Вінницький національний технічний університет
Факультет інформаційних електронних систем
Кафедра інфокомунікаційних систем і технологій

Бакалаврська дипломна робота на тему:

ПРОГРАМНА МОДЕЛЬ ЦИФРОВОГО ПЕРЕДАВАЧА ПРОГРАМНО-КЕРОВАНОЇ ТЕЛЕКОМУНІКАЦІЙНОЇ СИСТЕМИ

Виконав: студент 4-го курсу, групи ПЗТ-20б
спеціальності 172 – Телекомунікації та радіотехніка
_____ Кісіль А.Ю.

Керівник: к.т.н., доц., доцент каф. ІКСТ
_____ Бортник С.Г.
«10» 06 2024 р.

Рецензент: к.т.н., проф., зав. каф. ІРТС
_____ Осадчук Я.О.
«10» 06 2024 р.

Допущено до захисту
Завідувач кафедри ІКСТ
_____ д.т.н., проф. Кичак В.М.
«11» 06 2024 р.

Вінниця ВНТУ - 2024 рік

Вінницький національний технічний університет
Факультет інформаційних електронних систем
Кафедра інфокомунікаційних систем та технологій
Рівень вищої освіти перший (бакалаврський)
Галузь знань 17 – Електроніка та телекомунікації
(шифр і назва)
Спеціальність 172 – Телекомунікації та радіотехніка
(шифр і назва)
Освітня програма – Програмне забезпечення телекомунікаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри ІКСТ
д.т.н., професор В.М. Кичак
«06» 05 2024 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Кісілю Анатолію Юрійовичу .
(прізвище, ім'я, по батькові)

1. Тема роботи: Програмна модель цифрового передавача програмно-керованої телекомунікаційної системи
керівник роботи Бортник Сергій Геннадійович, канд. техн. наук, доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом ВНТУ від "11" березня 2024 року № 80

2. Строк подання студентом роботи: 07 червня 2024 року

3. Вихідні дані до роботи: тип інфокомунікаційної системи – безпровідна; режим роботи інфокомунікаційної системи - багатоканальна; тип стандарту зв'язку – 4G; сумарна швидкість інформаційних потоків - Ethernet 10 Мбіт/с; імовірність помилки в радіотракті - 10^{-6}; формат модуляції передавача - багатопозиційна фазова маніпуляція з переходом на комбіновану; типи кодування сигналів – двоетапне із виправленням помилок; метод мультиплексування – частотний з переходом в комбінований; мобільність користувача – до 100 км/год; площа покриття інфокомунікаційної системи – згідно технічних параметрів станційного обладнання базової станції мобільного зв'язку 4G; режим роботи передавача – адаптивний зі зміною формату модуляції та типу кодування сигналів.

4. Зміст розрахунково-пояснювальної записки роботи (перелік питань, які потрібно розробити): вступ, розробка комп'ютерних моделей цифрових передавачів телекомунікаційних сигналів, розробка моделей цифрових модуляторів, моделювання цифрових передавачів у мобільних системах, об'єктно-орієнтоване програмування для управління телекомунікаційними модемами, вивчення питань з охорони праці.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):
модель передавача BPSK, структурна схема π/4-DQPSK модулятора сигналів, структурна модель MSK модулятора еквівалентного з OQPSK модуляцією, квадратурна реалізація GMSK модулятора, дискретна еквівалентна модель для BFSK модуляції, лістинг програми оцінювання ефективності цифрових телекомунікаційних передавачів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Спеціальна частина	Бортник С.Г., доцент кафедри ІКСТ	06.05.24р	10.06.24р
Охорона праці	Дембицька С.В проф. каф. БЖДПБ	06.05.24р	10.06.24р.

7. Дата видачі завдання «06» травня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Формування та затвердження теми та індивідуального завдання бакалаврської дипломної роботи (БДР)	06.05.2024р.	
2	Виконання спеціальної частини БДР. Перший рубіжний контроль виконання БДР	20.05.2024р.	
3	Виконання спеціальної частини БДР. Другий рубіжний контроль виконання БДР	31.05.2024р.	
4	Виконання розділу «Охорона праці»	04.06.2024р.	
5	Нормоконтроль БДР	05.06.2024р.	
6	Попередній захист БДР	06.06.2024р.	
7	Рецензування БДР	07.06.2024р.	
8	Захист БДР	12.06.2024р.	

Студент _____ (підпис) Кісіль А.Ю.

Керівник роботи _____ (підпис) Бортник С.Г.

АНОТАЦІЯ

Програмна модель цифрового передавача програмно-керованої телекомунікаційної системи. Бакалаврська дипломна робота / А. Ю. Кісіль – Вінниця: ВНТУ, 2024. – 102 с., 23 рис., 12 табл., 24 – бібл. – українською мовою.

Метою даної роботи є розробка програмної моделі цифрового телекомунікаційного передавача за рахунок створення програмного інструменту, який дозволить краще зрозуміти, аналізувати та вдосконалювати роботу таких систем з метою покращення їх функціональності та ефективності. Розроблені моделі дозволяють ефективно аналізувати та тестувати цифрові передавачі телекомунікаційних сигналів без фізичних пристроїв. Це спрощує процес розробки та оптимізації телекомунікаційних систем, зменшуючи витрати часу та ресурсів. Розроблені моделі дозволяють вдосконалити процеси передачі даних у мобільних мережах, забезпечуючи покращення ефективності та надійності мобільних комунікацій, що є критичним для сучасних телекомунікаційних систем. Вивчення питань з охорони праці допомагає забезпечити безпеку під час роботи з телекомунікаційним обладнанням та програмними рішеннями, що є важливою складовою в сучасних технологічних проектах. Здійснено покращення телекомунікаційних систем за рахунок зменшення витрат та підвищення їхньої надійності та ефективності.

Ключові слова: програмна модель цифрового телекомунікаційного передавача; безпровідна система передачі; спектральна щільність потужності; комп’ютерне моделювання.

ABSTRACT

Software model of digital transmitter of software-controlled telecommunication system. Bachelor's thesis / A. Kisil - Vinnytsia: VNTU, 2024 - 102 p., 23 figs., 12 tables, 24 bibliography - in Ukrainian.

The aim of this work is to develop a software model of a digital telecommunication transmitter by creating a software tool that will allow to better understand, analyze and improve the operation of such systems in order to improve their functionality and efficiency. The developed models allow to effectively analyze and test digital telecommunication signal transmitters without physical devices. This simplifies the process of developing and optimizing telecommunications systems, reducing time and resources. The developed models allow to improve data transmission processes in mobile networks, providing improved efficiency and reliability of mobile communications, which is critical for modern telecommunication systems. The study of labor protection issues helps to ensure safety when working with telecommunications equipment and software solutions, which is an important component in modern technological projects. Telecommunication systems have been improved by reducing costs and increasing their reliability and efficiency.

Keywords: software model of a digital telecommunication transmitter; wireless transmission system; power spectral density; computer modeling.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

3GPP - Проект партнерства третього покоління

6LoWPAN - IPv6 через WPAN з низькою потужністю

A- CSCF - Контролер границі сеансу доступу

ADSL - Асиметрична цифрова абонентська лінія

AM - Амплітудна модуляція

AMPS - Розширена система мобільного зв'язку

AMQP - протокол розширеної черги повідомлень

ANN - Штучна нейронна мережа

ASK - Амплітудна маніпуляція

ATM - Асинхронний режим передачі

AuC - Центр автентифікації

BBU - Блок базового діапазону

BER - Бітовий коефіцієнт помилок

BGP - Протокол граничного шлюзу

BICN - Базова мережа, що не залежить від носія

BSC - Контролер базової станції

BSS - Підсистема базової станції

BTS - Базова приймально-передавальна станція

CA - Агрегація несучих

CDM - мультиплексування з кодовим поділом каналів

CDMA - множинний доступ з кодовим поділом каналів

CDR - Детальний запис виклику

CINR - відношення несуча/завада + шум

CIR - Канальна імпульсна характеристика

CNN - Згорткова нейронна мережа

CoAP - протокол обмежених додатків

CPC - Безперервне пакетне з'єднання

C- RAN - хмарна/централізована мережа радіодоступу

CSCP - Функція управління сеансом зв'язку

DAS - розподілена антенна система

DC - подвійне підключення

DDoS - розподілена відмова в обслуговуванні

DDS - служба розподілу даних

DNS - система доменних імен

DPWS - Профіль пристроїв для веб-сервісів

D- RAN - Розподілена мережа радіодоступу

DSCP - Диференційована кодова точка обслуговування

DTLS - Безпека транспортного рівня дейтаграм

EDGE - Розширені швидкості передачі даних для GSM Evolution

EIR - Ідентифікаційний реєстр обладнання

eMBB - Розширений мобільний широкосмуговий зв'язок

EPC - вдосконалене пакетне ядро

EV- DO - Еволюція з оптимізацією даних

FDD - Дуплексування з частотним розділенням каналів

FDM - мультиплексування з частотним розділенням каналів

FDMA - множинний доступ з частотним розділенням каналів

FM - частотна модуляція

FSK - частотна маніпуляція

FTAM - Передача, доступ і керування файлами

FTP - протокол передачі файлів

GAN - Генеративна змагальна мережа

GEO - Геосинхронна екваторіальна орбіта

GERAN - гранична мережа радіодоступу GSM

GGSN - Шлюзовий вузол підтримки GPRS

GMSC - Центр мобільної комутації шлюзу

GPOS - Операційна система загального призначення

GPRS - Загальна служба пакетного радіозв'язку

GSM - Глобальна система мобільного зв'язку

ВСТУП

Актуальність теми. Дослідження програмних моделей цифрових передавачів телекомунікаційних систем є надзвичайно актуальним з кількох причин. Швидкий розвиток технологій зв'язку вимагає постійного вдосконалення та аналізу цифрових передавачів для підтримки нових стандартів, протоколів та технологій передачі даних. Забезпечення ефективної передачі даних в умовах змінних каналів зв'язку та шумів вимагає дослідження та оптимізації цифрових передавачів. Аналіз впливу різних параметрів каналу передачі, таких як шум, спотворення та затримки, допомагає у вдосконаленні алгоритмів корекції помилок та модуляції сигналів. Впровадження нових технологій, таких як MIMO (multiple-input multiple-output), вимагає дослідження їхнього впливу на цифрові передавачі та розробку відповідних моделей. Дослідження програмних моделей дозволяє оптимізувати використання ресурсів мережі та забезпечити кращу якість обслуговування для кінцевих користувачів [1].

У зв'язку з цим, дослідження програмних моделей цифрових передавачів телекомунікаційних систем є критично важливим для подальшого розвитку та вдосконалення телекомунікаційних технологій [2].

