСТОТНО-ЧАСОВИХ ПЕРЕСТАНОВОК СИГНАЛІВ З МЕТОЮ ЗАБЕЗПЕЧЕННЯ ЗАХИСТУ ІНФОРМАЦІЇ

3.1 Керування процесом скремблювання за допомогою унікального коду

Будь-який алгоритм скремблювання не має практичної криптографічної цінності без надійного механізму керування параметрами перемішування. Перехід від статичних, наперед визначених перетворень до динамічно керованих процесів забезпечується введенням коду, який виконує роль секретного ключа системи. Без знання цього ключа сторонній спостерігач не зможе відновити інформацію, навіть маючи повний доступ до математичного опису алгоритмів перетворення.

Програмну генерацію елементів секретного ключа реалізовано у середовищі Matlab у межах головного керуючого скрипта main.m. Відповідно до архітектури розробленої двовимірної системи, загальний ключ формується з двох типів масивів даних: векторів перестановок та векторів інверсій. Генерація масивів виконується незалежно як для часової, так і для частотної областей обробки, що суттєво розширює загальний простір можливих комбінацій. Генерація векторів перестановок (time_perm та freq_perm) спирається на використання вбудованої функції randperm, яка формує послідовність псевдовипадкових цілих чисел без повторень. Розмірність цих векторів визначається кількістю часових фрагментів num_chunks та кількістю частотних підсмуг num_bands. Водночас для формування векторів інверсій (time_reflect та freq_reflect) застосовано генератор бінарних значень randi([0, 1]), який створює одновимірний масив нулів та одиниць. У цій структурі логічна одиниця вказує алгоритму на необхідність застосування операції дзеркального відображення до конкретного часового сегмента або спектральної підсмуги, тоді як нуль залишає початковий порядок відліків незмінним.

Динамічне керування процесом скремблювання забезпечується генерацією унікальних кодових послідовностей (ключів). Стійкість розробленої системи до атак методом прямого перебору (brute-force) визначається загальним об'ємом простору ключів K.

Для каскадної двовимірної архітектури загальний ключ складається з чотирьох незалежних векторів: вектора часових перестановок P_t, вектора часових інверсій R_t, вектора частотних перестановок P_f та вектора частотних інверсій R_f.

Кількість можливих комбінацій для перестановок M часових кадрів та B частотних підсмуг визначається факторіалами M! та B! відповідно. Кількість можливих станів для бінарних векторів інверсії становить 2^M та 2^B. Загальний простір ключів системи розраховується як добуток цих множин:

$$K = M! \cdot 2^M \cdot B! \cdot 2^B$$

Навіть при мінімальних значеннях розбиття (наприклад, M = 8 та B = 8), розмірність ключового простору становить $K \approx 1{,}6 \cdot 10^{14}$ комбінацій, що забезпечує достатній рівень обчислювальної складності для унеможливлення швидкого дескремблювання без знання авторизованого коду [3] [14].

У рамках програмного моделювання в середовищі MATLAB масиви перемішування та бінарні вектори інверсії формуються за допомогою стандартних математичних функцій randperm та randi [15]. Таке рішення є виправданим на етапі прототипування, оскільки дозволяє перевірити правильність роботи концепції двовимірного скремблювання та оцінити точність зворотного відновлення аудіосигналу. Проте під час проектування реальних комерційних комплексів безпеки базові генератори псевдовипадкових чисел не застосовуються через ризик їхнього передбачення. На практиці для обчислення робочих векторів перестановки та інверсії впроваджуються криптографічно стійкі генератори псевдовипадкових послідовностей. Зазвичай вони реалізуються на базі потокових шифрів або з використанням нелінійних хаотичних відображень [9]. Застосування таких

методів гарантує математично доведену непередбачуваність коду та робить систему стійкою до спрямованого криптоаналізу [11].

```
rng(67);
time_perm = randperm(num_chunks);
time_reflect = randi([0, 1], 1, num_chunks);
freq_perm = randperm(num_bands);
freq_reflect = randi([0, 1], 1, num_bands);

time_inv_perm = get_inverse_key(time_perm);
freq_inv_perm = get_inverse_key(freq_perm);
```

Формування такого комбінованого набору масивів дозволяє гнучко налаштовувати параметри прямого двовимірного приховування даних на передавальній стороні телекомунікаційного каналу. Оскільки згенеровані псевдовипадкові послідовності є унікальними для кожного сеансу зв'язку, успішна синхронізація та зворотне перетворення сигналу вимагають точного відтворення цих операцій.

Для легітимного приймання та дескремблювання сигналу необхідно згенерувати зворотний (інверсний) ключ. Алгоритм обчислення вектора індексів для зворотного перемішування реалізовано у допоміжній функції get_inverse_key.m. Зворотний ключ inv_perm_vector обчислюється за допомогою вбудованої функції sort, яка повертає масив початкових індексів елементів до їхнього впорядкування. Самі відсортовані значення під час цього обчислення відкидаються. Натомість масив оригінальних індексів елементів до моменту їх упорядкування зберігається. Саме цей масив індексів і стає шуканим зворотним ключем, здатним повернути часові або частотні блоки на їхні вихідні місця.

```
[~, inv_perm_vector] = sort(perm_vector);
```

Окремої уваги потребують вектори інверсії reflect_vector. Для відновлення початкового стану дзеркально відображених фрагментів спектра або часу немає потреби створювати окремий зворотний ключ чи виконувати додаткові матричні розрахунки.

Операція інверсії масиву відліків, яка програмно реалізується через функцію flipud, є симетричною. Це означає, що її повторне застосування до одного й того самого фрагмента даних автоматично скасовує попередні зміни та повертає його у вихідний (прямий) стан. Таким чином, на приймальній стороні для зняття інверсії без змін використовується той самий бінарний масив, що й під час прямого скремблювання.

3.2 Програмна реалізація каскадного двовимірного скремблювання у середовищі MATLAB

Архітектура двовимірного (2D) скремблера побудована за каскадним принципом, де для досягнення максимальної стійкості до криптоаналізу одномірні перетворення застосовуються послідовно одне за одним. Програмна реалізація цієї логіки кодування зосереджена у головному керуючому скрипті main.m. Вихідний аудіосигнал послідовно обробляється спочатку в часовій області (функція time_scramble), а потім — у частотній (freq_scramble), використовуючи згенеровані раніше масиви ключів. На виході цього другого каскаду формується фінальний захищений сигнал, структура якого є повністю деформованою як у часовому, так і в спектральному просторі.

```
scrambled_2d_step1 = time_scramble(original_audio, num_chunks, time_perm, time_reflect);
scrambled_2d = freq_scramble(scrambled_2d_step1, fs, num_bands, freq_perm, freq_reflect);
```

Таке поєднання забезпечує експоненціальне розширення ключового простору, що робить спроби зламу системи методом прямого перебору (brute force) технічно ускладненим [10] [4]. Водночас цифрова обробка не змінює

загальний частотний діапазон сигналу, повністю зберігаючи його сумісність із стандартними вузькосмуговими каналами зв'язку.

Згідно з базовими правилами криптографії та цифрової обробки сигналів, процес розшифрування багатовимірних систем має відбуватися у зворотному порядку відносно етапів кодування, що відповідає класичному алгоритмічному принципу LIFO (Last In, First Out) [4]. Оскільки під час прямого двовимірного перетворення останнім етапом були спектральні маніпуляції, алгоритм дескремблювання, реалізований у головному керуючому скрипті main.m, починається саме з відновлення частотної області.

```
      descrambled_2d_step1 = freq_descramble(scrambled_2d, fs, num_bands,
freq_inv_perm, freq_reflect);
      descrambled_2d = time_descramble(descrambled_2d_step1, num_chunks,
time_inv_perm, time_reflect);
```

Після виконання обох етапів дескремблювання формується фінальний одномірний масив аудіоданих, придатний для подальшого відтворення чи аналізу. Макроструктура відновленого масиву відповідає оригінальному вихідному сигналу. Неминучі відхилення зумовлені лише спектральними артефактами обробки, які виникають внаслідок роботи прямокутних фільтрів та матричних зсувів.

Для візуального контролю роботи всього розробленого двовимірного каскаду здійснюється генерація та аналіз відповідних спектрограм.

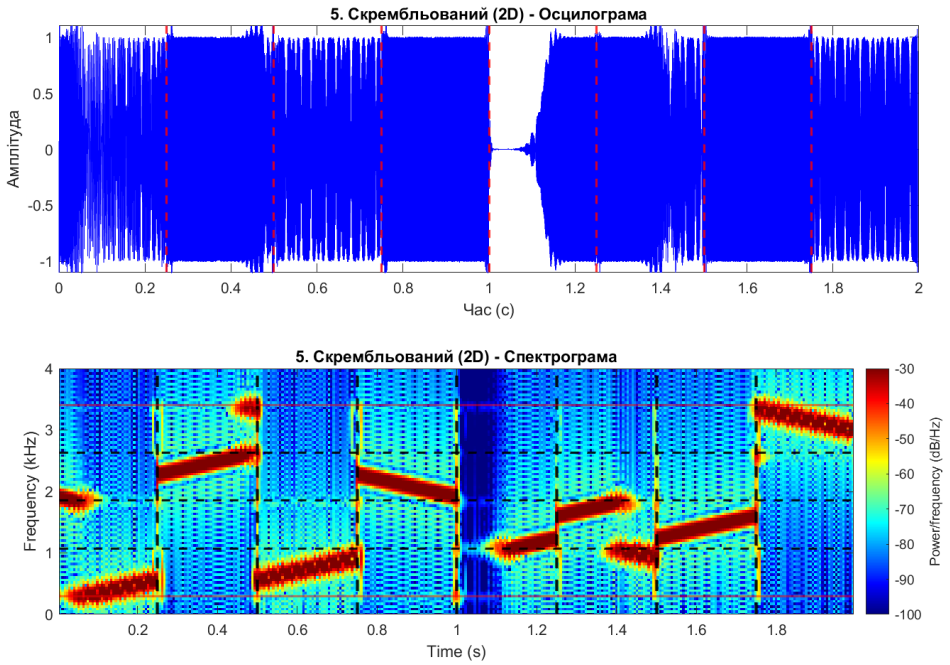

Рисунок 3.1 – Осцилограма та спектрограма сигналу після двовимірного каскадного скремблювання

Візуальний аналіз спектрограми двовимірного скрембльованого сигналу підтверджує руйнацію початкової структури сигналу. На відміну від одновимірних методів, фрагменти тестового chirp-сигналу хаотично розкидані одночасно і вздовж осі часу, і вздовж осі частот. Додатково значна частина блоків виявляється інвертованою в обох площинах. Візуально це нагадує двовимірну мозаїку, що наближається за своїми статистичними властивостями до двовимірного білого шуму, чим підтверджується високий рівень приховування вихідних даних [9].

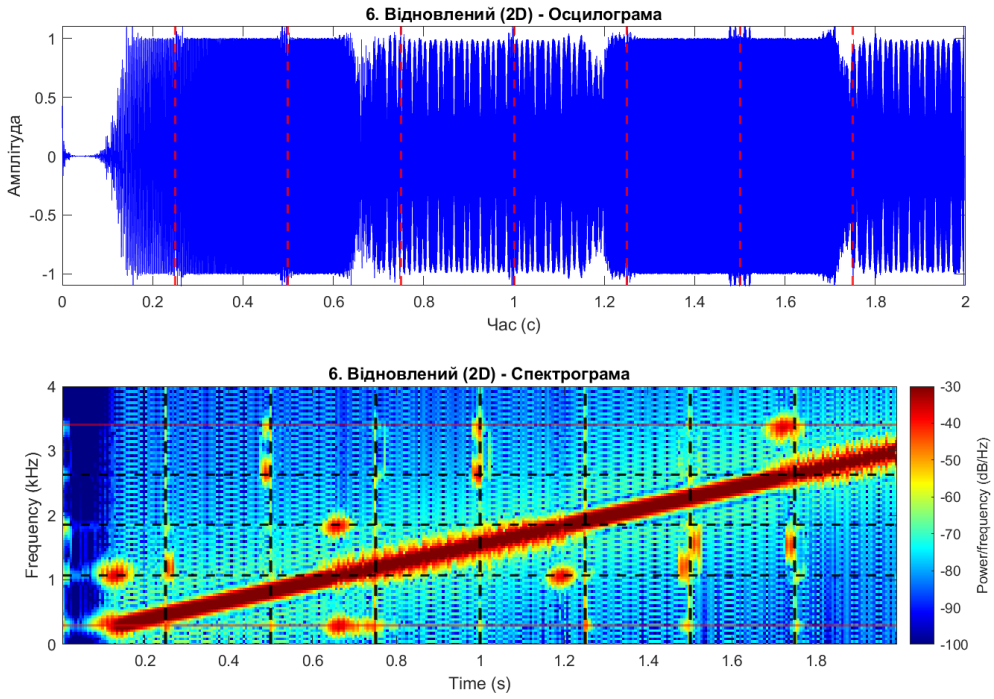

Рисунок 3.2 – Осцилограма та спектрограма відновленого сигналу після двовимірного дескремблювання

Під час дослідження відновленого сигналу проводиться аналіз його спектрограми та осцилограми. Отримані графіки демонструють, що макроструктура сигналу відновлена. Проте під час детального розгляду фіксується накладання сумарних артефактів обробки. Хоча ці залишкові явища чітко реєструються на графіках і можуть надавати звучанню специфічного тембрального забарвлення, вони виступають інженерною нормою для подібних систем скремблювання. Зазначені спотворення не руйнують загальну цілісність переданого повідомлення і не перешкоджають розбірливості мови.

3.3. Використання програмної моделі частотно-часових перестановок у навчальному процесі

Адаптація розробленої математичної моделі скремблювання для навчального процесу реалізована у вигляді віртуального лабораторного макета [11]. Програмний комплекс побудований за модульним принципом, де обчислювальні алгоритми відокремлені від блоків візуалізації та системи роботи з файлами. Такий поділ дозволяє здійснювати ізольоване дослідження

кожного етапу перетворення. Змінюючи конфігурацію виклику функцій, студенти оцінюють деградацію структурної цілісності сигналу окремо в часовій або спектральній площині, після чого аналізують сумарний ефект від каскадного двовимірного накладання.

Для забезпечення індивідуалізації завдань та можливості їх верифікації викладачем, ініціалізація генератора псевдовипадкових чисел у лабораторному макеті прив'язується до номера варіанта. Використання функції ініціалізації (rng) гарантує відтворюваність згенерованих векторів перестановок та масивів бінарних інверсій під час кожного сеансу моделювання. Зміна параметрів конфігурації, зокрема кількості часових фрагментів (num_chunks) та частотних підсмуг (num_bands), впливає на трансформацію простору ключів та рівень залишкової розбірливості повідомлення.