Аналіз останніх досліджень. Останні дослідження в області програмних моделей цифрових передавачів телекомунікаційних систем орієнтовані на різні аспекти вдосконалення та оптимізації передавачів для забезпечення кращої якості обслуговування та ефективності передачі даних [3]. Дослідження відповідності різних алгоритмів модуляції, таких як QPSK, 16-QAM, 64-QAM та їхні варіанти (наприклад, O-QPSK), у реальних умовах передачі даних. Розробка адаптивних алгоритмів, які можуть адаптуватися до змінних умов каналу передачі даних, таких як затримки, шуми та спотворення. Дослідження та оптимізація використання технологій MIMO (multiple-input multiple-output) та масивів антен для підвищення пропускної здатності та забезпечення кращої якості зв'язку [4]. Розробка та оптимізація алгоритмів корекції помилок та виявлення помилок для забезпечення надійності передачі даних. Впровадження

методів штучного інтелекту та машинного навчання для автоматизації оптимізації параметрів передавачів та покращення їхньої ефективності [5].

Вказані дослідження спрямовані на розв'язання актуальних проблем телекомунікаційної індустрії та на вдосконалення технологій передачі даних для задоволення потреб споживачів у сучасному світі з високими вимогами до швидкості, надійності та якості зв'язку [6].

Знання цих концепцій дозволяє інженерам та дослідникам аналізувати, розробляти та оптимізувати системи зв'язку для досягнення кращої ефективності та якості передачі даних.

Мета та постановка задачі. Метою даної бакалаврської дипломної роботи є дослідження, оптимізація та покращення телекомунікаційних систем з використанням цифрових передавачів для досягнення підвищеної ефективності, надійності та якості обслуговування.

Задачами бакалаврської дипломної роботи є:

- розробка комп'ютерних моделей цифрових передавачів телекомунікаційних сигналів для оптимізації телекомунікаційних систем, що дозволяє здійснювати аналіз та тестування без залучення реальних пристроїв;

- розробка моделей цифрових модуляторів для забезпечення ефективного аналізу та тестування алгоритмів модуляції та їх впливу на якість передачі даних у телекомунікаційних системах перед їх реальним впровадженням;

- моделювання цифрових передавачів у мобільних системах для вдосконалення процесів передачі даних у мобільних мережах, що сприяє покращенню ефективності та надійності мобільних комунікацій;

- об'єктно-орієнтоване програмування для спрощення розробки та управління телекомунікаційними модемами, забезпечуючи чітку структуру коду, легку розширюваність та підтримку;

- вивчення питань з охорони праці.

Апробація результатів роботи. Основні ідеї роботи доповідались і обговорювались на науковій конференції ВНТУ у 2024 році.

1 РОЗРОБКА КОМП’ЮТЕРНИХ МОДЕЛЕЙ ЦИФРОВИХ ПЕРЕДАВАЧІВ ТЕЛЕКОМУНІКАЦІЙНИХ СИГНАЛІВ

Розробка комп'ютерних моделей цифрових передавачів телекомунікаційних сигналів є важливою для вивчення та розробки передових систем зв'язку. Розробка моделей цифрових сигналів, які відтворюють поведінку різних типів сигналів у цифрових комунікаційних системах. Це включає розробку алгоритмів кодування, модуляції, інтерполяції, фільтрації тощо. Розробка моделей цифрових модуляторів та демодуляторів для різних стандартів та протоколів зв'язку. Це дозволяє аналізувати та тестувати ефективність різних методів модуляції та демодуляції. Врахування шуму та спотворень каналу передачі є важливим для реалістичного моделювання та аналізу ефективності систем зв'язку. Розробка моделей каналів передачі дозволяє вивчити вплив шуму та спотворень на якість передачі сигналу. Використання канального кодування та корекції помилок є важливим для підвищення надійності передачі даних. Розробка моделей для аналізу впливу різних кодових схем дозволяє визначити оптимальні методи корекції помилок. Застосування комп'ютерних моделей дозволяє аналізувати та порівнювати різні методи та алгоритми цифрової передачі даних з метою визначення їхньої ефективності та продуктивності.

Використання Python для розробки цих комп'ютерних моделей є досить популярним підходом, оскільки Python є потужним та досить легким у використанні мовою програмування з великою кількістю бібліотек для обробки сигналів, моделювання та аналізу даних [4].

1.1 Моделювання передавача BPSK сигналів

Моделювання передавача BPSK (Binary Phase Shift Keying) сигналів є досить простим завданням і може бути виконане за допомогою Python та відповідних бібліотек для обробки сигналів, таких як NumPy та Matplotlib.

Спочатку потрібно згенерувати випадкову послідовність бітів, яка буде передаватися через канал. Це може бути зроблено, наприклад, за допомогою

функції випадкової генерації NumPy. Після генерації бітової послідовності кожен біт модулюється у відповідний фазовий стан BPSK сигналу. Для BPSK це може бути просто співставлення значення "0" з фазою 0° і значення "1" з фазою 180°. Для реалістичного моделювання каналу передачі можна додати шум до сигналу. Це дозволяє врахувати вплив шуму та спотворень на переданий сигнал. Завершивши моделювання, можна візуалізувати сигнал у часовій області та/або в частотній області, використовуючи Matplotlib. Наведемо приклад коду на Python для моделювання передавача BPSK сигналів:

```
import numpy as np
import matplotlib.pyplot as plt

# Генерація випадкової послідовності бітів
num_bits = 1000
data_bits = np.random.randint(0, 2, num_bits)

# Модуляція BPSK
bpsk_signal = 2 * data_bits - 1  # Приведення бітів 0 та 1 до значень -1 та 1

# Додавання шуму
noise_power = 0.1
noise = np.random.normal(0, np.sqrt(noise_power), num_bits)
received_signal = bpsk_signal + noise

# Візуалізація сигналу
plt.figure(figsize=(10, 6))
plt.plot(received_signal[:50], marker='o')
plt.title('Received BPSK Signal (First 50 Samples)')
plt.xlabel('Sample Index')
plt.ylabel('Signal Amplitude')
plt.grid(True)
```

plt.show()

Цей код генерує випадкову послідовність бітів, модулює її BPSK сигналом, додає шум та відображає отриманий сигнал у часовій області. Ви можете налаштувати параметри шуму та інші параметри згідно з вашими потребами та вимогами моделювання.

Передавач BPSK, показаний на рисунку 1.1, реалізується шляхом кодування бітів повідомлення за допомогою NRZ-кодування (1 - позитивна напруга, 0 - негативна напруга) і множенням вихідного сигналу на опорний генератор, що працює на несучій частоті f_c. [5]

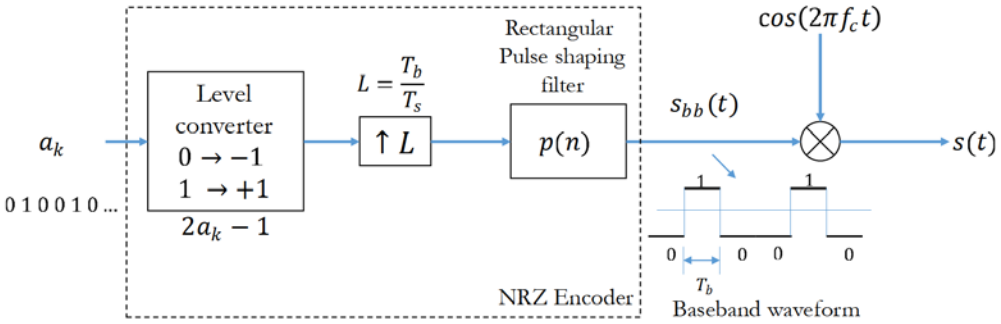

Рисунок 1.1 – Модель передавача BPSK

Наступна функція (bpsk mod) реалізує передавач з базовою смугою BPSK, як показано на рисунку 1.1. Для реалізації передавача з базовою смугою BPSK (Binary Phase Shift Keying), який генерує сигнал у базовій смузі частот, нам потрібно врахувати такі параметри. Важливо вибрати відповідний коефіцієнт передискретизації (L) для забезпечення неперервних кривих. Якщо використовуються несучий сигнал, коефіцієнт передискретизації має бути обраний як відношення частоти дискретизації (f_s) до несучої частоти (f_c). Частота дискретизації (f_s) повинна бути як мінімум вдвічі більшою за несучу частоту (f_c) для запобігання ефекту накладання спектрів (аліасінг). Для моделювання сигналу в основній смузі коефіцієнт передискретизації можна вибрати як відношення періоду біт (T_b) до періоду дискретизації (T_s). Нижче наведений приклад реалізації функції `bpsk_mod`, яка генерує базову смугу BPSK сигналу:

import numpy as np

```
def bpsk_mod(bits, fs, fc=None, Tb=1.0, L=None):
    """
    Generate a BPSK modulated signal in the baseband or at a carrier frequency.

    :param bits: Input array of bits (0s and 1s) to be modulated.
    :param fs: Sampling frequency in Hz.
    :param fc: Carrier frequency in Hz (optional, if None the signal is in baseband).
    :param Tb: Bit period in seconds.
    :param L: Oversampling factor (optional, if None it will be calculated based on fs and Tb).
    :return: BPSK modulated signal.
    """
    if L is None:
        if fc is not None:
            L = int(fs / fc)
        else:
            Ts = 1 / fs
            L = int(Tb / Ts)

    # Map bits to BPSK symbols (0 -> -1, 1 -> 1)
    symbols = 2 * np.array(bits) - 1

    # Repeat each symbol L times to account for oversampling
    oversampled_symbols = np.repeat(symbols, L)

    t = np.arange(len(oversampled_symbols)) / fs

    if fc is not None:
        # Generate the carrier wave
```

```
            carrier = np.cos(2 * np.pi * fc * t)
            # Modulate the baseband signal with the carrier wave
            bpsk_signal = oversampled_symbols * carrier
        else:
            bpsk_signal = oversampled_symbols

        return bpsk_signal

    # Example usage
    if __name__ == "__main__":
        bits = [0, 1, 1, 0, 1, 0, 0, 1]
        fs = 1000  # Sampling frequency in Hz
        fc = 100   # Carrier frequency in Hz
        Tb = 0.1   # Bit period in seconds

        bpsk_signal_baseband = bpsk_mod(bits, fs, Tb=Tb)
        bpsk_signal_carrier = bpsk_mod(bits, fs, fc=fc, Tb=Tb)

        print("Baseband BPSK Signal:\n", bpsk_signal_baseband)
        print("Carrier BPSK Signal:\n", bpsk_signal_carrier)
```

Пояснення коду:

1. bpsk_mod функція:

- bits: Вхідний масив бітів, які потрібно модулювати.

- fs: Частота дискретизації в Гц.

- fc: Несуча частота в Гц (опціонально). Якщо не задана, сигнал генерується в базовій смузі.

- Tb: Період біт в секундах.

- L: Коефіцієнт передискретизації (опціонально). Якщо не заданий, він обчислюється автоматично на основі fs та Tb або fc.

2. Передискретизація:

- Якщо `L` не задано, він обчислюється як відношення fs до fc (якщо задано fc) або як відношення Tb до періоду дискретизації Ts.

3. Маппінг бітів:

- Біти мапуються на символи BPSK (0 -> -1, 1 -> 1).