Лабораторний комплекс передбачає можливість вибору типу початкового тестового сигналу. Окрім реальних мовних сигналів у форматі .wav, застосовуються синтетичні сигнали: ЛЧМ сигнал (чірп) та тональний сигнал із синусоїдальною модуляцією частоти. Синтетичні сигнали використовуються для візуального аналізу геометричних перетворень на спектрограмі. Використання сигналів із лінійною або синусоїдальною структурою дозволяє наочно відстежити розриви, зміщення частотних підсмуг, переміщення часових сегментів та спостерігати виникнення крайових ефектів під час матричних маніпуляцій. Послідовне комбінування синтетичних та реальних сигналів забезпечує розуміння математичних аспектів цифрових перетворень та їхніх фізичних наслідків для аудіоданих.

Розгорнуті методичні вказівки щодо роботи з віртуальним макетом у середовищі MATLAB, покроковий порядок виконання дослідження та матриця індивідуальних варіантів завдань наведені у Додатку А до дипломної роботи.

Висновки до розділу 3

Досліджено, програмно реалізовано та математично обґрунтовано механізми керування процесом двовимірного скремблювання за допомогою

унікальних кодових послідовностей, які виконують роль секретного ключа системи. Доведено, що безпека аналогового приховування інформації повністю залежить від динамічної зміни параметрів перемішування, без знання яких сторонній спостерігач позбавлений можливості відновити первинну структуру мовного сигналу.

4. ДОСЛІДЖЕННЯ СТІЙКОСТІ СКРЕМБЛЬОВАНИХ СИГНАЛІВ ДО СИСТЕМ АВТОМАТИЧНОГО РОЗПІЗНАВАННЯ МОВИ ТА ТРАНЗИТНИХ ВОКОДЕРІВ

4.1 Методологія та критерії оцінки ефективності маскування

У сучасних телекомунікаційних мережах перехоплення та аналіз інформації здійснюється як традиційним способом, так і за допомогою масового застосування систем автоматичного розпізнавання мови (Automatic Speech Recognition, ASR). З огляду на це, об'єктивний критерій якості аналогового скремблювання має бути комплексним і включати перевірку стійкості зашифрованого сигналу як до сприйняття людським слухом, так і до транскрибації сучасними нейромережевими алгоритмами.

З іншого боку, актуальною залишається проблема викривлення аналогових сигналів під час їх проходження через навантажені мережі зв'язку. У сучасних мобільних мережах для збереження пропускної здатності застосовується транзитне стиснення голосу. На відміну від ідеального аналогового каналу тональної частоти, такий транзитний тракт не можна розглядати як лінійну стаціонарну систему передавання форми сигналу.

Зокрема, стандартний кодек GSM AMR-NB є параметричним мовним кодеком сімейства CELP/ACELP. Він не передає форму хвилі безпосередньо, а кожні 20 мс оцінює параметри голосового тракту людини (короткочасну спектральну огинаючу, основний тон, кодове збудження) і передає саме їх. Для природної мови такий підхід є ефективним, проте скрембльований сигнал після частотно-часових перестановок втрачає природну акустичну структуру. У цьому випадку транзитний AMR-кодер примусово апроксимує нетиповий для нього сигнал найближчим набором допустимих мовних параметрів, що вносить сильні нелінійні та незворотні спотворення.

Для тестування стійкості системи до таких умов було обрано вплив кодека стандарту GSM AMR-NB у режимі максимального стиснення (MR475, бітрейт 4.75 кбіт/с) [6] [16]. Цей режим автоматично вмикається базовими

станціями за умови максимального навантаження на мережу, характеризується найвищим рівнем параметричних втрат та є найгіршим (стресовим) сценарієм для каналу передачі. Головне завдання цього етапу тестування — довести, що навіть після агресивної нелінійної компресії в каналі зв'язку сигнал зберігає структурну цілісність, достатню для успішного дескремблювання та розуміння повідомлення легітимним користувачем.

Для відходу від суб'єктивних акустичних оцінок та уникнення громіздких табличних даних, у цьому дослідженні запропоновано автоматизовану методику тестування на основі метрики відсотка успішно розпізнаних слів (Word Recognition Rate, WRR) за допомогою нейромережі «Transkriptor» [10]. Оскільки сучасні ASR-системи наближаються до рівня людського сприйняття, використання такої автоматизованої метрики дозволяє отримати універсальну кількісну оцінку: вона одночасно демонструє рівень маскування від машинного перехоплення та виступає об'єктивним мірилом загальної розбірливості мови для людини.

Для автоматичного порівняння гіпотез (текстів, виданих нейромережею) з еталонним текстом було розроблено спеціалізований програмний скрипт. Він автоматизує розрахунок кількості прямих збігів слів та обчислює підсумковий відсоток розпізнавання. Це дозволяє агрегувати отримані дані та представити результати порівняльного аналізу різних алгоритмів у вигляді графічних залежностей.

4.2 Проведення експерименту та програмна оцінка результатів

Для проведення комплексного аналізу було сформовано набір тестових даних, що охоплює три розроблені методи захисту: часове, частотне та комбіноване двовимірне (2D) скремблювання. Процес тестування кожного алгоритму було стандартизовано та розбито на чотири послідовні етапи:

Базове оцінювання: Транскрибація чистого відкритого мовного сигналу для формування еталонного тексту та подальшого порівняння.

Оцінка маскування: Транскрибація зашифрованих масивів даних. Аналізується як прямий скрембльований сигнал, так і сигнал після впливу транзитного кодека.

Оцінка стійкості: Транскрибація повідомлення після його зворотного перетворення. Перевірка ідеального математичного відновлення та перевірка відновлення після незворотних фазових і амплітудних спотворень транзитного кодека MR475.

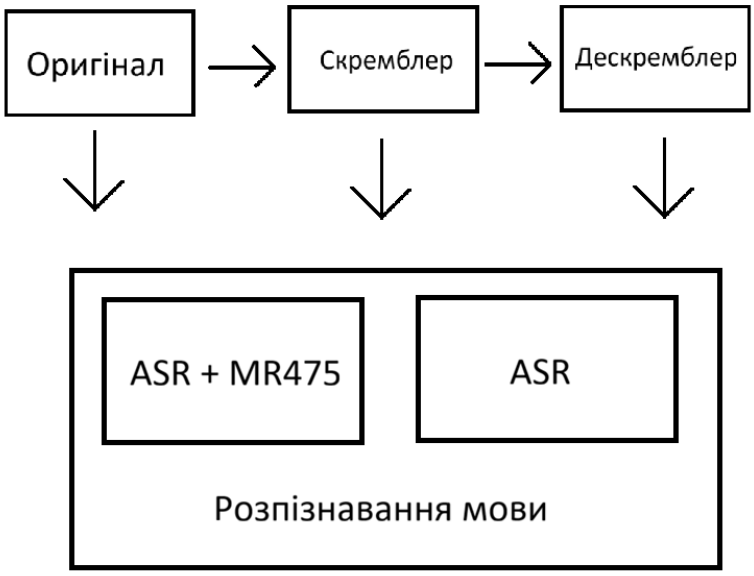

Рисунок 4.1 – Блок-схема маршрутизації сигналів під час тестування стійкості до систем автоматичного розпізнавання мови

Алгоритм автоматично очищує вхідні тексти від знаків пунктуації, зводить їх до нижнього регістру та послідовно перевіряє наявність кожного слова з еталона у транскрипції ASR. Такий підхід дозволяє ігнорувати порушення порядку слів та наявність сторонніх артефактів розпізнавання, коректно фіксуючи навіть частково вцілілі фрагменти мови.

За результатами покрокового аналізу скрипт обчислює підсумкову метрику — відсоток успішно розпізнаних слів (Word Recognition Rate, WRR) та формує порівняльну гістограму для всіх етапів експерименту.

4.3 Аналіз результатів тестування та висновки щодо стійкості алгоритмів

За результатами автоматизованої обробки транскрипцій для кожного етапу експерименту було побудовано порівняльну гістограму.

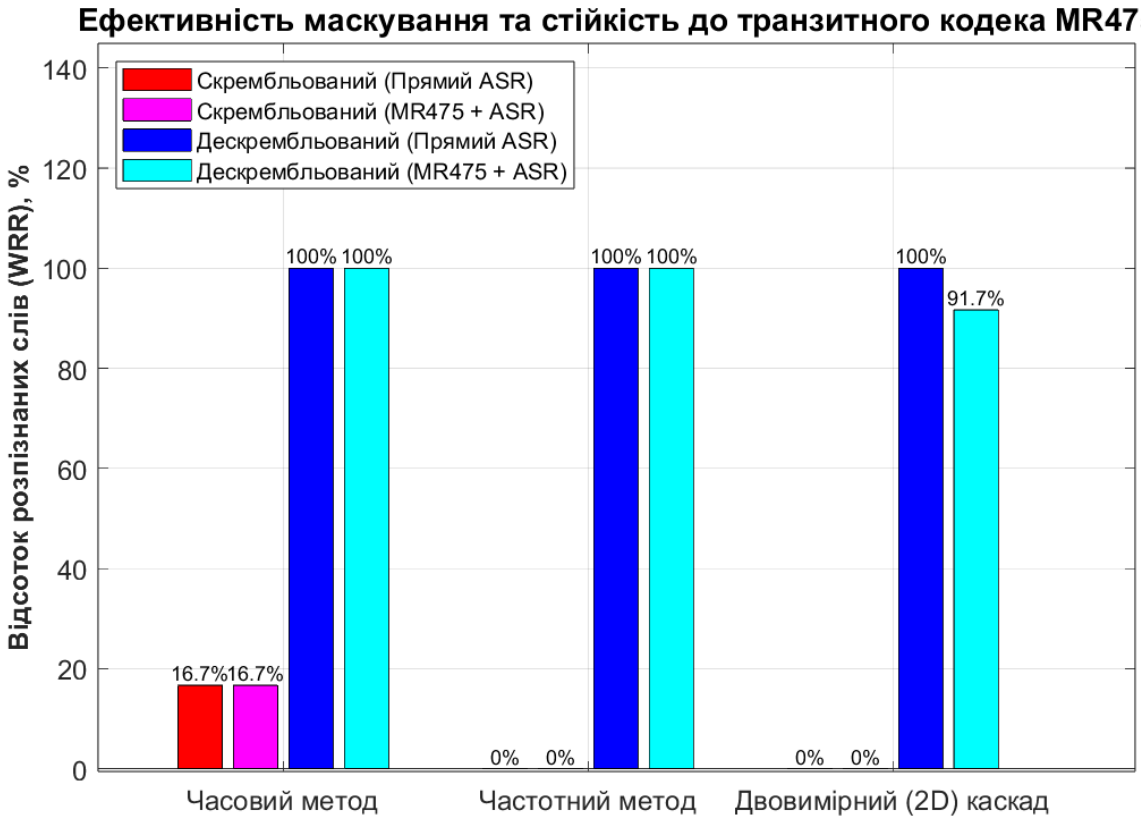

Рисунок 4.2 – Залежність відсотка розпізнаних слів (WRR) від методу скремблювання та впливу транзитного кодека MR475

Аналіз отриманих графічних залежностей дозволяє оцінити характеристики розроблених методів за двома критеріями: ефективністю маскування мовної інформації та стійкістю до деградації у вузькосмуговому каналі зв'язку.

Результати експерименту показали, що частотний та двовимірний (2D) методи забезпечують високий рівень маскування від ASR-систем: показник розпізнавання знизився до 0%, що вказує на критичне порушення структури акустичних ознак сигналу. Однак слід враховувати специфіку людського слуху: тоді як 2D-скрембльований сигнал стає нерозбірливим, виключно частотне перетворення залишає можливість розпізнавання окремих фонем або слів

підготовленим слухачем. Це підтверджує обмеженість одномірних підходів і демонструє, що саме двовимірний каскад забезпечує найбільш надійний рівень захисту від перехоплення в межах даного тестування.

Результати транскрибації дескрембльованих сигналів підтверджують ефективність розроблених систем. В умовах ідеального каналу (без впливу транзитного кодека) усі три алгоритми забезпечують стовідсоткове відновлення макроструктури мови, що доводить математичну точність прямих та зворотних перетворень. Під час моделювання умов комерційного каналу зв'язку, після параметричного стиснення кодеком GSM AMR у режимі MR475, відновлені сигнали також зберегли високу розбірливість. Зокрема, для часового та частотного методів залишкова розбірливість у межах тестової вибірки наблизилася до максимуму. Для найскладнішого двовимірного (2D) каскаду показник успішного розпізнавання (WRR) склав 91.7%. Таке незначне зниження є цілком очікуваним інженерним компромісом, оскільки багаторазові частотно-часові маніпуляції у поєднанні з інформаційними втратами транзитного кодека неминуче вносять незворотні фазові та амплітудні спотворення [7]. Попри це, підсумковий показник понад 90% гарантує розуміння змісту повідомлення легітимним отримувачем.

Висновки до розділу 4

Проведене автоматизоване тестування доводить, що перехід від одномірного до двовимірного каскадного скремблювання є цілком виправданим кроком. У межах проведених експериментів двовимірний алгоритм продемонстрував найвищий рівень захисту від перехоплення системами автоматичного розпізнавання (0% розпізнавання). Запропонований аналоговий підхід зберігає свою працездатність та забезпечує високу залишкову розбірливість (понад 91%) навіть за найгірших умов передачі у вузькосмугових мережах. Водночас, зважаючи на відносно невеликий обсяг тестової вибірки на поточному етапі, ці результати слід розглядати як базове підтвердження працездатності концепції. Для повноцінної валідації

розроблених методів та отримання більш репрезентативної статистики необхідно провести подальші дослідження з використанням розширених наборів мовних даних.

Результати розділу 4 слід трактувати як перевірку двох різних властивостей. Перша властивість — маскування змісту мовлення від систем автоматичного розпізнавання. Друга властивість — збереження достатньої часово-частотної структури для зворотного відновлення після проходження через параметричний AMR-кодек. Для лінійного каналу друга властивість випливає з математичної оборотності алгоритму. Для AMR-каналу вона повинна підтверджуватися експериментально, оскільки AMR-4750 є нелінійним нестаціонарним перетворювачем з втратами.