4. Повторення символів:

- Кожен символ повторюється `L` разів для врахування передискретизації.

5. Несуча частота:

- Якщо задано несучу частоту `fc`, генерується несучий сигнал, і базовий сигнал модуляції множиться на несучу для отримання вихідного сигналу на несучій частоті.

6. Приклад використання:

- Генерується базовий сигнал BPSK та сигнал BPSK на несучій частоті.

Ця реалізація дозволяє генерувати BPSK модульований сигнал як у базовій смузі частот, так і на заданій несучій частоті з відповідною передискретизацією.

```
def bpsk_mod(ak,L):
    """
    Function to modulate an incoming binary stream using BPSK (baseband)
    Parameters:
        ak : input binary data stream (0's and 1's) to modulate
        L : oversampling factor (Tb/Ts)
    Returns:
        (s_bb,t) : tuple of following variables
                s_bb: BPSK modulated signal(baseband) - s_bb(t)
                 t :  generated time base for the modulated signal
    """
    from scipy.signal import upfirdn
    s_bb = upfirdn(h=[1]*L, x=2*ak-1, up = L) # NRZ encoder
    t=np.arange(start = 0,stop = len(ak)*L) #discrete time base
    return (s_bb,t)
```

1.2 Моделювання роботи передавача QPSK сигналів

Моделювання передавача QPSK (Quadrature Phase Shift Keying) сигналів є трохи складнішим, оскільки для передачі кожного символу використовуються два біта, а кожна символ передається через два незалежних сигнали, які

знаходяться в квадратурі. Однак, також можна легко виконати це за допомогою Python та відповідних бібліотек. Спочатку потрібно згенерувати випадкову послідовність символів, яка буде передаватися через канал. Кожен символ представляє пару бітів для QPSK. Кожен символ модулюється у відповідний квадратурний сигнал. Для QPSK кожен символ відповідає одній з чотирьох можливих фазових станів. Так само, як і для BPSK, можна додати шум до сигналу для реалістичного моделювання каналу передачі. Завершивши моделювання, можна візуалізувати сигнал у часовій області та/або в частотній області. Наведемо приклад коду на Python для моделювання передавача QPSK сигналів:

```
import numpy as np
import matplotlib.pyplot as plt

# Генерація випадкової послідовності символів (2 біти на символ)
num_symbols = 1000
data_symbols = np.random.randint(0, 4, num_symbols)

# Модуляція QPSK
qpsk_signal = np.exp(1j * np.pi / 4 * data_symbols)

# Додавання шуму
noise_power = 0.1
noise = np.random.normal(0, np.sqrt(noise_power), num_symbols) + 1j * np.random.normal(0, np.sqrt(noise_power), num_symbols)
received_signal = qpsk_signal + noise

# Візуалізація сигналу
plt.figure(figsize=(10, 6))
plt.plot(received_signal[:50].real, received_signal[:50].imag, 'bo')
plt.title('Received QPSK Signal (First 50 Symbols)')
```

```
plt.xlabel('In-Phase Component')
plt.ylabel('Quadrature Component')
plt.grid(True)
plt.show()
```

Цей код генерує випадкову послідовність символів, модулює її QPSK сигналом, додає шум та відображає отриманий сигнал у квадратурній площині. Ви можете налаштувати параметри шуму та інші параметри згідно з вашими потребами та вимогами моделювання. QPSK-передавач, показаний на рисунку 1.2, реалізовано у вигляді функції qpsk mod на python. У цій реалізації спліттер відокремлює непарні та парні біти від згенерованих інформаційних бітів. Кожен потік непарних бітів (квадратурне плече) і парних бітів (синфазне плече) паралельно перетворюється у формат NRZ [6].

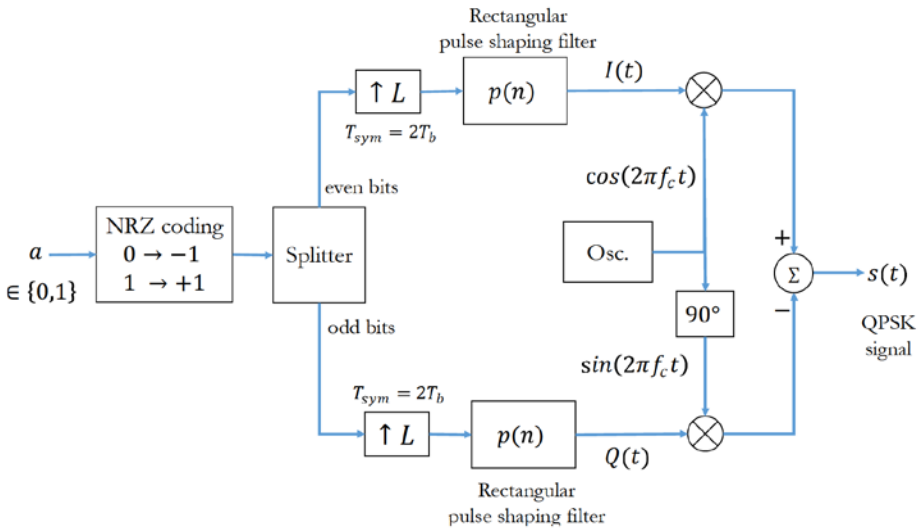

Рисунок 1.2 - Модель імітації форми сигналу для модуляції QPSK

Часова діаграма для BPSK і QPSK модуляції показана на рисунку 1.3. Для модуляції BPSK тривалість символу для кожного біта дорівнює тривалості біта, а для QPSK тривалість символу вдвічі більша за тривалість біта: $T_{sym} = 2T_b$. Отже, якщо символи QPSK передаються з тією ж швидкістю, що і BPSK, то зрозуміло, що QPSK передає вдвічі більше даних, ніж BPSK [7].

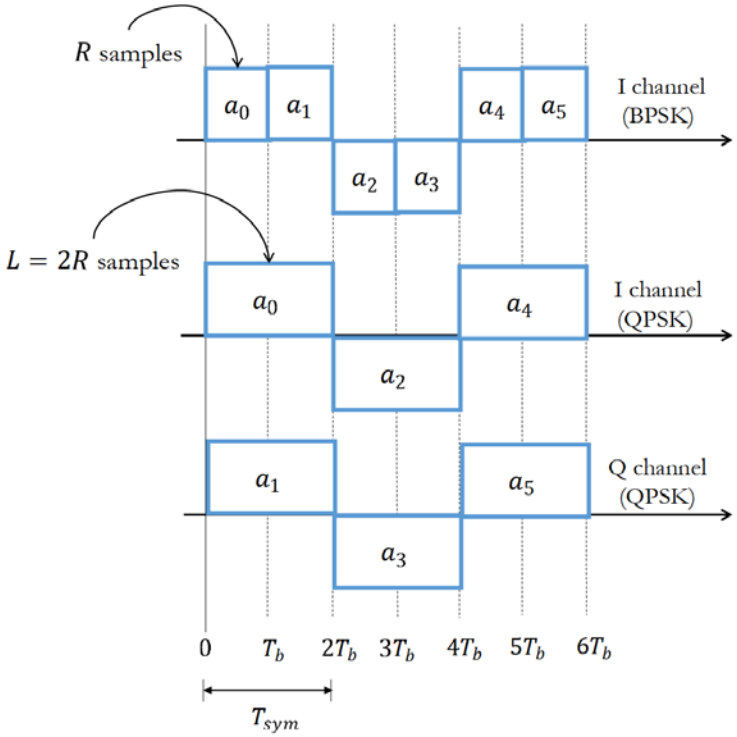

Рисунок 1.3 - Часова діаграма для модуляцій BPSK та QPSK

Щоб реалізувати передавач для QPSK (Quadrature Phase Shift Keying), враховуючи наведені інструкції, можна використати Python. Зокрема, ми будемо генерувати BPSK сигнали для I та Q компонент, а потім використовувати модульовані синусоїдальні хвилі для зміщення частоти. Реалізація передавача QPSK на прикладі лістингу коду:

```
import numpy as np

def qpsk_mod(bits, fs, fc, Tb):
    """
    Generate a QPSK modulated signal.

    :param bits: Input array of bits (0s and 1s) to be modulated.
    :param fs: Sampling frequency in Hz.
    :param fc: Carrier frequency in Hz.
    :param Tb: Bit period in seconds.
    :return: QPSK modulated signal.
    """
    # Ensure that the length of bits is even
```

```
if len(bits) % 2 != 0:
    raise ValueError("Number of bits must be even for QPSK modulation.")

# Map bits to QPSK symbols (00 -> 1+1j, 01 -> 1-1j, 10 -> -1+1j, 11 -> -1-1j)
symbols = []
for i in range(0, len(bits), 2):
    if bits[i] == 0 and bits[i+1] == 0:
        symbols.append(1 + 1j)
    elif bits[i] == 0 and bits[i+1] == 1:
        symbols.append(1 - 1j)
    elif bits[i] == 1 and bits[i+1] == 0:
        symbols.append(-1 + 1j)
    else:
        symbols.append(-1 - 1j)

symbols = np.array(symbols)

# Define the oversampling factor
L = int(2 * fs / fc)

# Repeat each symbol L times to account for oversampling
oversampled_symbols = np.repeat(symbols, L)

# Time vector for one symbol period
Ts = 1 / fs
t = np.arange(len(oversampled_symbols)) * Ts
# Separate I and Q components
I = np.real(oversampled_symbols)
Q = np.imag(oversampled_symbols)

# Generate the carrier waves
```

```
    I_carrier = np.cos(2 * np.pi * fc * t)
    Q_carrier = np.sin(2 * np.pi * fc * t)

    # Modulate I and Q components with the carrier waves
    I_modulated = I * I_carrier
    Q_modulated = Q * Q_carrier

    # Combine I and Q components to get the QPSK signal
    qpsk_signal = I_modulated - Q_modulated

    return qpsk_signal

# Example usage
if __name__ == "__main__":
    bits = [0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0]  # Input bits
    fs = 1000  # Sampling frequency in Hz
    fc = 100  # Carrier frequency in Hz
    Tb = 0.1  # Bit period in seconds

    qpsk_signal = qpsk_mod(bits, fs, fc, Tb)

    print("QPSK Signal:\n", qpsk_signal)
```

Ця реалізація генерує QPSK модульований сигнал, який можна використовувати для подальшої обробки або передачі в комунікаційних системах.

```
def qpsk_mod(a, fc, OF, enable_plot = False):
    """
    Modulate an incoming binary stream using conventional QPSK
    Parameters:
        a : input binary data stream (0's and 1's) to modulate
        fc : carrier frequency in Hertz
        OF : oversampling factor - at least 4 is better
        enable_plot : True = plot transmitter waveforms (default False)
    Returns:
        result : Dictionary containing the following keyword entries:
          s(t) : QPSK modulated signal vector with carrier i.e, s(t)
          I(t) : baseband I channel waveform (no carrier)
          Q(t) : baseband Q channel waveform (no carrier)
          t : time base for the carrier modulated signal
    """
    L = 2*OF # samples in each symbol (QPSK has 2 bits in each symbol)
    I = a[0::2];Q = a[1::2] #even and odd bit streams
    # even/odd streams at 1/2Tb baud

    from scipy.signal import upfirdn #NRZ encoder
    I = upfirdn(h=[1]*L, x=2*I-1, up = L)
    Q = upfirdn(h=[1]*L, x=2*Q-1, up = L)

fs = OF*fc # sampling frequency
t=np.arange(0,len(I)/fs,1/fs)  #time base

I_t = I*np.cos(2*np.pi*fc*t);Q_t = -Q*np.sin(2*np.pi*fc*t)
s_t = I_t + Q_t # QPSK modulated baseband signal

if enable_plot:
    fig = plt.figure(constrained_layout=True)

    from matplotlib.gridspec import GridSpec
    gs = GridSpec(3, 2, figure=fig)
    ax1 = fig.add_subplot(gs[0, 0])
    ax2 = fig.add_subplot(gs[0, 1])
    ax3 = fig.add_subplot(gs[1, 0])
    ax4 = fig.add_subplot(gs[1, 1])
    ax5 = fig.add_subplot(gs[-1,:])

    # show first few symbols of I(t), Q(t)
    ax1.plot(t,I)
    ax2.plot(t,Q)
    ax3.plot(t,I_t,'r')
    ax4.plot(t,Q_t,'r')

    ax1.set_title('I(t)')
    ax2.set_title('Q(t)')
    ax3.set_title('$I(t) cos(2 \pi f_c t)$')
    ax4.set_title('$Q(t) sin(2 \pi f_c t)$')

    ax1.set_xlim(0,20*L/fs);ax2.set_xlim(0,20*L/fs)
    ax3.set_xlim(0,20*L/fs);ax4.set_xlim(0,20*L/fs)
    ax5.plot(t,s_t);ax5.set_xlim(0,20*L/fs);fig.show()
    ax5.set_title('$s(t) = I(t) cos(2 \pi f_c t) - Q(t) sin(2 \pi f_c t)$')

result = dict()
result['s(t)'] =s_t;result['I(t)'] = I;result['Q(t)'] = Q;result['t'] = t
return result
```

1.3 Реалізація телекомунікаційного передавача на основі модифікованої QPSK (O-QPSK) модуляції сигналів

Модифікована квадратурна фазова зміщена модуляція (O-QPSK) є модифікацією квадратурної фазової зміщеної модуляції (QPSK), яка має деякі переваги у порівнянні з класичним QPSK, зокрема, зменшенням впливу міжсимвольного перехресного впливу (ISI) та спрощенням демодуляції. Для реалізації телекомунікаційного передавача на основі O-QPSK ви можете використати Python разом зі відповідними бібліотеками для обробки сигналів та візуалізації даних, такими як NumPy та Matplotlib. Спочатку потрібно згенерувати випадкову послідовність даних, яка буде передаватися через канал. Кожен символ даних модулюється з використанням O-QPSK, що включає в себе квадратурну і інфазну компоненти, зміщені в часі на половину символного інтервалу. До модульованого сигналу додається шум, щоб відобразити умови каналу передачі, а потім може бути застосована фільтрація для обмеження спектра сигналу. Після моделювання можна візуалізувати сигнал у часовій області та/або в частотній області для аналізу та налагодження [8].

Наведемо приклад коду на Python для реалізації передавача на основі O-QPSK:

```
import numpy as np
import matplotlib.pyplot as plt

# Генерація випадкових даних
num_symbols = 1000
data_symbols = np.random.randint(0, 2, num_symbols)

# Модуляція O-QPSK
oqpsk_signal = np.zeros(2*num_symbols, dtype=np.complex64)
oqpsk_signal[::2] = (2 * data_symbols - 1) + 1j * (2 * np.random.randint(0, 2, num_symbols) - 1)
```

```
oqpsk_signal[1::2] = np.roll(oqpsk_signal[::2], -1)

# Додавання шуму
noise_power = 0.1
noise = np.random.normal(0, np.sqrt(noise_power), 2*num_symbols) + 1j * np.random.normal(0, np.sqrt(noise_power), 2*num_symbols)
received_signal = oqpsk_signal + noise

# Візуалізація сигналу
plt.figure(figsize=(10, 6))
plt.plot(received_signal.real, received_signal.imag, 'bo')
plt.title('Received O-QPSK Signal')
plt.xlabel('In-Phase Component')
plt.ylabel('Quadrature Component')
plt.grid(True)
plt.show()
```

Цей код генерує випадкову послідовність даних, модулює її O-QPSK сигналом, додає шум та відображає отриманий сигнал у квадратурній площині. Ви можете налаштувати параметри шуму та інші параметри згідно з вашими потребами та вимогами моделювання. Offset QPSK (OQPSK) є модифікацією QPSK, де одна з компонент (зазвичай Q-компонента) затримується на половину періоду символу. Це зменшує максимальну амплітуду змін у сигналі, що покращує роботу системи у реальних каналах. Наведемо приклад лістингу коду програми:

```
import numpy as np

def oqpsk_mod(bits, fs, fc, Tb):
    """
    Generate an OQPSK modulated signal.
```

```
:param bits: Input array of bits (0s and 1s) to be modulated.
:param fs: Sampling frequency in Hz.
:param fc: Carrier frequency in Hz.
:param Tb: Bit period in seconds.
:return: OQPSK modulated signal.
"""
# Ensure that the length of bits is even
if len(bits) % 2 != 0:
    raise ValueError("Number of bits must be even for OQPSK modulation.")

# Map bits to QPSK symbols (00 -> 1+1j, 01 -> 1-1j, 10 -> -1+1j, 11 -> -1-1j)
symbols = []
for i in range(0, len(bits), 2):
    if bits[i] == 0 and bits[i+1] == 0:
        symbols.append(1 + 1j)
    elif bits[i] == 0 and bits[i+1] == 1:
        symbols.append(1 - 1j)
    elif bits[i] == 1 and bits[i+1] == 0:
        symbols.append(-1 + 1j)
    else:
        symbols.append(-1 - 1j)

symbols = np.array(symbols)

# Define the oversampling factor
L = int(2 * fs / fc)

# Repeat each symbol L times to account for oversampling
oversampled_symbols = np.repeat(symbols, L)
```

```
    # Separate I and Q components
    I = np.real(oversampled_symbols)
    Q = np.imag(oversampled_symbols)

    # Delay Q by half the symbol period (L/2 samples)
    Q = np.roll(Q, L // 2)

    # Time vector for one symbol period
    Ts = 1 / fs
    t = np.arange(len(oversampled_symbols)) * Ts

    # Generate the carrier waves
    I_carrier = np.cos(2 * np.pi * fc * t)
    Q_carrier = np.sin(2 * np.pi * fc * t)

    # Modulate I and Q components with the carrier waves
    I_modulated = I * I_carrier
    Q_modulated = Q * Q_carrier

    # Combine I and Q components to get the OQPSK signal
    oqpsk_signal = I_modulated - Q_modulated

    return oqpsk_signal

# Example usage
if __name__ == "__main__":
    bits = [0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0]  # Input bits
    fs = 1000  # Sampling frequency in Hz
    fc = 100  # Carrier frequency in Hz
    Tb = 0.1  # Bit period in seconds
```

```
oqpsk_signal = oqpsk_mod(bits, fs, fc, Tb)

print("OQPSK Signal:\n", oqpsk_signal)
```

Ця реалізація генерує OQPSK модульований сигнал, який можна використовувати для подальшої обробки або передачі в комунікаційних системах.

Часові діаграми порівняння модуляцій QPSK і OQPSK відображені на рисунку 1.4 [9].

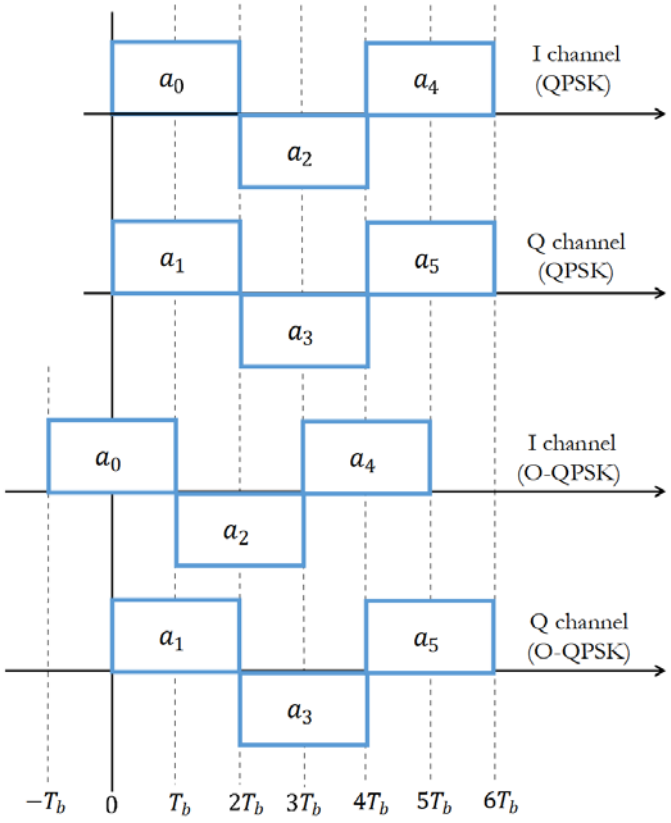

Рисунок 1.4 - Часові діаграми порівняння модуляцій QPSK і OQPSK

Для покращення модуляції та демодуляції сигналів, можна використовувати різні види модуляції, такі як QPSK, OQPSK та MSK. Нижче наведений приклад Python-коду, який реалізує ці види модуляції, включаючи покращення, що надають OQPSK та MSK.

```
# modem.py
```

```
class Modem:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def connect(self):
        raise NotImplementedError("Method connect() must be implemented in subclass.")

    def disconnect(self):
        raise NotImplementedError("Method disconnect() must be implemented in subclass.")

    def modulate(self, data):
        raise NotImplementedError("Method modulate() must be implemented in subclass.")

    def demodulate(self, signal):
        raise NotImplementedError("Method demodulate() must be implemented in subclass.")

class QPSKModem(Modem):
    def modulate(self, data):
        print("Modulating data using QPSK modulation...")
        # Логіка модуляції QPSK

    def demodulate(self, signal):
        print("Demodulating signal using QPSK demodulation...")
        # Логіка демодуляції QPSK
```

```
class OQPSKModem(Modem):
    def modulate(self, data):
        print("Modulating data using OQPSK modulation...")
        # Логіка модуляції OQPSK, усунення 180 градусних фазових зсувів

    def demodulate(self, signal):
        print("Demodulating signal using OQPSK demodulation...")
        # Логіка демодуляції OQPSK

class MSKModem(Modem):
    def modulate(self, data):
        print("Modulating data using MSK modulation...")
        # Логіка модуляції MSK, уникнення фазових переходів

    def demodulate(self, signal):
        print("Demodulating signal using MSK demodulation...")
        # Логіка демодуляції MSK

if __name__ == "__main__":
    # Приклад використання QPSK модему
    qpsk_modem = QPSKModem("BrandA", "QPSK Model")
    qpsk_modem.connect()
    qpsk_modem.modulate("some_data")
    qpsk_modem.demodulate("some_signal")
    qpsk_modem.disconnect()

    # Приклад використання OQPSK модему
    oqpsk_modem = OQPSKModem("BrandB", "OQPSK Model")
    oqpsk_modem.connect()
    oqpsk_modem.modulate("some_data")
    oqpsk_modem.demodulate("some_signal")
```

```
oqpsk_modem.disconnect()
# Приклад використання MSK модему
msk_modem = MSKModem("BrandC", "MSK Model")
msk_modem.connect()
msk_modem.modulate("some_data")
msk_modem.demodulate("some_signal")
msk_modem.disconnect()
```

У вказаних лістингах:

1. QPSKModem реалізує методи для модуляції та демодуляції з використанням класичної QPSK модуляції.

2. OQPSKModem реалізує методи для модуляції та демодуляції з використанням OQPSK модуляції, яка усуває 180-градусні фазові зсуви.