5. ДОСЛІДЖЕННЯ МОЖЛИВОСТІ РЕАЛІЗАЦІЇ СКРЕМБЛЮВАННЯ ТА ДЕСКРЕМБЛЮВАННЯ АУДІОСИГНАЛІВ, КЕРОВАНИХ КОДОМ В РЕАЛЬНОМУ ЧАСІ

5.1. Стислий опис навчального модуля EZ-Kit-Lite ADSP-2181 та аналіз можливості його застосування

Теоретичні алгоритми, змодельовані у високорівневих програмних середовищах, вимагають суттєвої адаптації під час перенесення на реальну цільову апаратну платформу. Реалізація спектрально-часових перетворень виконується на базі навчального модуля EZ-Kit-Lite із сигнальним процесором ADSP-2181. Цей обчислювальний пристрій характеризується 16-бітною архітектурою з фіксованою точкою (fixed-point) та базується на модифікованій гарвардській архітектурі [12] [17]. Наявність ізольованих шин для пам'яті програм та пам'яті даних дозволяє системі здійснювати одночасну вибірку інструкції та двох операндів за один машинний такт, що забезпечує достатню продуктивність для потокової цифрової обробки сигналів.

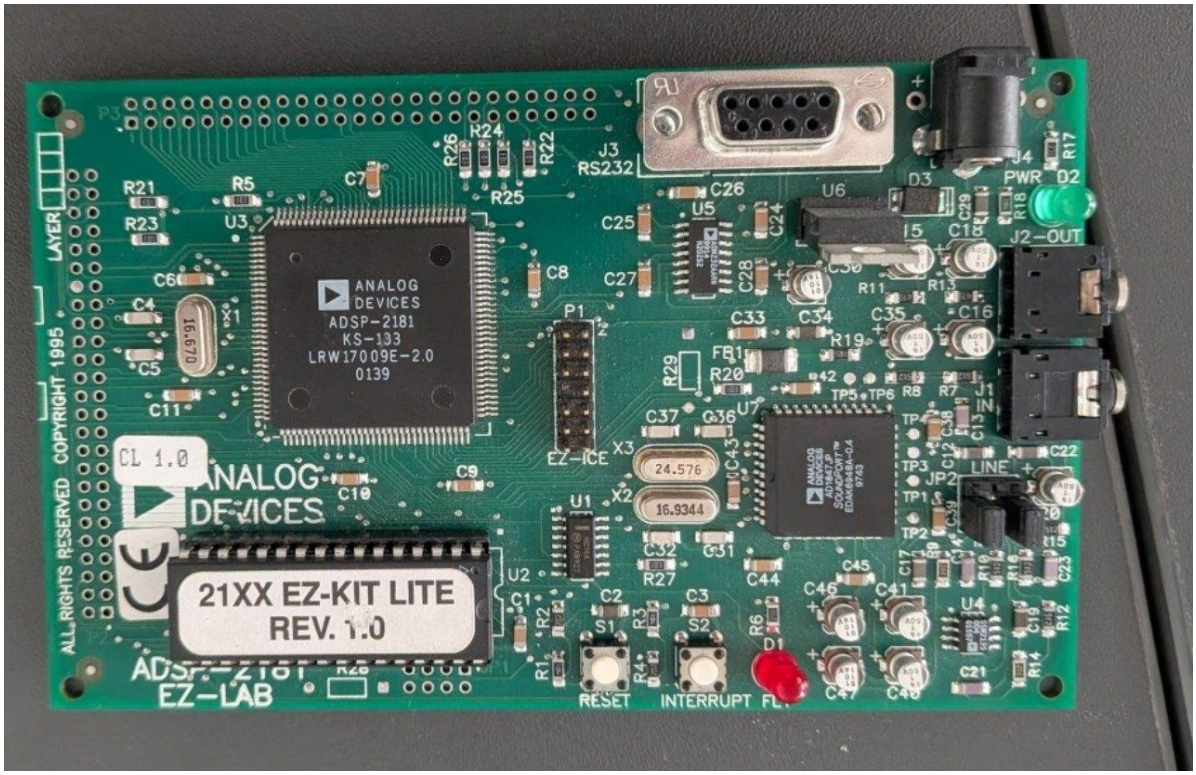

Рисунок 5.1 – Зовнішній вигляд навчального макета EZ-Kit Lite ADSP-2181.

Для розуміння шляху проходження та обробки сигналу необхідно розглянути внутрішню апаратну архітектуру цільового мікропроцесора (рис. 5.2).

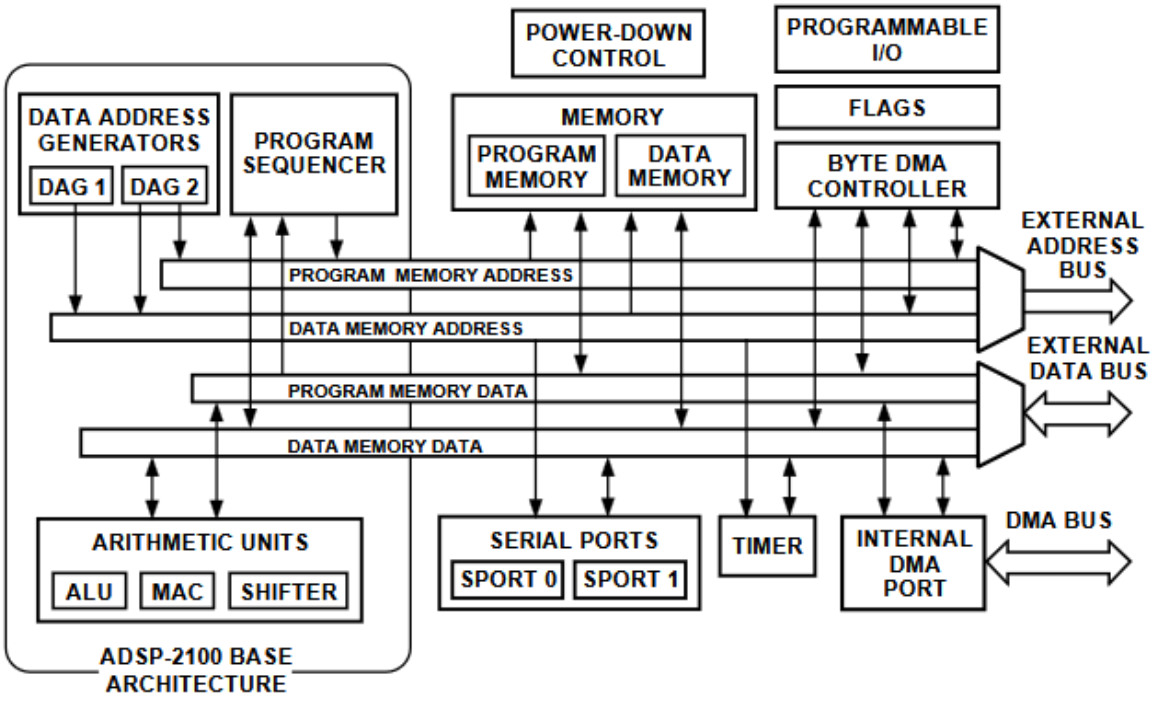

Рисунок 5.2 – Функціональна блок-схема внутрішньої архітектури мікропроцесора ADSP-2181 [12]

Архітектура забезпечує незалежні шини для пам'яті програм (Program Memory) та пам'яті даних (Data Memory). Це дозволяє арифметико-логічному пристрою (ALU) та помножувачу (MAC) отримувати інструкцію та два операнди за один машинний такт. Для обміну аудіоданими з кодеком AD1847 SoundPort використовується вбудований синхронний послідовний порт SPORT0. Важливу роль відіграють апаратні генератори адрес (DAG), які забезпечують швидку циклічну буферизацію та біт-реверсивну адресацію, що є необхідним для оптимізованого обчислення швидкого перетворення Фур'є (ШПФ) мовою ANSI-C.

Вибір даного мікропроцесора як цільової апаратної бази зумовлений зручністю його використання у навчальному процесі. Хоча цей чип є представником класичних рішень і за багатьма показниками поступається

новітнім розробкам, його архітектура залишається відкритою та максимально зрозумілою для вивчення. Відсутність складних багаторівневих абстракцій дозволяє студентам працювати безпосередньо з пам’яттю, розуміти специфіку арифметики з фіксованою точкою та налаштовувати потокове введення-виведення звуку на апаратному рівні. При цьому використання стандартизованої мови С робить написані алгоритми універсальними. У разі необхідності перенесення розробленої системи скремблювання на сучасне телекомунікаційне обладнання, створений код слугує гарною основою для адаптації на актуальні обчислювальні ядра, зокрема на мікроконтролери сімейства ARM Cortex-M або потужніші процесори Cortex-A.

Взаємодія ядра процесора із зовнішньою периферією, зокрема з аудіокодеком для виконання операцій аналого-цифрового та цифро-аналогового перетворення, реалізована за допомогою потоків (streams). Під час апаратної реалізації замість стандартного читання масивів дані надходять у спеціальні регістри процесора через механізм вводу-виводу.

На рівні коду мовою ANSI-C така архітектура вимагає специфічного декларування змінних вхідного та вихідних потоків stream_in, stream_out_scrambled та stream_out_descrambled. Використання кваліфікатора volatile для цих портів є вказівкою компілятору вимкнути будь-яку кешуючу оптимізацію звернень до відповідних комірок пам'яті. Значення цих регістрів оновлюються апаратно і є непередбачуваними для програми на кожній ітерації нескінченного циклу while(1), призначеного для обробки аудіокадрів у режимі реального часу.

```
volatile int stream_in;
volatile int stream_out_scrambled;
volatile int stream_out_descrambled;
```

Процес обробки потокового аудіо на базі платформи ADSP-2181 вимагає врахування її обчислювальних обмежень. З метою імітації умов стандартного

телефонного каналу зв'язку частота дискретизації обмежується значенням 8 кГц. У високорівневих математичних моделях для згладжування артефактів на межах оброблюваних фрагментів традиційно застосовується віконне короткочасне перетворення Фур'є з перекриттям кадрів та подальшим синтезом за алгоритмом Overlap-Add [10]. Однак, перенесення такого підходу на 16-бітний процесор із фіксованою точкою генерує значну проблему обчислювальної складності.

Перекриття кадрів у масштабі реального часу на обраній апаратній базі вимагає організації складної конвеєризації потоків даних із залученням каналів прямого доступу до пам'яті (DMA) та використання системи кільцевих "ping-pong" буферів. З метою уникнення перевантаження навчального макета було прийнято компроміс: відмовитися від ресурсомісткого перетворення з перекриттям на користь лінійної обробки ізольованими кадрами фіксованого розміру у 256 відліків. Цей метод обробки неминуче вносить певні блокові артефакти у відновлений звуковий сигнал, проте гарантує стабільне та безперервне виконання необхідних криптографічних маніпуляцій без порушення часових рамок реального часу. Зазначені апаратні обмеження та обрана стратегія покадрової обробки вимагають розробки спеціалізованого програмного забезпечення, яке розглядається у наступному підрозділі.

5.2. Програмна реалізація алгоритмів двовимірного скремблювання мовою ANSI-C

Під час перенесення високорівневої MATLAB-моделі на платформу ADSP-2181 виникає необхідність переходу від статичної файлової обробки загального масиву даних до динамічної потокової покадрової обробки в режимі реального часу (рис. 5.3).

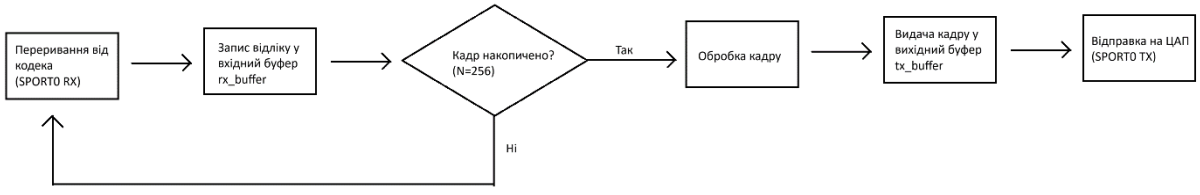

Рисунок 5.3 – Логіка покадрової обробки аудіосигналу в реальному часі

В апаратній реалізації вхідні відліки надходять не як готовий аудіофайл, а як безперервна послідовність значень від кодека через порт SPORT0. Тому програмна структура містить механізм переривань, вхідний буфер rx_buffer та вихідний буфер tx_buffer. Після накопичення кадру фіксованої довжини (256 відліків) виконується частотно-часова перестановка, після чого сформований скрембльований кадр послідовно передається на вихідний аудіотракт. Таке рішення без перекриття кадрів є необхідним компромісом для запобігання перевантаженню 16-бітного процесора.

Відповідно до наведеної логіки покадрової обробки, розробка програмного забезпечення мовою ANSI-C вимагає контролю над пам'яттю мікропроцесора. Усі згадані масиви для зберігання та проміжної обробки відліків (зокрема rx_buffer, tx_buffer та temp_time) оголошуються у коді як статичні глобальні змінні з фіксованим розміром FRAME_SIZE, що становить 256 відліків типу short. Використання такої статичної структури даних є необхідністю: у DSP-системах реального часу забороняється динамічне виділення пам'яті (наприклад, за допомогою стандартної функції malloc). Такий підхід дозволяє уникнути фрагментації пам'яті та гарантує стабільну швидкодію процесора під час безперервного надходження аудіопотоку.

```
short rx_buffer[FRAME_SIZE];
short temp_time[FRAME_SIZE];
short temp_freq[FRAME_SIZE];
short temp_desc[FRAME_SIZE];
short tx_buffer[FRAME_SIZE];
```

Для забезпечення процесу дескремблювання розроблено алгоритм автоматичної генерації зворотних параметрів криптографічної системи. Функція get_inverse_key_c на основі прямого масиву перестановок обчислює масив оригінальних інверсних індексів, що необхідні для відновлення початкового порядку сегментів на приймальній стороні. Алгоритмічне

перетворення в часовій області реалізовано у функції time_scramble_c. Це перетворення полягає у сегментації загального аудіокадру на окремі рівні сегменти, загальна кількість яких визначається макросом NUM_CHUNKS і дорівнює 4.

```
      void get_inverse_key_c(int* original_key, int* inverse_key, int num_elements)
{
          int i, j;
          for (i = 0; i < num_elements; i++) {
              int target = i + 1;
              for (j = 0; j < num_elements; j++) {
                  if (abs(original_key[j]) == target) {
                      inverse_key[i] = (original_key[j] < 0) ? -(j + 1) : (j + 1);
                      break;
                  }
              }
          }
      }
```

Процес перестановки здійснюється шляхом копіювання відліків за допомогою арифметики вказівників (src_ptr та dst_ptr), адресація яких розраховується згідно з абсолютним значенням поточного елемента ключа. Окремо алгоритм перевіряє полярність керуючого елемента: якщо поточне значення ключа є від'ємним, активується внутрішня часова інверсія. У такому випадку виконується дзеркальне відображення відліків усередині сегмента задом наперед під час їхнього запису в цільовий буфер.

```
      void time_scramble_c(short* input, short* output, int* key, int num_samples,
int num_chunks) {
          int chunk_size = num_samples / num_chunks;
          int i, j;
          for (i = 0; i < num_chunks; i++) {
              int current_key = key[i];
              int source_idx = abs(current_key) - 1;
              int is_inverted = (current_key < 0) ? 1 : 0;

              short* src_ptr = &input[source_idx * chunk_size];
```

```
                short* dst_ptr = &output[i * chunk_size];

                for (j = 0; j < chunk_size; j++) {
                    dst_ptr[j] = is_inverted ? src_ptr[chunk_size - 1 - j] :
src_ptr[j];
                }
            }
        }
```

Для переходу в частотну область алгоритм частотного скремблювання, реалізований у функції freq_scramble_c, вимагає специфічної підготовки пам'яті для апаратного виконання ШПФ. Робота з комплексними спектральними даними на базі процесора ADSP-2181 забезпечується шляхом використання окремих цілочисельних масивів in_real, in_imag для вхідних даних та fft_real, fft_imag для результатів перетворення, розмір кожного з яких складає 256 відліків.