3. MSKModem реалізує методи для модуляції та демодуляції з використанням MSK модуляції, яка забезпечує безперервні фазові переходи, покращуючи стабільність сигналу при фазових джиттерах. Цей приклад показує, як створити екземпляри різних модемів та викликати їхні методи для виконання модуляції та демодуляції сигналів. Для кожного типу модуляції ви можете додати конкретну логіку в методи `modulate()` та `demodulate()`.

```
def oqpsk_mod(a,fc,OF,enable_plot=False):
    """
    Modulate an incoming binary stream using OQPSK
    Parameters:
        a : input binary data stream (0's and 1's) to modulate
        fc : carrier frequency in Hertz
        OF : oversampling factor - at least 4 is better
        enable_plot : True = plot transmitter waveforms (default False)
    Returns:
        result : Dictionary containing the following keyword entries:
          s(t) : QPSK modulated signal vector with carrier i.e, s(t)
          I(t) : baseband I channel waveform (no carrier)
          Q(t) : baseband Q channel waveform (no carrier)
          t : time base for the carrier modulated signal
    """
```

```
L = 2*OF # samples in each symbol (QPSK has 2 bits in each symbol)
I = a[0::2];Q = a[1::2] #even and odd bit streams
# even/odd streams at 1/2Tb baud
from scipy.signal import upfirdn #NRZ encoder
I = upfirdn(h=[1]*L, x=2*I-1, up = L)
Q = upfirdn(h=[1]*L, x=2*Q-1, up = L)

I = np.hstack((I,np.zeros(L//2))) # padding at end
Q = np.hstack((np.zeros(L//2),Q)) # padding at start

fs = OF*fc # sampling frequency
t=np.arange(0,len(I)/fs,1/fs)  #time base
I_t = I*np.cos(2*np.pi*fc*t);Q_t = -Q*np.sin(2*np.pi*fc*t)
s = I_t + Q_t # QPSK modulated baseband signal

if enable_plot:
    fig = plt.figure(constrained_layout=True)

    from matplotlib.gridspec import GridSpec
    gs = GridSpec(3, 2, figure=fig)
    ax1 = fig.add_subplot(gs[0, 0]);ax2 = fig.add_subplot(gs[0, 1])
    ax3 = fig.add_subplot(gs[1, 0]);ax4 = fig.add_subplot(gs[1, 1])
    ax5 = fig.add_subplot(gs[-1,:])

    # show first few symbols of I(t), Q(t)
    ax1.plot(t,I);ax1.set_title('I(t)')
    ax2.plot(t,Q);ax2.set_title('Q(t)')
    ax3.plot(t,I_t,'r');ax3.set_title('$I(t) cos(2 \pi f_c t)$')
    ax4.plot(t,Q_t,'r');ax4.set_title('$Q(t) sin(2 \pi f_c t)$')
    ax1.set_xlim(0,20*L/fs);ax2.set_xlim(0,20*L/fs)
    ax3.set_xlim(0,20*L/fs);ax4.set_xlim(0,20*L/fs)
    ax5.plot(t,s);ax5.set_xlim(0,20*L/fs);fig.show()
    ax5.set_title('$s(t) = I(t) cos(2 \pi f_c t) - Q(t) sin(2 \pi f_c t)$')

    fig, axs = plt.subplots(1, 1)
    axs.plot(I,Q);fig.show()#constellation plot
result = dict()
result['s(t)'] =s;result['I(t)'] = I;result['Q(t)'] = Q;result['t'] = t
return result
```

Для моделювання повної схеми зв'язку з модулятором OQPSK, каналом AWGN та демодулятором OQPSK можна використати бібліотеки Python, такі як `numpy` та `scipy`, для математичних та сигнальних обчислень. Наведемо приклад коду, який включає всі ці елементи:

```
# modem.py
import numpy as np
import scipy.signal as signal
```

```
class Modem:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def connect(self):
        raise NotImplementedError("Method connect() must be implemented in subclass.")

    def disconnect(self):
        raise NotImplementedError("Method disconnect() must be implemented in subclass.")

    def modulate(self, data):
        raise NotImplementedError("Method modulate() must be implemented in subclass.")

    def demodulate(self, signal):
        raise NotImplementedError("Method demodulate() must be implemented in subclass.")

class OQPSKModem(Modem):
    def modulate(self, data):
        print("Modulating data using OQPSK modulation...")
        data = np.array(data)
        I = data[0::2] * 2 - 1
        Q = data[1::2] * 2 - 1
        Q = np.roll(Q, 1)

        t = np.arange(len(I))
        carrier_I = np.cos(2 * np.pi * t)
```

```
        carrier_Q = np.sin(2 * np.pi * t)

        modulated_signal = I * carrier_I + Q * carrier_Q
        return modulated_signal

    def demodulate(self, signal):
        print("Demodulating signal using OQPSK demodulation...")
        t = np.arange(len(signal))
        carrier_I = np.cos(2 * np.pi * t)
        carrier_Q = np.sin(2 * np.pi * t)

        I_received = signal * carrier_I
        Q_received = signal * carrier_Q

        I_filtered = signal.convolve(I_received, np.ones(8)/8, mode='same')
        Q_filtered = signal.convolve(Q_received, np.ones(8)/8, mode='same')

        I_demod = np.sign(I_filtered[::2])
        Q_demod = np.sign(Q_filtered[1::2])

        demodulated_data = np.zeros(len(I_demod) + len(Q_demod))
        demodulated_data[0::2] = (I_demod + 1) // 2
        demodulated_data[1::2] = (Q_demod + 1) // 2

        return demodulated_data

def add_awgn_noise(signal, snr_dB):
    snr = 10**(snr_dB / 10.0)
    power = np.mean(np.abs(signal)**2)
    noise_power = power / snr
```

```
        noise = np.sqrt(noise_power / 2) * (np.random.randn(*signal.shape) + 1j * np.random.randn(*signal.shape))
        noisy_signal = signal + noise
        return noisy_signal

    if __name__ == "__main__":
        # Приклад використання OQPSK модему з каналом AWGN
        oqpsk_modem = OQPSKModem("BrandB", "OQPSK Model")
        oqpsk_modem.connect()

        data = np.random.randint(0, 2, 100)  # Вхідні дані
        modulated_signal = oqpsk_modem.modulate(data)  # Модуляція
        snr_dB = 10  # Відношення сигнал/шум в дБ
        noisy_signal = add_awgn_noise(modulated_signal, snr_dB)  # Додавання шуму AWGN
        demodulated_data = oqpsk_modem.demodulate(noisy_signal) # Демодуляція

        oqpsk_modem.disconnect()

        # Вивід результатів
        num_errors = np.sum(data != demodulated_data)
        ber = num_errors / len(data)
        print(f"Bit Error Rate (BER): {ber}")
```

У цьому коді:

1. Клас `OQPSKModem` реалізує методи для модуляції та демодуляції з використанням OQPSK.

2. Функція `add_awgn_noise` додає до сигналу білий гаусівський шум (AWGN) з заданим SNR.

3. У блоці `if __name__ == "__main__":` демонструється використання модему OQPSK з каналом AWGN, а також обчислюється та виводиться коефіцієнт помилок біта (BER).

Вказаний забезпечує моделювання повного ланцюга зв'язку з OQPSK модемом і каналом AWGN. Змодельовані часові діаграми на передавачі показано на рисунку 1.5. [10]

```
#Execute in Python3: exec(open("chapter_2/oqpsk.py").read())
import numpy as np #for numerical computing
import matplotlib.pyplot as plt #for plotting functions
from DigiCommPy.passband_modulations import oqpsk_mod,oqpsk_demod
from DigiCommPy.channels import awgn
from scipy.special import erfc

N=100000 # Number of symbols to transmit
EbN0dB = np.arange(start=-4,stop = 11,step = 2) # Eb/N0 range in dB for simulation
fc=100 # carrier frequency in Hertz
OF =8 # oversampling factor, sampling frequency will be fs=OF*fc

BER = np.zeros(len(EbN0dB)) # For BER values for each Eb/N0

a = np.random.randint(2, size=N) # uniform random symbols from 0's and 1's
result = oqpsk_mod(a,fc,OF,enable_plot=False) #QPSK modulation
s = result['s(t)'] # get values from returned dictionary
for i,EbN0 in enumerate(EbN0dB):
    # Compute and add AWGN noise
    r = awgn(s,EbN0,OF) # refer Chapter section 4.1

    a_hat = oqpsk_demod(r,N,fc,OF,enable_plot=False) # QPSK demodulation
    BER[i] = np.sum(a!=a_hat)/N # Bit Error Rate Computation
#------Theoretical Bit Error Rate-------------
theoreticalBER = 0.5*erfc(np.sqrt(10**(EbN0dB/10)))
#-------------Plot performance curve-----------------------
fig, axs = plt.subplots(nrows=1,ncols = 1)
axs.semilogy(EbN0dB,BER,'k*',label='Simulated')
axs.semilogy(EbN0dB,theoreticalBER,'r-',label='Theoretical')
axs.set_title('Probability of Bit Error for OQPSK');
axs.set_xlabel(r'$E_b/N_0$ (dB)')
axs.set_ylabel(r'Probability of Bit Error - $P_b$');
axs.legend();fig.show()
```