Критичною вимогою для коректного виконання перетворення є застосування спеціальної директиви компілятора #pragma align 256 перед оголошенням кожного із зазначених масивів. Використання цієї директиви необхідно, оскільки апаратні генератори адрес (Data Address Generators, DAG) сигнального процесора вимагають суворого вирівнювання пам'яті під час виконання специфічних інструкцій цифрової обробки [12].

```
#pragma align 256
int in_real[FRAME_SIZE];
#pragma align 256
int in_imag[FRAME_SIZE];
#pragma align 256
int fft_real[FRAME_SIZE];
#pragma align 256
int fft_imag[FRAME_SIZE];
```

Фізика апаратного процесу полягає в тому, що оптимізована асемблерна функція fft256 використовує апаратний механізм циклічної буферизації пам'яті

(circular buffering) у комбінації з біт-реверсивною адресацією для надшвидкого доступу до операндів під час обчислення.

Без цього вирівнювання апаратне обчислення адрес за модулем працюватиме некоректно: під час переходу через межу масиву вказівник вказуватиме на хибні комірки пам'яті (відбудеться некоректне циклічне перенесення адреси). Порушення цієї вимоги не призведе до фізичної зупинки чи збою процесора, але неминуче спричинить тихе пошкодження сусідніх областей пам'яті та повну руйнацію спектральних даних аудіосигналу на етапі обчислення перетворення.

Логіка частотного скремблювання, реалізована у функції freq_scramble_c, полягає в частотній селекції та перестановці спектральних компонент. Спектр аудіосигналу, отриманий після прямого апаратного перетворення, поділяється на окремі робочі смуги, які переміщуються на нові позиції відповідно до закладеного масиву ключів.

```
void freq_scramble_c(short* input, short* output, int* key, int num_samples, int num_bands) {
    int i, j, k;
    int exp_fft, exp_ifft, total_exp, shift;

    // 1. Підготовка
    for (i = 0; i < FRAME_SIZE; i++) {
        in_real[i] = (int)input[i];
        in_imag[i] = 0;
    }

    // 2. ШПФ
    exp_fft = fft256(in_real, in_imag, fft_real, fft_imag);

    // Очищаємо буфери для збирання нового спектру
    for (i = 0; i < FRAME_SIZE; i++) {
        in_real[i] = 0;
        in_imag[i] = 0;
    }

    // Зберігаємо краї (DC та Найквіста)
```

```
            in_real[0] = fft_real[0]; in_imag[0] = fft_imag[0];
            for (i = 125; i <= 128; i++) {
                in_real[i] = fft_real[i];
                in_imag[i] = fft_imag[i];
            }

            // 3. ПЕРЕСТАНОВКА СМУГ (Scrambling)
            for (i = 0; i < num_bands; i++) {
                int current_key = key[i];
                int source_band_idx = abs(current_key) - 1;
                int is_inverted = (current_key < 0) ? 1 : 0;

                int src_start = 1 + source_band_idx * BINS_PER_BAND;
                int dst_start = 1 + i * BINS_PER_BAND;

                for (j = 0; j < BINS_PER_BAND; j++) {
                    int src_idx = is_inverted ? src_start + (BINS_PER_BAND - 1 - j) :
src_start + j;
                    int dst_idx = dst_start + j;

                    in_real[dst_idx] = fft_real[src_idx];
                    in_imag[dst_idx] = fft_imag[src_idx];
                }
            }
```

Для забезпечення коректної роботи зворотного перетворення алгоритм формує дзеркальне відображення для відповідних симетричних індексів FRAME_SIZE - k. Цей процес реалізований через обчислення комплексного спряження уявних компонентів, де уявна частина приймає значення - in_imag[k].

```
            // 4. Ермітова СИМЕТРІЯ
            for (k = 1; k < 128; k++) {
                in_real[FRAME_SIZE - k] = in_real[k];
                in_imag[FRAME_SIZE - k] = -in_imag[k]; // Сопряжение
            }

            // 5. ЗШПФ
```

```
        exp_ifft = ifft256(in_real, in_imag, fft_real, fft_imag);

        // 6. Компенсація
        total_exp = exp_fft + exp_ifft;
        shift = total_exp - 8;

        for(i = 0; i < FRAME_SIZE; i++) {
            long temp = (long)fft_real[i];

            if (shift > 0) temp = temp << shift;
            else if (shift < 0) temp = temp >> (-shift);

            if (temp > 32767) temp = 32767;
            if (temp < -32768) temp = -32768;

            output[i] = (short)temp;
        }
    }
```

Після виконання спектральних перестановок у матрицях уявних і дійсних компонентів здійснюється перехід назад у часову область. Для цього викликається вбудована апаратно-оптимізована функція зворотного ШПФ ifft256. Оскільки арифметика процесора ADSP-2181 базується на фіксованій точці, операції інтегрування у частотній області призводять до природного розростання динамічного діапазону даних, що загрожує переповненням 16-бітної розрядної сітки. Компенсація цього ефекту реалізується через обчислення сумарної експоненти прямого та зворотного перетворень (total_exp = exp_fft + exp_ifft) та розрахунок величини динамічного побітового зсуву за формулою shift = total_exp - 8. На основі отриманого значення виконується побітове зміщення масиву результатів ліворуч або праворуч через оператори зсуву << та >>. Для запобігання виходу значень за межі апаратного формату short у коді впроваджено механізм обмеження (кліпінгу), який фіксує відліки на рівнях максимальних амплітуд від -32768 до 32767 перед їх записом у вихідний буфер.

```
void main() {
    int i;

    get_inverse_key_c(time_key, time_inv_key, NUM_CHUNKS);
    get_inverse_key_c(freq_key, freq_inv_key, NUM_BANDS);

    while(1) {
        // --- ЧИТАННЯ ---
        for(i = 0; i < FRAME_SIZE; i++) {
            rx_buffer[i] = stream_in;
        }

        // --- ШИФРУВАННЯ (2D) ---
        time_scramble_c(rx_buffer, temp_time, time_key, FRAME_SIZE,
NUM_CHUNKS);
        freq_scramble_c(temp_time, temp_freq, freq_key, FRAME_SIZE,
NUM_BANDS);

        // --- ДЕШИФРУВАННЯ ---
        freq_scramble_c(temp_freq, temp_desc, freq_inv_key, FRAME_SIZE,
NUM_BANDS);
        time_scramble_c(temp_desc, tx_buffer, time_inv_key, FRAME_SIZE,
NUM_CHUNKS);

        // --- ВИВОД ---
        for(i = 0; i < FRAME_SIZE; i++) {
            stream_out_scrambled = temp_freq[i];
            stream_out_descrambled = tx_buffer[i];
        }
    }
}
```

Головна функція програми main() організує загальну координацію каскадів обробки аудіосигналу в реальному часі. На початковому етапі, до входу в основний цикл, виконується одноразова генерація інверсних ключів для часової та частотної областей за допомогою функції get_inverse_key_c. Безперервна потокова обробка відбувається у нескінченному циклі while(1), де

відліки покроково зчитуються з апаратного регістра stream_in у буфер rx_buffer до заповнення кадру (256 точок). Двовимірне скремблювання здійснюється шляхом послідовного виклику функції часових перестановок time_scramble_c та передачі проміжного масиву на вхід модуля частотного шифрування freq_scramble_c. Фінальні відліки зашифрованої суміші та відновленого аудіосигналу паралельно виводяться у відповідні периферійні регістри stream_out_scrambled та stream_out_descrambled.

5.3. Симуляційне тестування розробленого програмного забезпечення та аналіз апаратних артефактів.

Тестування розроблених алгоритмів базується на імітації апаратних процесів сигнального процесора. Підготовка вхідного аудіопотоку для симулятора VisualDSP++ реалізується за допомогою програмного модуля generate_stream.m, який виконує функцію віртуального аналого-цифрового перетворювача (АЦП). Базовим тестовим впливом виступає лінійно-частотно-модульований (ЛЧМ) сигнал. Для узгодження з 16-бітною архітектурою сигнального процесора з фіксованою точкою масив значень із рухомою точкою піддається програмному квантуванню: амплітуди масштабуються на максимальне значення розрядної сітки (32767) із подальшим перетворенням у цілочисельний формат int16 [20].

```
original_int16 = int16(original_audio * 32767);

% Експорт у файл для симулятора Visual DSP++
filename = 'input_stream.dat';
fid = fopen(filename, 'w');
for i = 1:length(original_int16)
    % Пишемо по одному числу в рядок (десятковий формат)
    fprintf(fid, '%d\n', original_int16(i));
end
fclose(fid);
```

Сформований масив відліків експортується у текстовий дамп-файл input_stream.dat та інтегрується в налаштування віртуального вхідного порту симулятора. Такий підхід забезпечує потактову вибірку та обробку даних виконуваним ANSI-C кодом, імітуючи апаратний обмін потоковою інформацією з фізичним аудіокодеком у режимі реального часу.

Зчитування та візуальний аналіз результатів моделювання здійснюється скриптом analyze_dsp.m, який завантажує згенеровані симулятором вихідні дампи пам'яті scrambled_out.dat та descrambled_out.dat. Критичним етапом постобробки є програмна нормалізація амплітуди отриманих масивів. Під час апаратного обчислення ШПФ процесор застосовує побітові зсуви (Block Floating Point) для запобігання переповненню розрядної сітки, що викликає затухання загального рівня сигналу. Розрахунок та застосування компенсаційного коефіцієнта scale_factor нівелює це приглушення та відновлює початковий масштаб гучності, забезпечуючи коректні умови для порівняння обробленого аудіосигналу з еталонним.

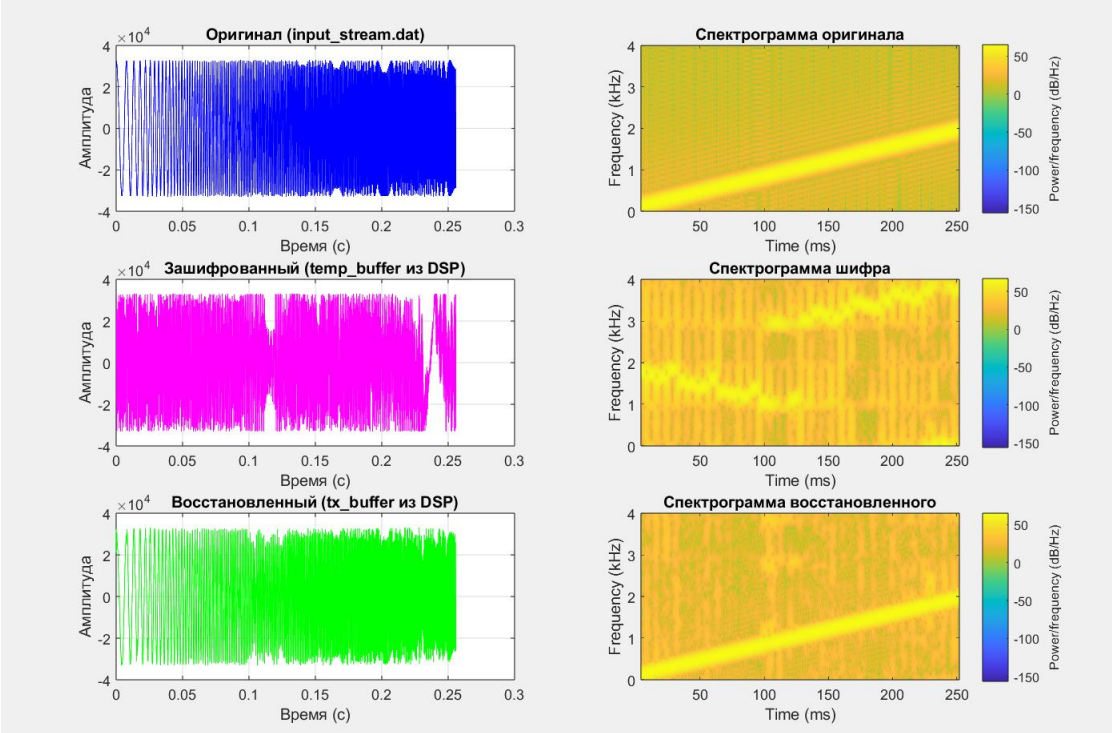

Рисунок 5.2 – Результати симуляційного тестування алгоритмів у середовищі VisualDSP++

Аналіз згенерованих осцилограм і спектрограм фіксує наявність залишкових візуальних та акустичних артефактів. Виникнення цих спотворень зумовлене обчислювальними обмеженнями 16-бітної арифметики і є об'єктивним технічним компромісом, необхідним для забезпечення стабільної роботи алгоритмів двовимірного скремблювання в режимі реального часу. Попри наявність шумів квантування та блокових ефектів, результати симуляції підтверджують збереження загальної цілісності переданого повідомлення та працездатність розробленого програмного забезпечення на базі цільового сигнального процесора.

5.4. Використання апаратно-програмного макета в навчальному процесі

Розроблений та верифікований програмний комплекс адаптований для інтеграції у навчальний процес у межах практичних занять із дисципліни «Сигнальні процесори в радіосистемах». Лабораторний макет побудовано за концепцією апаратного моделювання в контурі (Hardware-in-the-Loop), що дозволяє студентам взаємодіяти з архітектурою процесора ADSP-2181 через симулятор середовища VisualDSP++.