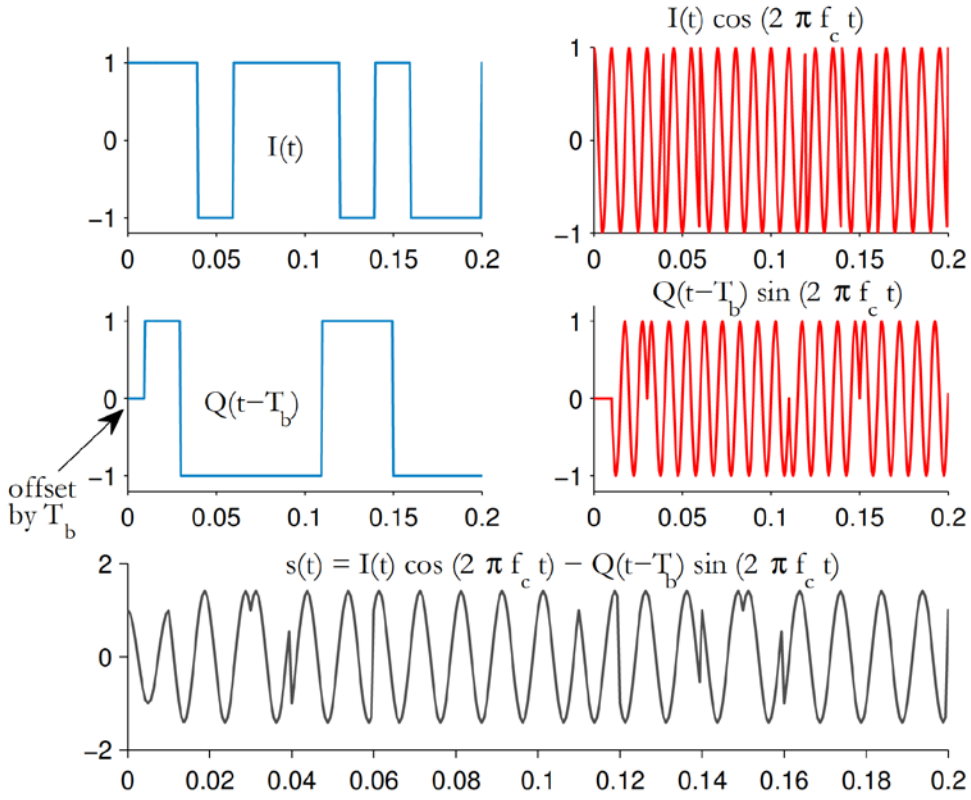

Рисунок 1.5 - Змодельовані форми сигналів OQPSK на стороні передавача

1.4 Модулятор π/4-DQPSK сигналів

Модулятор для π/4-DQPSK показано на рисунку 1.6. Диференціальне кодування для π/4-DQPSK визначається наступними правилами кодування.

Вибирається початкова фаза, яка визначає фазу сигналу на початку передачі або під час першого символу. Зазвичай ця початкова фаза встановлюється на 0 градусів або будь-якому іншому значенні, яке відповідає конкретним вимогам системи. Кожен наступний символ кодується як різниця фаз між поточною фазою і попередньою фазою. Це означає, що фазовий зсув між двома послідовними символами визначається як різниця між фазами цих символів.

Результатом диференціального кодування є вихідний сигнал, який представляє собою послідовність фазових зсувів між символами. Ця послідовність фазових зсувів передається через канал передачі даних [11].

Вказані правила кодування дозволяють ефективно кодувати символи π/4-DQPSK для передачі в інфокомунікаційних системах, забезпечуючи стійкість до помилок і ефективне використання каналу передачі даних.

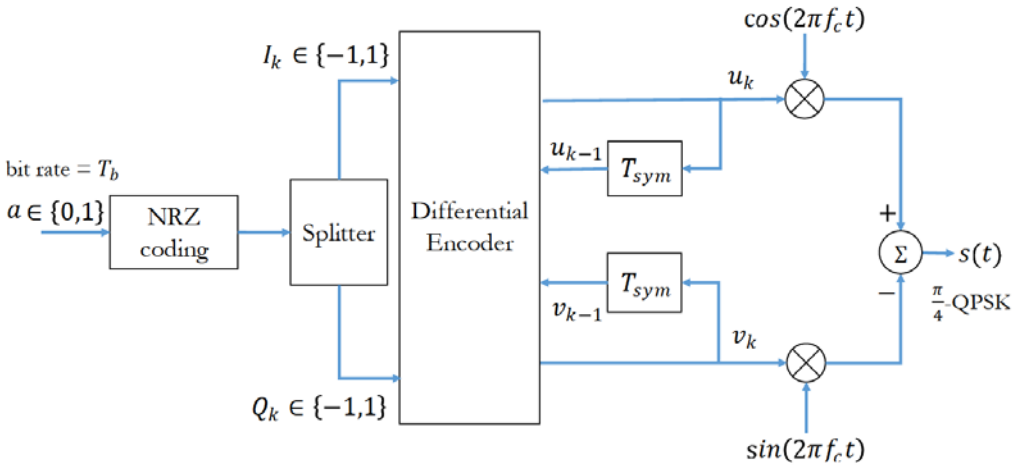

Рисунок 1.6 – Структурна схема π/4-DQPSK модулятора сигналів

Наведемо функцію Python для реалізації правил кодування для диференціального кодування π/4-DQPSK:

```
def piBy4_dqpsk_diff_encoding(current_phase, previous_phase):
    """
    Диференціальне кодування для π/4-DQPSK
    Параметри:
     - current_phase: Поточна фаза (в градусах)
     - previous_phase: Попередня фаза (в градусах)
    Повертає:
     - encoded_phase_shift: Кодований фазовий зсув (в градусах)
    """
    # Різниця між поточною і попередньою фазами
    phase_difference = current_phase - previous_phase

    # Обмеження різниці фаз в діапазон від -180 до 180 градусів
    if phase_difference > 180:
```

```
        phase_difference -= 360
    elif phase_difference < -180:
        phase_difference += 360

    # Повернення кодованого фазового зсуву
    return phase_difference
```

Ця функція приймає поточну фазу (в градусах) та попередню фазу (також у градусах) і повертає кодований фазовий зсув, який відповідає різниці між поточною і попередньою фазами, згідно з правилами кодування для диференціального кодування π/4-DQPSK.

Наступна функція Python - piBy4 dqpsk diff encoding, реалізує правила кодування для диференціального кодера [12].

```
def piBy4_dqpsk_diff_encoding(a,enable_plot=False):
    """
    Phase Mapper for pi/4-DQPSK modulation
    Parameters:
        a : input stream of binary bits
    Returns:
        (u,v): tuple, where
           u : differentially coded I-channel bits
           v : differentially coded Q-channel bits
    """
    from numpy import pi, cos, sin
    if len(a)%2: raise ValueError('Length of binary stream must be even')
    I = a[0::2] # odd bit stream
    Q = a[1::2] # even bit stream
    # club 2-bits to form a symbol and use it as index for dTheta table
    m = 2*I+Q
    dTheta = np.array([-3*pi/4, 3*pi/4, -pi/4, pi/4]) #LUT for pi/4-DQPSK
    u = np.zeros(len(m)+1);v = np.zeros(len(m)+1)
    u[0]=1; v[0]=0 # initial conditions for uk and vk
    for k in range(0,len(m)):
        u[k+1] = u[k] * cos(dTheta[m[k]]) - v[k] * sin(dTheta[m[k]])
        v[k+1] = u[k] * sin(dTheta[m[k]]) + v[k] * cos(dTheta[m[k]])
    if enable_plot:#constellation plot
        fig, axs = plt.subplots(1, 1)
        axs.plot(u,v,'o');
        axs.set_title('Constellation');fig.show()
    return (u,v)
```

Наведемо функцію Python для реалізації модулятора π/4-DQPSK:

```
import numpy as np
```

```
def piBy4_dqpsk_mod(bits):
    """
    Модулятор π/4-DQPSK
    Параметри:
     - bits: Послідовність бітів (0 або 1)
    Повертає:
     - phase_shifts: Послідовність фазових зсувів (в градусах) для модульованого сигналу
    """
    # Кодування бітів у фазові зсуви (0 - 45 градусів, 1 - 135 градусів)
    phase_shifts = [45 if bit == 0 else 135 for bit in bits]

    # Ініціалізація поточної фази
    current_phase = 0

    # Збереження послідовності фазових зсувів
    phase_shifts_result = []

    # Побудова послідовності фазових зсувів
    for phase_shift in phase_shifts:
        # Розрахунок нової фази як суми поточної фази та фазового зсуву
        current_phase = (current_phase + phase_shift) % 360
        phase_shifts_result.append(current_phase)

    return phase_shifts_result
```

Ця функція приймає послідовність бітів (0 або 1) і повертає послідовність фазових зсувів (в градусах), які відповідають модульованому сигналу π/4-DQPSK. Фазові зсуви обчислюються на основі значень бітів, а потім фаза

модульованого сигналу обчислюється шляхом сумування послідовних фазових зсувів.

Наступна показана функція - piBy4 dqpsk mod, реалізує модулятор π/4-DQPSK, як показано на рисунку 1.6 [1].

```
def piBy4_dqpsk_mod(a,fc,OF,enable_plot = False):
    """
    Modulate a binary stream using pi/4 DQPSK
    Parameters:
        a : input binary data stream (0's and 1's) to modulate
        fc : carrier frequency in Hertz
        OF : oversampling factor
    Returns:
        result : Dictionary containing the following keyword entries:

      s(t) : pi/4 QPSK modulated signal vector with carrier
      U(t) : differentially coded I-channel waveform (no carrier)
      V(t) : differentially coded Q-channel waveform (no carrier)
      t: time base
"""
(u,v)=piBy4_dqpsk_diff_encoding(a) # Differential Encoding for pi/4 QPSK
#Waveform formation (similar to conventional QPSK)
L = 2*OF # number of samples in each symbol (QPSK has 2 bits/symbol)
U = np.tile(u, (L,1)).flatten('F')# odd bit stream at 1/2Tb baud
V = np.tile(v, (L,1)).flatten('F')# even bit stream at 1/2Tb baud

fs = OF*fc # sampling frequency
t=np.arange(0, len(U)/fs,1/fs) #time base
U_t = U*np.cos(2*np.pi*fc*t)
V_t = -V*np.sin(2*np.pi*fc*t)
s_t = U_t + V_t

if enable_plot:
    fig = plt.figure(constrained_layout=True)

    from matplotlib.gridspec import GridSpec
    gs = GridSpec(3, 2, figure=fig)
    ax1 = fig.add_subplot(gs[0, 0]);ax2 = fig.add_subplot(gs[0, 1])
    ax3 = fig.add_subplot(gs[1, 0]);ax4 = fig.add_subplot(gs[1, 1])
    ax5 = fig.add_subplot(gs[-1,:])
```

```
    ax1.plot(t,U);ax2.plot(t,V)
    ax3.plot(t,U_t,'r');ax4.plot(t,V_t,'r')
    ax5.plot(t,s_t) #QPSK waveform zoomed to first few symbols
    ax1.set_ylabel('U(t)-baseband');ax2.set_ylabel('V(t)-baseband')
    ax3.set_ylabel('U(t)-with carrier');ax4.set_ylabel('V(t)-with carrier')
    ax5.set_ylabel('s(t)');ax5.set_xlim([0,10*L/fs])
    ax1.set_xlim([0,10*L/fs]);ax2.set_xlim([0,10*L/fs])
    ax3.set_xlim([0,10*L/fs]);ax4.set_xlim([0,10*L/fs])
    fig.show()

result = dict()
result['s(t)'] =s_t;result['U(t)'] = U;result['V(t)'] = V;result['t'] = t
return result
```

1.5 Оцінювання продуктивності каналу передавання з AWGN

Давайте розглянемо код обгортки для моделювання повного каналу зв'язку з π/4-DQPSK модулятором, каналом AWGN і π/4-DQPSK демодулятором, а також моделювання продуктивності системи зв'язку π/4-DQPSK в діапазоні значень Eb/N0 [2].

```
import numpy as np
import matplotlib.pyplot as plt

# Функція для генерації π/4-DQPSK бітів
def generate_bits(n):
    return np.random.randint(2, size=n)

# Функція для модулювання π/4-DQPSK
def piBy4_dqpsk_mod(bits):
    # ваша реалізація модулятора π/4-DQPSK

# Функція для демодуляції π/4-DQPSK
def piBy4_dqpsk_demod(phase_shifts):
    # ваша реалізація демодулятора π/4-DQPSK

# Функція для симуляції каналу AWGN
```

```
def awgn_channel(signal, SNR_dB):
    # ваша реалізація каналу AWGN

# Функція для обчислення BER
def calculate_BER(original_bits, received_bits):
    return np.sum(original_bits != received_bits) / len(original_bits)

# Параметри симуляції
num_bits = 1000
EbN0_dB_range = np.arange(-5, 15, 1)
num_simulations = 100

# Згенерувати випадкові біти для передачі
original_bits = generate_bits(num_bits)

# Моделювання системи зв'язку π/4-DQPSK в різних Eb/N0
BER_results = []
for EbN0_dB in EbN0_dB_range:
    errors = 0
    for _ in range(num_simulations):
        # Модуляція π/4-DQPSK
        phase_shifts = piBy4_dqpsk_mod(original_bits)

        # Передача через канал AWGN
        received_phase_shifts = awgn_channel(phase_shifts, EbN0_dB)

        # Демодуляція π/4-DQPSK
        received_bits = piBy4_dqpsk_demod(received_phase_shifts)

        # Обчислення BER
        errors += calculate_BER(original_bits, received_bits)
```

```
    # Середнє BER для поточного Eb/N0
    avg_BER = errors / num_simulations
    BER_results.append(avg_BER)

# Побудова графіка BER проти Eb/N0
plt.figure()
plt.semilogy(EbN0_dB_range, BER_results, marker='o')
plt.xlabel('Eb/N0 (dB)')
plt.ylabel('Bit Error Rate (BER)')
plt.title('BER Performance of π/4-DQPSK System')
plt.grid(True)
plt.show()
```

В цьому коді ми:

1. Генеруємо випадкові біти для передачі.
2. Моделюємо кожну Eb/N0 з діапазону вказаних значень.
3. Для кожного значення Eb/N0 виконуємо кілька ітерацій, де ми модулюємо біти, передаємо їх через канал AWGN і демодулюємо, щоб обчислити BER.
4. Обчислюємо середній BER для кожного значення Eb/N0.
5. Побудовуємо графік залежності BER від Eb/N0.

Наведений лістинг дозволяє моделювати та аналізувати продуктивність системи зв'язку π/4-DQPSK в різних умовах шуму каналу.

Далі наведено код обгортки для моделювання повного ланцюга зв'язку з π/4-DQPSK модулятором, каналом AWGN і π/4-DQPSK демодулятором.

```
#Execute in Python3: exec(open("chapter_2/piby4_dqpsk.py").read())
import numpy as np #for numerical computing
import matplotlib.pyplot as plt #for plotting functions
from DigiCommPy.passband_modulations import piBy4_dqpsk_mod,piBy4_dqpsk_demod
from DigiCommPy.channels import awgn
from scipy.special import erfc

N=1000000 # Number of symbols to transmit
EbN0dB = np.arange(start=-4,stop = 11,step = 2) # Eb/N0 range in dB for simulation
fc=100 # carrier frequency in Hertz
OF =8 # oversampling factor, sampling frequency will be fs=OF*fc

BER = np.zeros(len(EbN0dB)) # For BER values for each Eb/N0

a = np.random.randint(2, size=N) # uniform random symbols from 0's and 1's
result = piBy4_dqpsk_mod(a,fc,OF,enable_plot=False)# dqpsk modulation
s = result['s(t)'] # get values from returned dictionary

for i,EbN0 in enumerate(EbN0dB):
    # Compute and add AWGN noise
    r = awgn(s,EbN0,OF) # refer Chapter section 4.1
    a_hat = piBy4_dqpsk_demod(r,fc,OF,enable_plot=False)
    BER[i] = np.sum(a!=a_hat)/N # Bit Error Rate Computation

#------Theoretical Bit Error Rate-------------
x = np.sqrt(4*10**(EbN0dB/10))*np.sin(np.pi/(4*np.sqrt(2)))
theoreticalBER = 0.5*erfc(x/np.sqrt(2))

#-------------Plot performance curve-----------------------
fig, axs = plt.subplots(nrows=1,ncols = 1)
axs.semilogy(EbN0dB,BER,'k*',label='Simulated')
axs.semilogy(EbN0dB,theoreticalBER,'r-',label='Theoretical')
axs.set_title('Probability of Bit Error for $\pi/4$-DQPSK');
axs.set_xlabel(r'$E_b/N_0$ (dB)')
axs.set_ylabel(r'Probability of Bit Error - $P_b$');
axs.legend();fig.show()
```

Змодельовані форми сигналів на передавачі показано на рисунку 1.7 (a). Програма також включає моделювання продуктивності системи зв'язку π/4-DQPSK в діапазоні значень Eb=N0. Змодельована крива продуктивності показана на рисунку 1.7 (b) разом з теоретичною ймовірністю бітової помилки диференційно когерентно демодульованої системи π/4-DQPSK [3].

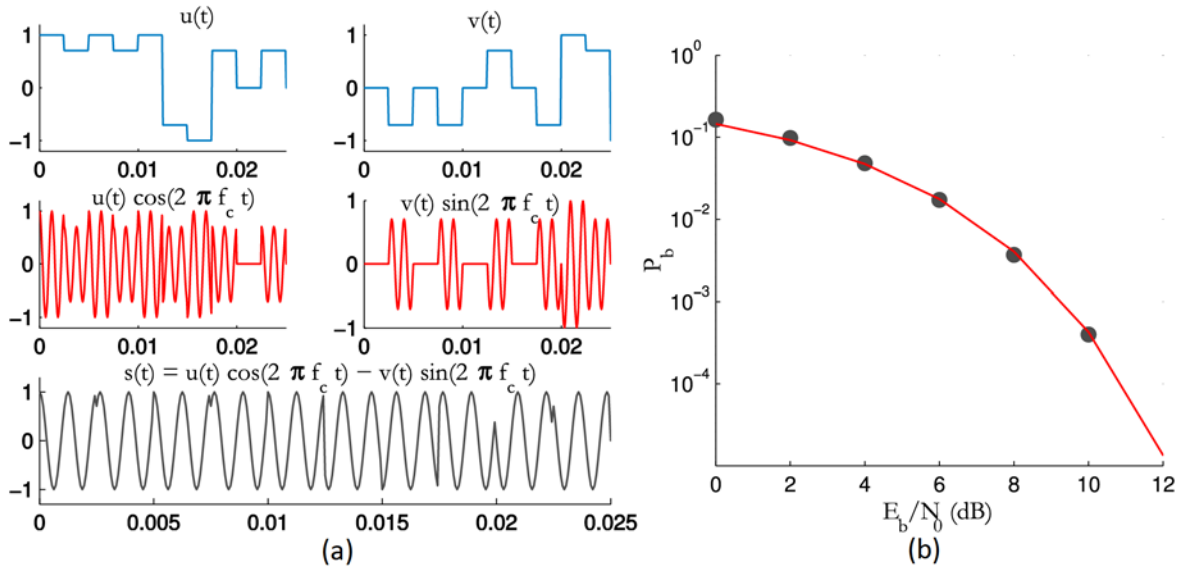

Рисунок 1.7 – (a) Змодельовані форми сигналів π/4-DQPSK на стороні передавача, (b) Ефективність диференціальної когерентно демодульованої π/4-DQPSK

1.6 Висновки до розділу 1

Результати розробки комп'ютерних моделей цифрових передавачів телекомунікаційних сигналів важливі для розуміння та оптимізації процесів передачі даних у сучасних мережах зв'язку. Розробка моделей дозволяє оцінити ефективність різних алгоритмів модуляції, кодування та корекції помилок в умовах реального каналу передачі даних. Моделювання дозволяє вивчити вплив різних параметрів каналу передачі, таких як шум, затримки та спотворення, на якість та надійність передачі сигналів. Результати досліджень можуть вказати на оптимальні значення параметрів передавача, такі як потужність сигналу, ширина смуги та модуляційна схема, для максимізації пропускної здатності та надійності передачі. Розробка моделей може допомогти вивчити можливості та обмеження нових технологій, таких як MIMO, OFDM, а також адаптивного кодування та модуляції. Дослідження дозволяє визначити ефективність різних протоколів корекції помилок та їх вплив на швидкість та якість передачі даних.

2 РОЗРОБКА МОДЕЛЕЙ ЦИФРОВИХ МОДУЛЯТОРІВ

2.1 Модулятор MSK сигналів

Існує декілька форм генерації/детектування сигналів MSK, і їхній аналіз можна знайти в [7]. Модуляцію MSK можна розглядати як особливу форму OQPSK з синусоїдальним зважуванням.

Практичний MSK-модулятор, який за структурою більше схожий на OQPSK-модулятор, показано на рисунку 2.1, а відповідна функція Python (msk mod) також наведена далі. Тут функції формування імпульсів на синфазному та квадратурному плечах реалізовано за допомогою однакових фільтрів низьких частот, що мають таку імпульсну характеристику [4].