Дидактична мета розробки полягає не лише у застосуванні алгоритмів частотно-часових перестановок, а й у практичному дослідженні впливу апаратних обмежень процесора на якість обробки аудіосигналів. Для розширення інженерних навичок індивідуальні завдання базуються на зміні параметрів вхідного потоку: форми тестового сигналу, його частотного діапазону та коефіцієнта початкової амплітуди. Варіативність амплітуд дозволяє студентам фіксувати ефекти обмеження (кліпінгу) внаслідок переповнення розрядної сітки під час обчислення ШПФ, а також деградацію корисного спектра через шуми квантування при роботі з сигналами низького рівня.

Детальні інструкції з налаштування віртуальних потоків даних (Streams), порядок проведення апаратної симуляції та матриця індивідуальних завдань

для виконання другої лабораторної роботи наведені у Додатку Б до дипломної роботи.

Висновки до Розділу 5

Досліджено та реалізовано процес перенесення математичних алгоритмів на цільову апаратну платформу — сигнальний процесор ADSP-2181 на базі навчального модуля EZ-Kit-Lite. Для забезпечення обробки аудіопотоку в режимі реального часу програмне забезпечення було розроблено мовою ANSI-C з урахуванням обмежень 16-бітної архітектури з фіксованою точкою.

З метою подолання обчислювальних бар'єрів здійснено перехід до покадрової обробки фіксованими блоками по 256 відліків та застосовано апаратно-оптимізовані функції ШПФ із вирівнюванням пам'яті для коректної роботи генераторів адрес. Для запобігання переповненню розрядної сітки під час спектральних маніпуляцій впроваджено механізми динамічного побітового зсуву та кліпінгу.

Тестування реалізованих алгоритмів у середовищі VisualDSP++ довело їхню працездатність: розроблений комплекс здійснює пряме та зворотне двовимірне скремблювання без порушення часових рамок реального часу. Наявні залишкові акустичні артефакти визначені як допустимий компроміс для систем із фіксованою точкою, що підтверджує ефективність обраного інженерного підходу та придатність системи для практичного застосування у навчальних радіосистемах.

ВИСНОВКИ

У дипломній роботі вирішено актуальне інженерне завдання щодо підвищення рівня конфіденційності мовних каналів телекомунікаційних систем із обмеженою пропускною здатністю. Аналіз існуючих підходів показав, що класичне одномірне скремблювання є вразливим до спектрального криптоаналізу, а використання цифрових методів шифрування вимагає значних обчислювальних ресурсів і вносить суттєві затримки. Доведено, що двовимірне каскадне скремблювання є оптимальним компромісом, оскільки воно експоненціально розширює простір ключів та підвищує стійкість до перехоплення, зберігаючи вихідну вузьку смугу пропускання аналогового каналу.

У середовищі MATLAB створено та верифіковано комплекс алгоритмів для виконання матричних часових та частотних перестановок із застосуванням внутрішньої спектрально-часової інверсії блоків. Аналіз згенерованих спектрограм та осцилограм підтвердив повну руйнацію вихідної структури сигналу та довів математичну оборотність процесів дескремблювання за наявності легітимного ключа. Високу стійкість алгоритмів доведено результатами автоматизованого тестування за допомогою систем автоматичного розпізнавання мови (ASR), де двовимірний каскад забезпечив максимальний рівень маскування — 0% розпізнаних слів стороннім спостерігачем. Водночас після впливу агресивної компресії транзитного кодека стандарту GSM AMR-NB (режим MR475) залишкова розбірливість дескрембльованого повідомлення склала понад 91,7%, що гарантує надійне розуміння інформації легітимним користувачем.

Створені математичні алгоритми було успішно портовано мовою ANSI-C на 16-бітний сигнальний процесор із фіксованою точкою ADSP-2181 для роботи в реальному часі. Під час розробки вирішено критичні апаратні обмеження: здійснено перехід до покадрової обробки, виконано суворе вирівнювання пам’яті для коректної роботи генераторів адрес під час

обчислення ШПФ, а також впроваджено механізми динамічного побітового зсуву та кліпінгу. Розроблений програмно-апаратний комплекс адаптовано для інтеграції у навчальний процес як лабораторні макети з метою вивчення принципів цифрової обробки сигналів та методів аналогового приховування інформації.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sadkhan S. B., Al-Saffar A. A. An overview of speech encryption techniques. International Journal of Engineering Research and Development. 2012. Vol. 3, No. 4. P. 29–38.
2. Jassim M. H., Al-Haidari F. A. Speech scrambling based on principal component analysis. Journal of University of Babylon. 2009. Vol. 17, No. 1. P. 1–10.
3. Mitchell C. J., Piper F. C. A classification of time element speech scramblers. Journal of the Institution of Electronic and Radio Engineers. 1985. Vol. 55, No. 11-12. P. 391–396.
4. Del Re E., Fantacci R., Maffucci D. A new speech signal scrambling method for secure communications: Theory, implementation, and security evaluation. IEEE Transactions on Communications. 1989. Vol. 37, No. 1. P. 133–140.
5. Nimbhorkar A., et al. Performance evaluation of speech scrambling. International Journal of Computer Applications. 2015. Vol. 121, No. 13. P. 15–19.
6. Halachev M., Georgiev G. Study of different types coders for GSM. Proceedings of Technical University of Sofia. 2014. Vol. 64, No. 2. P. 9–14.
7. Katugampala N., Villette S., Kondoz A. M. RT E2E secure voice comm. over GSM voice channel. Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Philadelphia, PA, USA. 2005. P. 141–144.
8. Secure End-to-End Voice Communication: A Comprehensive Review of Steganography, Modem-Based Cryptography, and Chaotic Cryptography Techniques. IEEE Access. 2024. URL: https://ieeexplore.ieee.org/ (дата звернення: 14.05.2024).
9. Li Y., et al. Secure speech content based on scrambling and adaptive hiding. Multimedia Tools and Applications. 2018. Vol. 77. P. 12345–12360.

10. Hybrid ciphering and frequency domain scrambling for secure speech transmission through MIMO-OFDM system. Journal of Communications. 2022. Vol. 17, No. 5. P. 340–348.
11. Обеспечение безопасности передачи информации в радиотехнических системах с примерами в проектах LabVIEW+. Навчальний посібник. Київ, 2020. 150 с.
12. ADSP-218x DSP Hardware Reference. Norwood, MA : Analog Devices Inc., 2001. 374 p.
13. Oppenheim A. V., Schafer R. W. Discrete-Time Signal Processing. 3rd ed. Pearson, 2009. 1120 p.
14. Schneier B. Applied Cryptography: Protocols, Algorithms, and Source Code in C. 20th Anniversary Edition. Wiley, 2015. 784 p.
15. MATLAB Documentation. MathWorks. URL: https://www.mathworks.com/help/matlab/ (дата звернення: 20.05.2024).
16. ETSI TS 146 090 V17.0.0. Digital cellular telecommunications system (Phase 2+) (GSM); Adaptive Multi-Rate (AMR) speech transcoding. ETSI, 2022. 54 p.
17. Kuo S. M., Lee B. H., Tian W. Real-Time Digital Signal Processing: Implementations and Applications. 2nd ed. John Wiley & Sons, 2006. 648 p.
18. Rabiner L. R., Schafer R. W. Theory and Applications of Digital Speech Processing. Pearson, 2010. 1056 p.
19. Proakis J. G., Manolakis D. G. Digital Signal Processing: Principles, Algorithms, and Applications. 4th ed. Pearson, 2006. 1104 p.
20. Smith S. W. The Scientist and Engineer's Guide to Digital Signal Processing. California Technical Publishing, 1997. 650 p.

Додаток А – Методичні вказівки до лабораторної роботи №1

Лабораторна робота №1 Тема: Дослідження алгоритмів одномірного та двовимірного скремблювання аудіосигналів у середовищі MATLAB.

1. Мета роботи та структура проекту

Мета роботи полягає у дослідженні процесів одномірного (часового і частотного) та двовимірного (каскадного) скремблювання аудіосигналів. Завдання включає ознайомлення з принципами сегментації та перестановки елементів матриці сигналу, а також практичну оцінку впливу розмірностей матриць на рівень приховування інформації.

Робоча директорія лабораторного макета має чітку ієрархію та містить головний виконуваний скрипт і систему вкладених папок для організації вхідних та вихідних даних:

- main.m — головний скрипт, з якого запускається повний цикл дослідження.
- Папка algorithms/ — містить усі обчислювальні модулі системи:
  - time_scramble.m / time_descramble.m — функції прямого та зворотного часового перетворення.
  - freq_scramble.m / freq_descramble.m — функції прямого та зворотного частотного перетворення.
  - get_inverse_key.m — математичний модуль генерації зворотного індексного масиву для розшифрування.
  - analyze_signals.m — модуль побудови осцилограм та спектрограм.
- Папка audio_input/ — директорія для розміщення користувацьких тестових аудіофайлів у форматі .wav перед початком обробки.
- Папка audio_output/ — генерується автоматично системою для збереження зашифрованих та розшифрованих аудіофайлів.
- Папка plots_output/ — генерується автоматично для збереження графічних результатів візуалізації у форматі .png.

2. Підготовка до роботи та налаштування параметрів варіанта

Перед початком моделювання необхідно налаштувати головний скрипт main.m відповідно до параметрів індивідуального завдання (див. таблицю варіантів).

Вибір тестового сигналу. У блоці ініціалізації скрипта значення змінної signal_choice визначає тип вхідного сигналу для дослідження:

- 1 — лінійно-частотно-модульований сигнал (чірп), призначений для базового аналізу геометричних розривів;
- 2 — тональний сигнал із синусоїдальною модуляцією частоти, застосовується для дослідження ефектів на складніших частотних траєкторіях;
- 3 — власний мовний аудіофайл. Для його використання необхідно попередньо розмістити файл формату .wav у папці audio_input/ та вказати його точну назву у змінній filename.

Налаштування розмірностей перетворень. Рівень фрагментації (сегментації) мовного сигналу задається двома основними константами:

- num_chunks — кількість фрагментів, на які розбивається аудіосигнал у часовій області;
- num_bands — кількість підсмуг, на які ділиться корисний частотний спектр (у межах від 300 до 3400 Гц) під час спектрального скремблювання.

Ініціалізація генератора ключів. Для забезпечення унікальності та відтворюваності результатів, генерація векторів перестановок і масивів інверсій здійснюється детерміновано. Для налаштування системи на конкретне завдання перед функціями генерації ключів (у блоці Налаштування параметрів скремблера) необхідно застосувати функцію ініціалізації rng(V), де V — номер індивідуального варіанта студента. Це гарантує формування стабільного псевдовипадкового коду для прямого та зворотного перетворень під час виконання роботи.

Варіант	Тип тестового сигналу	Кількість часових фрагментів	Кількість частотних підсмуг
1	Чірп	4	4
2	Синусоїда	8	4
3	Чірп	4	8
4	Синусоїда	6	8
5	Чірп	8	6
6	Синусоїда	6	6

Таблиця А.1 – Варіанти завдань

3. Порядок виконання основного дослідження

1. Відкрийте головний скрипт main.m у середовищі MATLAB. У відповідних блоках коду встановіть значення змінної signal_choice, параметри num_chunks та num_bands, а також ініціалізатор rng згідно з індивідуальним варіантом.

2. Запустіть скрипт на виконання. Програма автоматично виконає три послідовні етапи обробки (часове, частотне та двовимірне скремблювання) і збереже результати у вкладені директорії проекту.
3. Відкрийте папку plots_output/ та проаналізуйте графіки одномірного часового скремблювання. Зафіксуйте зміну порядку слідування сегментів на осі часу та наявність дзеркально відображених (інвертованих) блоків. Перевірте збереження вихідного частотного діапазону перетвореного сигналу.
4. Проаналізуйте спектрограму одномірного частотного скремблювання. Відстежте координатні перенесення спектральних підсмуг та локальні частотні інверсії. Зафіксуйте наявність фазових розривів на межах стикування зміщених блоків, що візуалізуються як шумові артефакти.
5. Проаналізуйте результати каскадного двовимірного (2D) скремблювання. Оцініть загальний рівень деформації початкової структури тестового сигналу та формування хаотичної двовимірної мозаїки.
6. Змініть значення signal_choice на 3 для завантаження реального мовного сигналу (наприклад, файл speech.wav). Повторно запустіть скрипт.
7. Відкрийте папку audio_output/ та проведіть суб'єктивну акустичну оцінку згенерованих аудіофайлів. Визначте рівень залишкової розбірливості мови після застосування кожного з методів кодування та оцініть акустичний вплив залишкових спотворень у відновленому (дескрембльованому) сигналі.

4. Контрольні запитання

1. У чому полягає математичний принцип виконання часових та частотних перестановок під час скремблювання мовного сигналу?
2. Як саме зміна кількості часових фрагментів (num_chunks) та частотних підсмуг (num_bands) впливає на обсяг простору ключів і рівень приховування інформації?
3. З якою метою у лабораторному макеті використовується детермінована ініціалізація псевдовипадкових чисел (функція rng) замість абсолютно випадкової генерації векторів?
4. Які фізичні причини викликають появу шумових артефактів (фазових розривів та ефекту просочування спектра) на межах стикування спектральних блоків під час частотного скремблювання?
5. Чому під час каскадного двовимірного дескремблювання алгоритми зворотного перетворення застосовуються у порядку, зворотному до прямого кодування (LIFO)?
6. Як працює алгоритм зняття внутрішньої інверсії часових сегментів та спектральних підсмуг на приймальній стороні без використання додаткових математичних обчислень?

Додаток Б – Методичні вказівки до лабораторної роботи №2

Лабораторна робота №2 Тема: Апаратна реалізація методів скремблювання на сигнальному процесорі ADSP-2181 у середовищі Visual DSP++.

1. Мета роботи та методика дослідження

Мета роботи полягає у практичному дослідженні впливу апаратних обмежень 16-бітного сигнального процесора з фіксованою точкою на якість обробки аудіосигналів під час виконання алгоритмів двовимірного скремблювання.

Дослідження базується на концепції апаратного моделювання в контурі (Hardware-in-the-Loop). Оскільки пряме введення аудіофайлів у симулятор процесора неможливе, методика передбачає попередню генерацію та квантування тестових сигналів у середовищі MATLAB, їхню потактову обробку цільовим ANSI-C кодом у симуляторі VisualDSP++ та подальше повернення отриманих дампів пам'яті в MATLAB для візуального аналізу й розрахунку компенсаційних коефіцієнтів.

2. Підготовка до роботи

Етап підготовки виконується у середовищі MATLAB і полягає у формуванні масиву вхідних даних. Оскільки симулятор Visual DSP++ вимагає даних у 16-бітному цілочисельному форматі, згенерований сигнал необхідно попередньо адаптувати.

У скрипті generate_stream.m задайте характеристики сигналу згідно з варіантом:

- **signal_type** — тип сигналу (1 — лінійний чірп, 2 — нестаціонарний гармонійний сигнал);
- **f_start** та **f_end** — межі частотного діапазону або параметри базової частоти та девіації (у герцах).

Змінну **amp_coeff** (коефіцієнт амплітуди) на початковому етапі встановіть рівною 1.0. Під час виконання лабораторної роботи цей параметр буде змінюватися для дослідження впливу амплітуди на виникнення шумів квантування та кліпінгу.

Запустіть скрипт. Алгоритм розрахує сигнал, відмасштабує його на максимальне значення розрядної сітки (32767) та виконає перетворення у формат int16.

Переконайтеся, що у робочій директорії створено файл input_stream.dat. Він повинен містити один стовпець цілих чисел у діапазоні від -32768 до 32767. Цей масив готовий до імпорту в середовище Visual DSP++.

Варіант	Тип сигналу	Частоти: початкова / кінцева (Гц)
1	Чірп	100 / 2000
2	Синусоїда	1000 / 500
3	Чірп	500 / 3000
4	Синусоїда	1500 / 800
5	Чірп	300 / 3400
6	Синусоїда	800 / 300

Таблиця Б.1 – Варіанти завдань для апаратного моделювання

3. Порядок виконання роботи

1. Відкрийте середовище Visual DSP++. Завантажте проект лабораторної роботи та додайте файл main.c (через меню **Project → Add to Project → Files**).
2. Виконайте компіляцію проекту (клавіша **F7**). Переконайтеся у відсутності помилок у вікні повідомлень.
3. Відкрийте меню **Settings → Streams** для налаштування портів вводу-виводу.
4. **Налаштування вхідного потоку:** Натисніть **Add**.
    - У блоці *Source* оберіть *File*, вкажіть шлях до згенерованого input_stream.dat та оберіть формат *Signed Integer*.
    - У блоці *Destination* оберіть *Debug target*, встановіть *Device* як *Data Memory I/O Port*, а в полі *Address* вкажіть ім'я змінної stream_in. Натисніть **OK**.

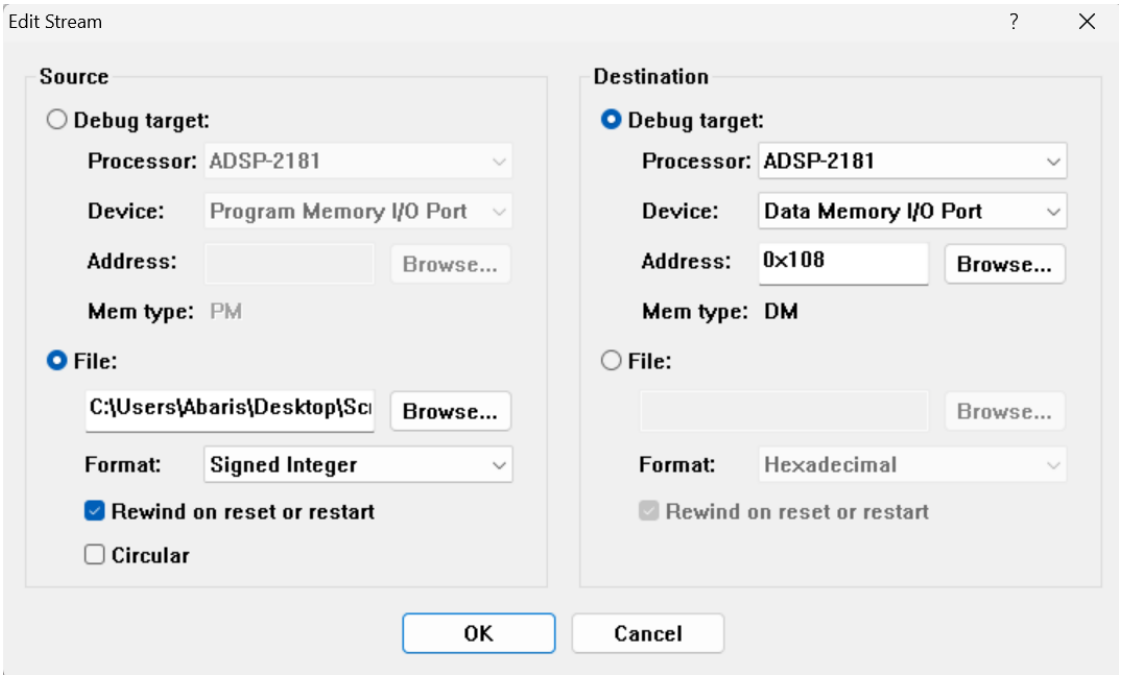

Рисунок Б.1 – Налаштування вхідного потоку

5. **Налаштування вихідних потоків:** Натисніть **Add** для створення ще двох потоків.
   - Для зашифрованого сигналу: у блоці *Source* оберіть *Debug target* та вкажіть змінну stream_out_scrambled. У блоці *Destination* оберіть *File*, вкажіть назву файлу scrambled_out.dat і формат *Signed Integer*.
   - Для відновленого сигналу: у блоці *Source* оберіть *Debug target* та вкажіть змінну stream_out_descrambled. У блоці *Destination* оберіть *File*, вкажіть назву файлу descrambled_out.dat і формат *Signed Integer*.
6. Запустіть симуляцію (клавіша **F5**). Дочекайтеся автоматичної зупинки процесу після зчитування всього масиву даних з вхідного файлу.
7. Відкрийте середовище MATLAB та запустіть скрипт analyze_dsp.m. Зафіксуйте виведене значення коефіцієнта scale_factor. Проаналізуйте отримані спектрограми та осцилограми.
8. Поверніться до скрипта generate_stream.m. Змініть значення amp_coeff на 0.1 (для дослідження шумів квантування) та на 2.0 (для дослідження кліпінгу).
9. Для кожного нового значення amp_coeff згенеруйте файл input_stream.dat, повторно запустіть симуляцію у Visual DSP++ (крок 6) та проаналізуйте результати в MATLAB (крок 7). Порівняйте графіки.

4. Контрольні запитання

1. Яка мета попереднього квантування тестового сигналу у 16-бітний цілочисельний формат перед його завантаженням у симулятор Visual DSP++?
2. Які фізичні процеси всередині сигнального процесора зумовлюють необхідність застосування компенсаційного коефіцієнта scale_factor на етапі постобробки?
3. Як зменшення коефіцієнта вхідної амплітуди впливає на відношення сигнал/шум та рівень шумів квантування у відновленому аудіосигналі?
4. Які візуальні ознаки на осцилограмі та спектрограмі свідчать про переповнення розрядної сітки процесора (кліпінг)?
5. З якою метою масиви даних для обчислення ШПФ мовою ANSI-C супроводжуються директивою вирівнювання пам'яті #pragma align 256?

Додаток В – ПРОГРАМНИЙ КОД

Додаток В.1 – freq_descramble.m

```
function [descrambled_audio] = freq_descramble(audio_in, fs, num_bands, inv_perm_vector, reflect_vector)
    % FREQ_DESCRAMBLE Відновлення частотного спектра (з обмеженням 3.4 кГц)

    window_size = 1024;
    overlap = 512;
    orig_length = length(audio_in);

    % Пряме короткочасне перетворення Фур'є (STFT)
    [S, ~, ~] = stft(audio_in, fs, 'Window', hamming(window_size), ...
                     'OverlapLength', overlap, 'FFTLength', window_size, ...
                     'FrequencyRange', 'onesided');

    % Ініціалізуємо нулями, щоб відсікти можливі шуми поза нашою смугою
    S_restored_order = zeros(size(S));
    N_bins = size(S, 1);

    % Задаємо ті ж самі межі мовного каналу (у Гц)
    f_min = 300;
    f_max = 3400;

    % Переводимо частоти в індекси бінів FFT
    bin_min = max(2, round(f_min * window_size / fs) + 1);
    bin_max = min(N_bins - 1, round(f_max * window_size / fs) + 1);

    % Кількість робочих бінів у цільовій смузі
    usable_bins = bin_max - bin_min + 1;
    bins_per_band = floor(usable_bins / num_bands);

    % КРОК 1: Зворотна перестановка (повертаємо смуги на їхні початкові місця)
    for current_idx = 1:num_bands
        original_idx = inv_perm_vector(current_idx); % Де смуга знаходилася спочатку

        % Беремо з поточної позиції (яку ми отримали з каналу зв'язку)
        curr_start = bin_min + (current_idx - 1) * bins_per_band;
        curr_end = curr_start + bins_per_band - 1;

        % Кладемо в оригінальну позицию (де вона має бути для відновлення)
        orig_start = bin_min + (original_idx - 1) * bins_per_band;
        orig_end = orig_start + bins_per_band - 1;

        S_restored_order(orig_start:orig_end, :) = S(curr_start:curr_end, :);
    end

    S_descrambled = S_restored_order;
```

```
    % КРОК 2: Зняття частотної інверсії (повторний переворот)
    % Важливо: робимо це вже на відновлених оригінальних позиціях!
    for i = 1:num_bands
        if reflect_vector(i) == 1
            idx_start = bin_min + (i - 1) * bins_per_band;
            idx_end = idx_start + bins_per_band - 1;
            S_descrambled(idx_start:idx_end, :) = flipud(S_descrambled(idx_start:idx_end, :));
        end
    end

    % Зворотне перетворення Фур'є (ISTFT)
    descrambled_audio = istft(S_descrambled, fs, 'Window', hamming(window_size), ...
                              'OverlapLength', overlap, 'FFTLength', window_size, ...
                              'FrequencyRange', 'onesided');

    % Очищаємо від уявних артефактів
    descrambled_audio = real(descrambled_audio);

    % Динамічна фіксація довжини (повертаємо до оригінального розміру)
    if length(descrambled_audio) < orig_length
        % Доповнюємо вектор-стовпець нулями
        descrambled_audio = [descrambled_audio; zeros(orig_length - length(descrambled_audio), 1)];
    elseif length(descrambled_audio) > orig_length
        % Обрізаємо зайві відліки
        descrambled_audio = descrambled_audio(1:orig_length);
    end
end
```

Додаток В.2 – freq_scramble.m

```
function [scrambled_audio] = freq_scramble(audio_in, fs, num_bands, perm_vector, reflect_vector)
    % FREQ_SCRAMBLE Пряме частотне скреблювання

    window_size = 1024;
    overlap = 512;
    orig_length = length(audio_in);

    % Пряме короткочасне перетворення Фур'є (STFT)
    [S, f, t] = stft(audio_in, fs, 'Window', hamming(window_size), ...
                    'OverlapLength', overlap, 'FFTLength', window_size, ...
                    'FrequencyRange', 'onesided');

    % Створюємо порожню матрицю для результату (очищаємо все сміття за межами нашої смуги)
    S_scrambled = zeros(size(S));
    N_bins = size(S, 1);

    % Задаємо межі телефонного/мовного каналу (у Гц)
    f_min = 300;
    f_max = 3400;

    % Переводимо частоти в індекси бінів FFT
    bin_min = max(2, round(f_min * window_size / fs) + 1);
    bin_max = min(N_bins - 1, round(f_max * window_size / fs) + 1);

    % Кількість робочих бінів у цільовій смузі
    usable_bins = bin_max - bin_min + 1;
    bins_per_band = floor(usable_bins / num_bands);

    % Коригуємо верхню межу, щоб відкинути остачу від ділення
    bin_max = bin_min + bins_per_band * num_bands - 1;

    % Обробка кожної смуги
    for i = 1:num_bands
        % Визначаємо індекси вихідної смуги
        source_start = bin_min + (i - 1) * bins_per_band;
        source_end = source_start + bins_per_band - 1;
        band_block = S(source_start:source_end, :);

        % КРОК 1: Спектральна інверсія (віддзеркалення смуги)
        if reflect_vector(i) == 1
            band_block = flipud(band_block);
        end

        % КРОК 2: Перестановка в нову позицію згідно з вектором
        target_idx = perm_vector(i);
        target_start = bin_min + (target_idx - 1) * bins_per_band;
```

```
        target_end = target_start + bins_per_band - 1;

        S_scrambled(target_start:target_end, :) = band_block;
    end

    % Зворотне перетворення Фур'є (ISTFT)
    scrambled_audio = istft(S_scrambled, fs, 'Window', hamming(window_size), ...
                            'OverlapLength', overlap, 'FFTLength', window_size, ...
                            'FrequencyRange', 'onesided');

    % Відкидаємо уявну частину, залишаючи лише дійсний сигнал
    scrambled_audio = real(scrambled_audio);

    % Динамічна фіксація довжини (повертаємо до оригінального розміру)
    if length(scrambled_audio) < orig_length
        % Доповнюємо вектор-стовпець нулями
        scrambled_audio = [scrambled_audio; zeros(orig_length - length(scrambled_audio), 1)];
    elseif length(scrambled_audio) > orig_length
        % Обрізаємо зайві відліки
        scrambled_audio = scrambled_audio(1:orig_length);
    end
end
```

Додаток В.3 – get_inverse_key.m

```
function [inv_perm_vector] = get_inverse_key(perm_vector)
    % GET_INVERSE_KEY Обчислює вектор індексів для зворотної перестановки.
    % У криптографії це називається знаходженням зворотної підстановки.

    % Вхідні параметри:
    % perm_vector - Вихідний вектор перестановки (наприклад, [3, 1, 4, 2]).

    % Значення, що повертається:
    % inv_perm_vector - Вектор, який поверне елементи на початкові позиції (наприклад, [2, 4, 1,
3]).

    % Функція sort сортує масив за зростанням.
    % Синтаксис [~, inv_perm_vector] означає, що самі відсортовані значення
    % нам не потрібні (ми відкидаємо їх за допомогою тильди ~),
    % але ми зберігаємо другий аргумент — індекси вихідних елементів.
    % Ці індекси і є ідеальним математичним інверсним ключем.
    [~, inv_perm_vector] = sort(perm_vector);
end
```

Додаток В.4 – time_descramble.m

```
        function [descrambled_audio] = time_descramble(audio_in, num_chunks, inv_perm_vector,
reflect_vector)
            % TIME_DESCRAMBLE Відновлення часового сигналу

            if isrow(audio_in)
                audio_in = audio_in'; % Переводимо у вектор-стовпець
            end

            total_samples = length(audio_in);

            samples_per_chunk = ceil(total_samples / num_chunks);

            pad_length = (samples_per_chunk * num_chunks) - total_samples;

            % Доповнюємо нулями, якщо довжина сигналу не ділиться націло
            if pad_length > 0
                audio_in = [audio_in; zeros(pad_length, 1)];
            end

            % Тепер reshape отримає цілі числа і спрацює ідеально
            chunk_matrix = reshape(audio_in, samples_per_chunk, num_chunks);

            % КРОК 1: Зняття інверсії (СПОЧАТКУ знімаємо інверсію із зашифрованих блоків)
            for i = 1:num_chunks
                if reflect_vector(i) == 1
                    chunk_matrix(:, i) = flipud(chunk_matrix(:, i));
                end
            end

            % КРОК 2: Зворотна перестановка (повертаємо стовпці на вихідні позиції)
            restored_order_matrix = chunk_matrix(:, inv_perm_vector);

            % Розгортаємо матрицю назад в одновимірний масив
            descrambled_audio = restored_order_matrix(:);
        end
```

Додаток В.5 – time_scramble.m

```
function [scrambled_audio] = time_scramble(audio_in, num_chunks, perm_vector, reflect_vector)
    % TIME_SCRAMBLE Пряме часове скремблювання

    % Переводимо у вектор-стовпець, якщо на вхід подано рядок
    if isrow(audio_in)
        audio_in = audio_in';
    end

    total_samples = length(audio_in);
    samples_per_chunk = ceil(total_samples / num_chunks);
    pad_length = (samples_per_chunk * num_chunks) - total_samples;

    % Доповнюємо нулями, якщо довжина сигналу не ділиться націло на кількість фрагментів
    if pad_length > 0
        audio_in = [audio_in; zeros(pad_length, 1)];
    end

    % Перетворюємо на матрицю, де кожен стовпець — це окремий фрагмент часу
    chunk_matrix = reshape(audio_in, samples_per_chunk, num_chunks);

    % КРОК 1: Перестановка фрагментів місцями згідно з вектором perm_vector
    scrambled_matrix = chunk_matrix(:, perm_vector);

    % КРОК 2: Інверсія (віддзеркалення) всередині вже переставлених фрагментів
    for i = 1:num_chunks
        if reflect_vector(i) == 1
            % Перевертаємо стовпці у матриці scrambled_matrix
            scrambled_matrix(:, i) = flipud(scrambled_matrix(:, i));
        end
    end

    % Розгортаємо матрицю назад в одновимірний масив
    scrambled_audio = scrambled_matrix(:);
end
```

Додаток В.6 – main.m

```
clear; clc; close all;

% 1. Зчитування еталонного тексту (Оригінал)
ref_text = fileread('0_original.txt');

% 2. Налаштування назв файлів з транскрипціями для кожного алгоритму
% Формат: {Скрембльований, Скрембльований+MR475, Дескрембльований, Дескрембльований+MR475}
files_time = {'1_scrambled_time.txt', '1_scrambled_time_MR475.txt', ...
              '2_descrambled_time.txt', '2_descrambled_time_MR475.txt'};

files_freq = {'3_scrambled_freq.txt', '3_scrambled_freq_MR475.txt', ...
              '4_descrambled_freq.txt', '4_descrambled_freq_MR475.txt'};

files_2d   = {'5_scrambled_2d.txt', '5_scrambled_2d_MR475.txt', ...
              '6_descrambled_2d.txt', '6_descrambled_2d_MR475.txt'};

all_files = {files_time, files_freq, files_2d};
num_methods = length(all_files);
num_states = 4;

% Матриця для зберігання результатів WRR
wrr_results = zeros(num_methods, num_states);

% 3. Цикл обробки файлів та розрахунку WRR
for i = 1:num_methods
    current_method_files = all_files{i};
    for j = 1:num_states
        file_name = current_method_files{j};
        if exist(file_name, 'file')
            test_text = fileread(file_name);
            wrr_results(i, j) = calculate_wrr(ref_text, test_text);
        else
            warning('Файл %s не знайдено. Встановлено WRR = 0.', file_name);
            wrr_results(i, j) = 0;
        end
    end
end

% Додатково порахуємо вплив MR475 на чистий оригінал (для довідки)
if exist('0_original_MR475.txt', 'file')
    orig_mr475_text = fileread('0_original_MR475.txt');
    orig_mr475_wrr = calculate_wrr(ref_text, orig_mr475_text);
    fprintf('Оригінал після кодека MR475 (Базова деградація): %.2f%%\n', orig_mr475_wrr);
end

% 4. Візуалізація результатів (Побудова гістограми)
figure('Name', 'Аналіз стійкості до ASR', 'Color', 'w', 'Position', [100, 100, 950, 550]);
```

```
% Побудова згрупованої гістограми
b = bar(wrr_results, 'grouped');

% Налаштування кольорів (стильні та контрастні)
b(1).FaceColor = [0.8500 0.3250 0.0980]; % Помаранчево-червоний
b(2).FaceColor = [0.6350 0.0780 0.1840]; % Темно-бордовий
b(3).FaceColor = [0.4660 0.6740 0.1880]; % Світло-зелений
b(4).FaceColor = [0.0000 0.4470 0.7410]; % Класичний синій

% Оформлення осей
set(gca, 'XTickLabel', {'Часовий метод', 'Частотний метод', 'Двовимірний (2D) каскад'}, 'FontSize',
12);
ylabel('Відсоток розпізнаних слів (WRR), %', 'FontSize', 12, 'FontWeight', 'bold');
title('Ефективність маскування та стійкість до транзитного кодека MR475', 'FontSize', 14);
ylim([0 110]);
grid on;

% Легенда
legend({'Скрембльований (Прямий ASR)', 'Скрембльований (MR475 + ASR)', ...
        'Дескрембльований (Прямий ASR)', 'Дескрембльований (MR475 + ASR)'}, ...
        'Location', 'northwest', 'FontSize', 10);

% Додавання значень над стовпчиками
for i = 1:num_states
    xtips = b(i).XEndPoints;
    ytips = b(i).YEndPoints;
    labels = string(round(b(i).YData, 1)) + "%";
    text(xtips, ytips, labels, 'HorizontalAlignment', 'center', ...
         'VerticalAlignment', 'bottom', 'FontSize', 9);
end

% ========================================================================
% Допоміжна функція розрахунку WRR (Word Recognition Rate)
% ========================================================================
function wrr = calculate_wrr(reference_text, asr_text)
    % Чистка від розділових знаків та переведення в нижній регістр
    ref_clean = lower(erasePunctuation(reference_text));
    asr_clean = lower(erasePunctuation(asr_text));

    % Розбиття на масиви слів
    ref_words = split(ref_clean);
    asr_words = split(asr_clean);

    % Видалення порожніх елементів
    ref_words(cellfun('isempty', ref_words)) = [];
    asr_words(cellfun('isempty', asr_words)) = [];

    % Відстань Левенштейна
    num_errors = editDistance(ref_words, asr_words);
    num_total_words = length(ref_words);
```

```
    if num_total_words > 0
        wrr = max(0, (1 - (num_errors / num_total_words))) * 100;
    else
        wrr = 0;
    end
end
```

Додаток В.7 – analyze_signals.m

```
function analyze_signals(audio_data, fs, figure_title, num_chunks, num_bands, plots_dir)
    % ANALYZE_SIGNALS Будує та зберігає графіки

    fig = figure('Name', figure_title, 'Position', [100, 100, 1000, 700], 'Color', 'w');

    % --- ОСЦИЛОГРАМА ---
    subplot(2, 1, 1);
    total_samples = length(audio_data);
    total_time = total_samples / fs;
    t = (0:total_samples-1) / fs;

    plot(t, audio_data, 'b', 'LineWidth', 0.1);

    title([figure_title, ' - Осцилограма'], 'FontWeight', 'bold');
    xlabel('Час (с)');
    ylabel('Амплітуда');
    xlim([0, total_time]);
    ylim([-1.1, 1.1]);

    % Малюємо вертикальні лінії розбиття на часові фрагменти
    if num_chunks > 1
        hold on;
        chunk_duration = total_time / num_chunks;
        for i = 1:(num_chunks - 1)
            xline(i * chunk_duration, 'r--', 'LineWidth', 1.5, 'Alpha', 0.8);
        end
        hold off;
    end

    % --- СПЕКТРОГРАМА ---
    subplot(2, 1, 2);
    window = hamming(512);
    noverlap = 256;
    nfft = 1024;

    spectrogram(audio_data, window, noverlap, nfft, fs, 'yaxis');
    title([figure_title, ' - Спектрограма'], 'FontWeight', 'bold');
    colormap('jet');
    clim([-100, -30]);

    % НАБЛИЖАЄМО ГРАФІК: Обмежуємо вісь Y до 4 кГц
    ylim([0, 4]);

    if num_chunks > 1 || num_bands > 1
        hold on;

        % Вертикальні лінії для часових сегментів
```

```
        if num_chunks > 1
            chunk_duration = total_time / num_chunks;
            for i = 1:(num_chunks - 1)
                xline(i * chunk_duration, 'k--', 'LineWidth', 2, 'Alpha', 0.9);
            end
        end

        % Горизонтальні лінії для частотних смуг
        if num_bands > 1
            f_min_khz = 0.3;  % 300 Гц у кГц
            f_max_khz = 3.4;  % 3400 Гц у кГц
            band_height_khz = (f_max_khz - f_min_khz) / num_bands;

            % Для наочності виділяємо червоним кольором загальні межі скремблювання
            yline(f_min_khz, 'r-', 'LineWidth', 1.5, 'Alpha', 0.5);
            yline(f_max_khz, 'r-', 'LineWidth', 1.5, 'Alpha', 0.5);

            % Малюємо самі розділювачі смуг
            for i = 1:(num_bands - 1)
                yline(f_min_khz + i * band_height_khz, 'k--', 'LineWidth', 1.5, 'Alpha', 0.9);
            end
        end
        hold off;
    end

    % Автоматичне збереження в папку
    if nargin >= 6 && ~isempty(plots_dir)
        % Замінюємо пробіли та дужки для безпечного імені файлу
        safe_name = strrep(figure_title, ' ', '_');
        safe_name = strrep(safe_name, '.', '');
        safe_name = strrep(safe_name, '(', '');
        safe_name = strrep(safe_name, ')', '');
        saveas(fig, fullfile(plots_dir, [safe_name, '.png']));
    end
end
```

Додаток В.8 – speech_test.m

```
clear; clc; close all;

% 1. Зчитування еталонного тексту (Оригінал) з примусовим UTF-8
ref_text = read_utf8('transcript/0_original.txt');

% 2. Налаштування назв файлів з транскрипціями для кожного алгоритму
files_time = {'transcript/1_scrambled_time.txt', 'transcript/1_scrambled_time_MR475.txt', ...
              'transcript/2_descrambled_time.txt', 'transcript/2_descrambled_time_MR475.txt'};

files_freq = {'transcript/3_scrambled_freq.txt', 'transcript/3_scrambled_freq_MR475.txt', ...
              'transcript/4_descrambled_freq.txt', 'transcript/4_descrambled_freq_MR475.txt'};

files_2d   = {'transcript/5_scrambled_2d.txt', 'transcript/5_scrambled_2d_MR475.txt', ...
              'transcript/6_descrambled_2d.txt', 'transcript/6_descrambled_2d_MR475.txt'};

all_files = {files_time, files_freq, files_2d};
num_methods = length(all_files);
num_states = 4;

% Матриця для зберігання результатів WRR
wrr_results = zeros(num_methods, num_states);

% 3. Цикл обробки файлів та розрахунку WRR
for i = 1:num_methods
    current_method_files = all_files{i};
    for j = 1:num_states
        file_name = current_method_files{j};
        if exist(file_name, 'file')
            test_text = read_utf8(file_name);
            fprintf('\n--- Аналіз файлу: %s ---\n', file_name);
            wrr_results(i, j) = calculate_wrr(ref_text, test_text);
        else
            warning('Файл %s не знайдено. Встановлено WRR = 0.', file_name);
            wrr_results(i, j) = 0;
        end
    end
end

% 4. Візуалізація результатів
figure('Name', 'Аналіз стійкості до ASR', 'Color', 'w', 'Position', [100, 100, 950, 550]);
b = bar(wrr_results, 'grouped');

b(1).FaceColor = "red";
b(2).FaceColor = "magenta";
b(3).FaceColor = "blue";
b(4).FaceColor = "cyan" ;
```

```
        set(gca, 'XTickLabel', {'Часовий метод', 'Частотний метод', 'Двовимірний (2D) каскад'}, 'FontSize',
12);
        ylabel('Відсоток розпізнаних слів (WRR), %', 'FontSize', 12, 'FontWeight', 'bold');
        title('Ефективність маскування та стійкість до транзитного кодека MR475', 'FontSize', 14);
        ylim([0 110]);
        grid on;

        legend({'Скрембльований (Прямий ASR)', 'Скрембльований (MR475 + ASR)', ...
                'Дескрембльований (Прямий ASR)', 'Дескрембльований (MR475 + ASR)'}, ...
                'Location', 'northwest', 'FontSize', 10);

        for i = 1:num_states
            xtips = b(i).XEndPoints;
            ytips = b(i).YEndPoints;
            labels = string(round(b(i).YData, 1)) + "%";
            text(xtips, ytips, labels, 'HorizontalAlignment', 'center', ...
                 'VerticalAlignment', 'bottom', 'FontSize', 9);
        end

        % ДОПОМІЖНА ФУНКЦІЯ: Правильне читання UTF-8 файлів
        function textData = read_utf8(fname)
            fid = fopen(fname, 'r', 'n', 'UTF-8');
            if fid == -1
                textData = '';
                return;
            end
            textData = fread(fid, '*char')';
            fclose(fid);
        end


        function wrr = calculate_wrr(reference_text, asr_text)
            if isempty(strtrim(asr_text))
                fprintf('Файл порожній або ASR нічого не розпізнав.\n');
                wrr = 0;
                return;
            end

            % видаляємо ТІЛЬКИ знаки пунктуації, не чіпаючи букви
            ref_clean = regexprep(lower(reference_text), '[.,!?;:''"()\[\]{}\-–_`’]', '');
            asr_clean = regexprep(lower(asr_text), '[.,!?;:''"()\[\]{}\-–_`’]', '');

            % Розбиваємо на слова
            ref_words = cellstr(split(strtrim(ref_clean)));
            asr_words = cellstr(split(strtrim(asr_clean)));

            % Видаляємо порожні комірки
            ref_words(cellfun('isempty', ref_words)) = [];
            asr_words(cellfun('isempty', asr_words)) = [];

            num_total_words = length(ref_words);
```

```
    if num_total_words == 0
        wrr = 0;
        return;
    end

    matches = 0;
    unmatched_words = {};

    % Шукаємо слова
    for i = 1:num_total_words
        idx = find(strcmp(asr_words, ref_words{i}), 1);
        if ~isempty(idx)
            matches = matches + 1;
            asr_words(idx) = []; % Видаляємо знайдене
        else
            unmatched_words{end+1} = ref_words{i};
        end
    end

    % Вивід у консоль
    fprintf('Оригінал слів: %d | Знайдено збігів: %d\n', num_total_words, matches);

    wrr = (matches / num_total_words) * 100;
end
```

Додаток В.9 – generate_stream.m

```
clear; clc; close all;

% --- НАЛАШТУВАННЯ ПАРАМЕТРІВ ВАРІАНТА ---
fs = 8000;              % Частота дискретизації (8 кГц - стандарт для голосу)
total_samples = 2048;   % Генеруємо 8 фреймів по 256 відліків
t = (0:total_samples-1)' / fs;

signal_type = 2;      % 1 - Лінійний чірп, 2 - Нестаціонарний гармонійний (FM)
amp_coeff = 2.0;      % Коефіцієнт амплітуди
f_start = 100;        % Початкова частота / Базова частота (Гц)
f_end = 2000;         % Кінцева частота / Девіація (Гц)

% --- ГЕНЕРАЦІЯ ТЕСТОВОГО СИГНАЛУ ---
if signal_type == 1
    % Лінійно-частотно-модульований сигнал (Chirp)
    original_audio = chirp(t, f_start, t(end), f_end);
elseif signal_type == 2
    % Нестаціонарний гармонійний сигнал (FM)
    % Використовуємо f_start як частоту носійної, f_end як девіацію
    original_audio = sin(2*pi*f_start*t - (f_end/2)*cos(2*pi*2*t));
else
    error('Невідомий тип сигналу. Встановіть signal_type = 1 або 2.');
end

% Застосування коефіцієнта амплітуди перед квантуванням
original_audio = original_audio * amp_coeff;

% Імітація жорсткого кліпінгу віртуального АЦП
% Якщо сигнал перевищує допустимі межі, він зрізається
original_audio(original_audio > 1.0) = 1.0;
original_audio(original_audio < -1.0) = -1.0;

% Квантування у 16-бітний формат для цифрового сигнального процесора (DSP)
% Множимо на 32767, щоб перевести дроби [-1; 1] у цілі числа [-32768; 32767]
original_int16 = int16(original_audio * 32767);

% --- ЕКСПОРТ ДАНИХ ДЛЯ VISUAL DSP++ ---
filename = 'input_stream.dat';
fid = fopen(filename, 'w');

if fid == -1
    error('Не вдалося створити файл. Перевірте права доступу до робочої директорії.');
end

% Записуємо по одному числу в рядок (у десятковому форматі)
for i = 1:length(original_int16)
    fprintf(fid, '%d\n', original_int16(i));
```

```
end
fclose(fid);

disp(['Файл ', filename, ' успішно згенеровано!']);
disp(['Тип сигналу: ', num2str(signal_type), ', Коефіцієнт амплітуди: ', num2str(amp_coeff)]);
disp('Тепер підключіть цей файл до вхідного порту Input Stream у Visual DSP++.');
```

Додаток В.10 – analyze_dsp.m

```
clear; clc; close all;

% --- НАЛАШТУВАННЯ ПАРАМЕТРІВ ---
fs = 8000;
% Беремо 8 фрейми поспіль (1024 відліки), щоб на графіку було видно сходинки та переходи
frames_to_show = 8;
frame_size = 256 * frames_to_show;
t_frame = (0:frame_size-1)' / fs;

%% 1. ЗАВАНТАЖЕННЯ ДАНИХ З VISUAL DSP++
try
    raw_input = load('input_stream.dat');
    raw_scrambled = load('scrambled_out.dat');
    raw_descrambled = load('descrambled_out.dat');
catch
    error('Помилка: Файли не знайдено!');
end

original_frame = double(raw_input(1:frame_size));
scrambled_frame = double(raw_scrambled(1:frame_size));
descrambled_frame = double(raw_descrambled(1:frame_size));

%% 2. НОРМАЛІЗАЦІЯ (Компенсація масштабування ШПФ)
% Знаходимо максимальну амплітуду оригінального сигналу
max_orig = max(abs(original_frame));
% Визначаємо коефіцієнт приглушення сигналу після обробки в DSP
scale_factor = max_orig / max(abs(descrambled_frame));
% Повертаємо вихідним сигналам їхню початкову гучність (масштабуємо)
descrambled_frame = descrambled_frame * scale_factor;
scrambled_frame = scrambled_frame * scale_factor;

%% 3. ВІЗУАЛІЗАЦІЯ РЕЗУЛЬТАТІВ
% Створюємо велике вікно на білому фоні для гарного експорту у звіт
figure('Name', 'Аналіз 2D-скреблювання на ADSP-2181', 'Position', [100, 100, 1000, 800], 'Color',
'w');

% Налаштування параметрів спектрограми для високої деталізації
win = hamming(128);
noverlap = 120;
nfft = 512;

% --- ОРИГІНАЛ ---
subplot(3, 2, 1);
plot(t_frame, original_frame, 'b');
title('Оригінал: Осцилограма', 'FontWeight', 'bold');
xlabel('Час (с)'); ylabel('Амплітуда'); grid on;
```

```
subplot(3, 2, 2);
spectrogram(original_frame, win, noverlap, nfft, fs, 'yaxis');
title('Оригінал: Спектрограма', 'FontWeight', 'bold');
colormap('jet');

% --- СКРЕМБЛЬОВАНИЙ (Після ШПФ та перестановки) ---
subplot(3, 2, 3);
plot(t_frame, scrambled_frame, 'm');
title('2D-Шифр: Осцилограма', 'FontWeight', 'bold');
xlabel('Час (с)'); ylabel('Амплітуда'); grid on;

subplot(3, 2, 4);
spectrogram(scrambled_frame, win, noverlap, nfft, fs, 'yaxis');
title('2D-Шифр: Спектрограма', 'FontWeight', 'bold');

% --- ДЕСКРЕМБЛЬОВАНИЙ (Відновлений) ---
subplot(3, 2, 5);
plot(t_frame, descrambled_frame, 'g');
title('Відновлений: Осцилограма', 'FontWeight', 'bold');
xlabel('Час (с)'); ylabel('Амплітуда'); grid on;

subplot(3, 2, 6);
spectrogram(descrambled_frame, win, noverlap, nfft, fs, 'yaxis');
title('Відновлений: Спектрограма', 'FontWeight', 'bold');


disp(['Коефіцієнт компенсації амплітуди (scale_factor): ', num2str(scale_factor)]);
```

Додаток В.11 – main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <ffts.h>

// --- НАЛАШТУВАННЯ СИСТЕМИ ---
#define FRAME_SIZE 256
#define NUM_CHUNKS 4
#define NUM_BANDS 4
#define BINS_PER_BAND 31

// --- ЗМІННІ ДЛЯ ПОТОКІВ (Streams) ---
volatile int stream_in;
volatile int stream_out_scrambled;   // Для вивантаження зашифрованих даних (каші)
volatile int stream_out_descrambled; // Для вивантаження чистого відновленого звуку

// --- ГЛОБАЛЬНІ БУФЕРИ АУДІО ---
short rx_buffer[FRAME_SIZE];
short temp_time[FRAME_SIZE];
short temp_freq[FRAME_SIZE];
short temp_desc[FRAME_SIZE];
short tx_buffer[FRAME_SIZE];

// ========================================================================
// ВАЖЛИВО: ГАРАНТУЄМО КРУГОВІ МЕЖІ ДЛЯ АПАРАТНОГО ШПФ
// ========================================================================
// Процесори сімейства ADSP вимагають жорсткого вирівнювання пам'яті
// для швидких операцій зворотної адресації (bit-reversal) у ШПФ.
#pragma align 256
int in_real[FRAME_SIZE];
#pragma align 256
int in_imag[FRAME_SIZE];
#pragma align 256
int fft_real[FRAME_SIZE];
#pragma align 256
int fft_imag[FRAME_SIZE];

// --- КЛЮЧИ ШИФРУВАННЯ ---
// Формат: число = звідки беремо фрагмент, знак мінус (-) = інвертуємо його
int time_key[NUM_CHUNKS] = {1, 4, -3, 2};
int freq_key[NUM_BANDS]  = {3, -1, 4, 2};

int time_inv_key[NUM_CHUNKS];
int freq_inv_key[NUM_BANDS];

// --- ГЕНЕРАЦІЯ ЗВОРОТНОГО КЛЮЧА ---
void get_inverse_key_c(int* original_key, int* inverse_key, int num_elements) {
    int i, j;
```

```
    for (i = 0; i < num_elements; i++) {
        int target = i + 1;
        for (j = 0; j < num_elements; j++) {
            if (abs(original_key[j]) == target) {
                // Якщо в оригіналі була інверсія, у зворотному ключі вона теж має бути
                inverse_key[i] = (original_key[j] < 0) ? -(j + 1) : (j + 1);
                break;
            }
        }
    }
}

// --- ЧАСОВЕ СКРЕМБЛЮВАННЯ ---
void time_scramble_c(short* input, short* output, int* key, int num_samples, int num_chunks) {
    int chunk_size = num_samples / num_chunks;
    int i, j;
    for (i = 0; i < num_chunks; i++) {
        int current_key = key[i];
        int source_idx = abs(current_key) - 1;
        int is_inverted = (current_key < 0) ? 1 : 0;

        short* src_ptr = &input[source_idx * chunk_size];
        short* dst_ptr = &output[i * chunk_size];

        for (j = 0; j < chunk_size; j++) {
            // Копіюємо або задом наперед (інверсія), або прямо
            dst_ptr[j] = is_inverted ? src_ptr[chunk_size - 1 - j] : src_ptr[j];
        }
    }
}

// --- ЧАСТОТНЕ СКРЕМБЛЮВАННЯ ---
void freq_scramble_c(short* input, short* output, int* key, int num_samples, int num_bands) {
    int i, j, k;
    int exp_fft, exp_ifft, total_exp, shift;

    // 1. Підготовка (переведення 16-бітних відліків у 32-бітні для ШПФ)
    for (i = 0; i < FRAME_SIZE; i++) {
        in_real[i] = (int)input[i];
        in_imag[i] = 0;
    }

    // 2. Пряме ШПФ
    exp_fft = fft256(in_real, in_imag, fft_real, fft_imag);

    // Очищаємо буфери для збірки нового спектра
    for (i = 0; i < FRAME_SIZE; i++) {
        in_real[i] = 0;
        in_imag[i] = 0;
    }
```

```
// Зберігаємо краї (DC-компоненту та частоту Найквіста)
in_real[0] = fft_real[0]; in_imag[0] = fft_imag[0];
for (i = 125; i <= 128; i++) {
    in_real[i] = fft_real[i];
    in_imag[i] = fft_imag[i];
}

// 3. ПЕРЕСТАНОВКА СМУГ (Scrambling)
for (i = 0; i < num_bands; i++) {
    int current_key = key[i];
    int source_band_idx = abs(current_key) - 1;
    int is_inverted = (current_key < 0) ? 1 : 0;

    int src_start = 1 + source_band_idx * BINS_PER_BAND;
    int dst_start = 1 + i * BINS_PER_BAND;

    for (j = 0; j < BINS_PER_BAND; j++) {
        int src_idx = is_inverted ? src_start + (BINS_PER_BAND - 1 - j) : src_start + j;
        int dst_idx = dst_start + j;

        in_real[dst_idx] = fft_real[src_idx];
        in_imag[dst_idx] = fft_imag[src_idx];
    }
}

// 4. ЕРМІТОВА СИМЕТРІЯ (Критично для відновлення чистого звуку!)
// Звуковий сигнал завжди дійсний, тому його спектр має бути симетричним
for (k = 1; k < 128; k++) {
    in_real[FRAME_SIZE - k] = in_real[k];
    in_imag[FRAME_SIZE - k] = -in_imag[k]; // Комплексне спряження (мінус)
}

// 5. Зворотне ШПФ
exp_ifft = ifft256(in_real, in_imag, fft_real, fft_imag);

// 6. Компенсація масштабування та виведення
// ШПФ в ADSP використовує арифметику з фіксованою комою і "стискає" сигнал,
// щоб уникнути переповнення. Ми повинні відновити гучність зсувом бітів.
total_exp = exp_fft + exp_ifft;
shift = total_exp - 8;

for(i = 0; i < FRAME_SIZE; i++) {
    long temp = (long)fft_real[i];

    if (shift > 0) temp = temp << shift;
    else if (shift < 0) temp = temp >> (-shift);

    // Кліппінг: захист від переповнення 16-бітного типу
    if (temp > 32767) temp = 32767;
    if (temp < -32768) temp = -32768;
```

```
        output[i] = (short)temp;
    }
}

// ======================================================================
// ГОЛОВНА ПРОГРАММА
// ======================================================================
void main() {
    int i;

    // Одноразова генерація ключів розшифрування при запуску
    get_inverse_key_c(time_key, time_inv_key, NUM_CHUNKS);
    get_inverse_key_c(freq_key, freq_inv_key, NUM_BANDS);

    while(1) {
        // --- ЧИТАННЯ ---
        // Забираємо рівно 1 фрейм (256 відліків) з вхідного порту
        for(i = 0; i < FRAME_SIZE; i++) {
            rx_buffer[i] = stream_in;
        }

        // --- ШИФРУВАННЯ (2D) ---
        time_scramble_c(rx_buffer, temp_time, time_key, FRAME_SIZE, NUM_CHUNKS);
        freq_scramble_c(temp_time, temp_freq, freq_key, FRAME_SIZE, NUM_BANDS);

        // --- ДЕШИФРУВАННЯ ---
        // Виконується строго у зворотному порядку до шифрування!
        freq_scramble_c(temp_freq, temp_desc, freq_inv_key, FRAME_SIZE, NUM_BANDS);
        time_scramble_c(temp_desc, tx_buffer, time_inv_key, FRAME_SIZE, NUM_CHUNKS);

        // --- ВИВЕДЕННЯ ---
        // Відправляємо паралельно обидва потоки в порти симулятора
        for(i = 0; i < FRAME_SIZE; i++) {
            stream_out_scrambled = temp_freq[i];
            stream_out_descrambled = tx_buffer[i];
        }
    }
}
```