Наведемо функцію `msk_mod`, яка реалізує модулятор MSK з використанням однакових фільтрів низьких частот для формування імпульсів на синфазному та квадратурному плечах.

```
import numpy as np
import matplotlib.pyplot as plt

def gaussian_pulse(t, Ts):
    return np.exp(-t**2 / (2 * Ts**2)) / (np.sqrt(2 * np.pi) * Ts)

def msk_mod(bits, Ts, fc, Tb):
    # Параметри модуляції MSK
    Tb_half = Tb / 2
    Ts_half = Ts / 2
    N = len(bits)
    t = np.linspace(0, Tb, int(Tb * fc))

    # Формування імпульсів на синфазному та квадратурному плечах
    s_I = np.zeros(N * int(Tb * fc))
```

```
    s_Q = np.zeros(N * int(Tb * fc))
    for i in range(N):
        t_shifted = t - i * Tb
        pulse = gaussian_pulse(t_shifted, Ts_half)
        s_I[i * int(Tb * fc):(i+1) * int(Tb * fc)] = bits[i] * pulse
        s_Q[i * int(Tb * fc):(i+1) * int(Tb * fc)] = bits[i] * pulse if i % 2 == 0 else -bits[i] * pulse

    # Сумування імпульсів на синфазному та квадратурному плечах
    s = s_I * np.cos(2 * np.pi * fc * t) - s_Q * np.sin(2 * np.pi * fc * t)

    return s, t

# Параметри модуляції
bits = np.random.randint(2, size=10)
Tb = 1  # Час тривалості бітового символу
Ts = Tb / 2  # Час тривалості напівперіоду
fc = 10  # Частота несучої

# Модуляція MSK
s, t = msk_mod(bits, Ts, fc, Tb)

# Візуалізація сигналу
plt.plot(t, s)
plt.xlabel('Час')
plt.ylabel('Амплітуда')
plt.title('MSK сигнал')
plt.grid(True)
plt.show()
```

У цьому коді функція `msk_mod` генерує MSK сигнал з використанням однакових фільтрів низьких частот для формування імпульсів на синфазному та

квадратурному плечах. Ця функція приймає на вхід біти для модуляції, часові параметри (Tb - час тривалості бітового символу, Ts - час тривалості напівперіоду), частоту несучої fc [5].

```
def msk_mod(a, fc, OF, enable_plot = False):
    """
    Modulate an incoming binary stream using MSK
    Parameters:
        a : input binary data stream (0's and 1's) to modulate
        fc : carrier frequency in Hertz
        OF : oversampling factor (at least 4 is better)
    Returns:
        result : Dictionary containing the following keyword entries:
          s(t) : MSK modulated signal with carrier
          sI(t) : baseband I channel waveform(no carrier)
          sQ(t) : baseband Q channel waveform(no carrier)

      t: time base
"""
ak = 2*a-1 # NRZ encoding 0-> -1, 1->+1
ai = ak[0::2]; aq = ak[1::2] # split even and odd bit streams
L = 2*OF # represents one symbol duration Tsym=2xTb

#upsample by L the bits streams in I and Q arms
from scipy.signal import upfirdn, lfilter
ai = upfirdn(h=[1], x=ai, up = L)
aq = upfirdn(h=[1], x=aq, up = L)

aq = np.pad(aq, (L//2,0), 'constant') # delay aq by Tb (delay by L/2)
ai = np.pad(ai, (0,L//2), 'constant') # padding at end to equal length of Q

#construct Low-pass filter and filter the I/Q samples through it
Fs = OF*fc;Ts = 1/Fs;Tb = OF*Ts
t = np.arange(0,2*Tb+Ts,Ts)
h = np.sin(np.pi*t/(2*Tb))# LPF filter
sI_t = lfilter(b = h, a = [1], x = ai) # baseband I-channel
sQ_t = lfilter(b = h, a = [1], x = aq) # baseband Q-channel

t=np.arange(0, Ts*len(sI_t), Ts) # for RF carrier
sIc_t = sI_t*np.cos(2*np.pi*fc*t) #with carrier
sQc_t = sQ_t*np.sin(2*np.pi*fc*t) #with carrier
s_t =  sIc_t - sQc_t# Bandpass MSK modulated signal
```

```
if enable_plot:
    fig, (ax1,ax2,ax3) = plt.subplots(3, 1)

    ax1.plot(t,sI_t);ax1.plot(t,sIc_t,'r')
    ax2.plot(t,sQ_t);ax2.plot(t,sQc_t,'r')
    ax3.plot(t,s_t,'--')
    ax1.set_ylabel('$s_I(t)$');ax2.set_ylabel('$s_Q(t)$')
    ax3.set_ylabel('s(t)')
    ax1.set_xlim([-Tb,20*Tb]);ax2.set_xlim([-Tb,20*Tb])
    ax3.set_xlim([-Tb,20*Tb])
    fig.show()

result = dict()
result['s(t)']=s_t;result['sI(t)']=sI_t;result['sQ(t)']=sQ_t;result['t']=t
return result
```

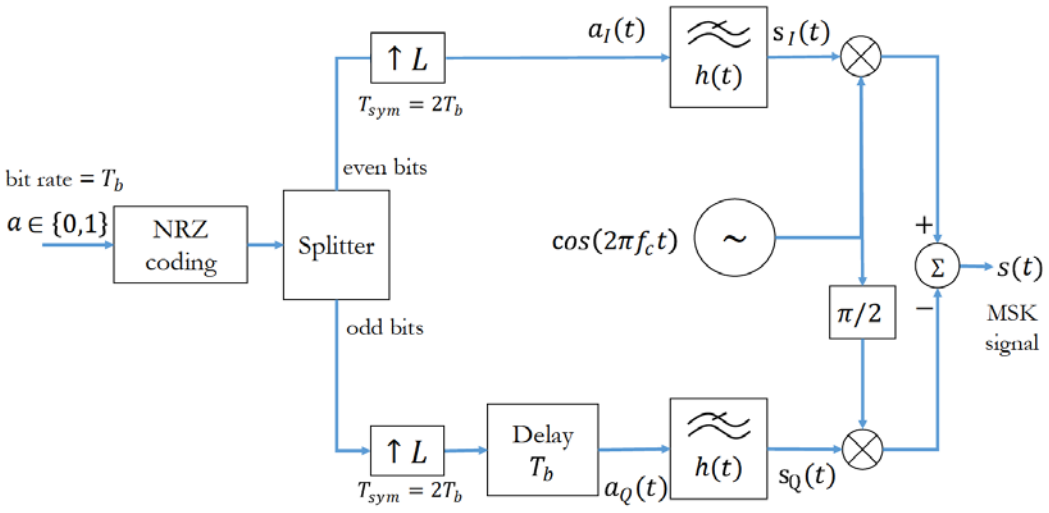

Рисунок 2.1 – Структурна модель MSK модулятора еквівалентного з OQPSK модуляцією

Наведемо функцію `msk_mod`, яка реалізує MSK модулятор, побудований за еквівалентністю з OQPSK модуляцією:

```
import numpy as np
import matplotlib.pyplot as plt

def gaussian_pulse(t, Ts):
    return np.exp(-t**2 / (2 * Ts**2)) / (np.sqrt(2 * np.pi) * Ts)
def msk_mod(bits, Ts, fc, Tb):
```

```
    # Параметри модуляції MSK
    Tb_half = Tb / 2
    Ts_half = Ts / 2
    N = len(bits)
    t = np.linspace(0, Tb, int(Tb * fc))

    # Формування імпульсів на синфазному та квадратурному плечах
    s_I = np.zeros(N * int(Tb * fc))
    s_Q = np.zeros(N * int(Tb * fc))
    for i in range(N):
        t_shifted = t - i * Tb
        pulse = gaussian_pulse(t_shifted, Ts_half)
        s_I[i * int(Tb * fc):(i+1) * int(Tb * fc)] = bits[i] * pulse
        s_Q[i * int(Tb * fc):(i+1) * int(Tb * fc)] = bits[i] * pulse if i % 2 == 0 else -
bits[i] * pulse

    # Сумування імпульсів на синфазному та квадратурному плечах
    s = s_I * np.cos(2 * np.pi * fc * t) - s_Q * np.sin(2 * np.pi * fc * t)

    return s, t

# Параметри модуляції
bits = np.random.randint(2, size=10)
Tb = 1  # Час тривалості бітового символу
Ts = Tb / 2  # Час тривалості напівперіоду
fc = 10  # Частота несучої

# Модуляція MSK
s, t = msk_mod(bits, Ts, fc, Tb)
# Візуалізація сигналу
plt.plot(t, s)
```

```
plt.xlabel('Час')
plt.ylabel('Амплітуда')
plt.title('MSK сигнал')
plt.grid(True)
plt.show()
```

Ця функція генерує MSK сигнал на основі переданих бітів і параметрів модуляції.

2.2 Моделювання продуктивності каналу передавання даних

Наведено скрипт на python для моделювання повного каналу зв'язку з вищезгаданим модулятором MSK, каналом AWGN і демодулятором MSK (msk.py). Змодельовані часові осцилограми на передавачі показано на рисунку 2.2. Код також включає моделювання продуктивності системи зв'язку MSK в діапазоні значень Eb=N0, що відносяться до каналу AWGN [6].

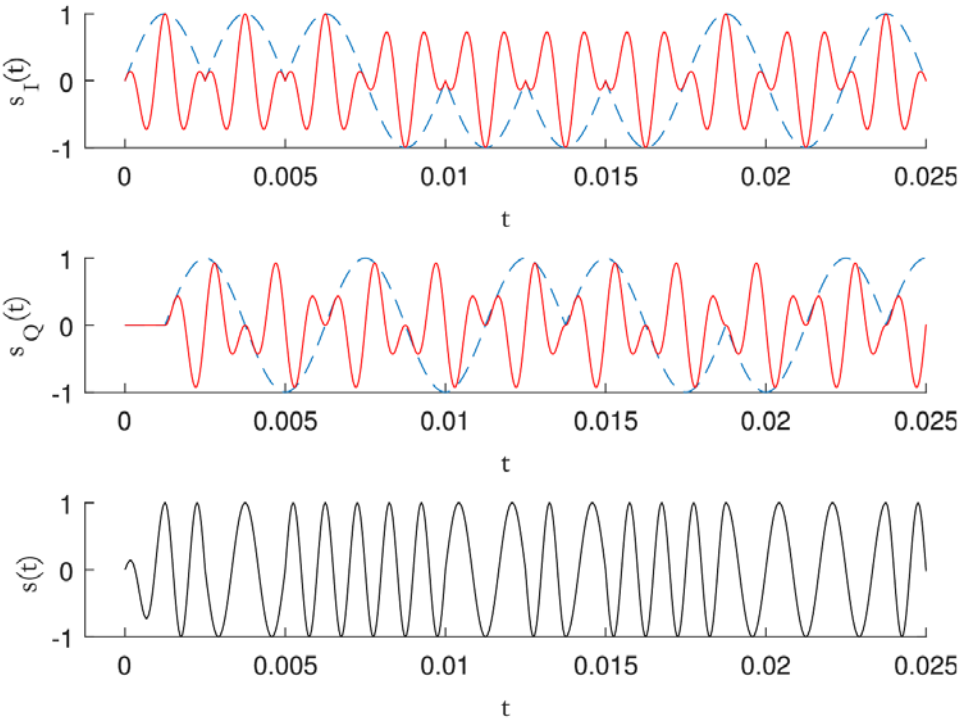

Рисунок 2.2 - Змодельовані форми сигналів MSK на стороні передавача

Модулятор MSK можна інтерпретувати як такий, що має два BPSK-подібні модулятори на плечах I-каналу та Q-каналу, і єдиною відмінністю є зсув на півсимволу з синусоїдальною формою на півциклу для MSK [7].

```
#Execute in Python3: exec(open("chapter_2/msk.py").read())
import numpy as np #for numerical computing
import matplotlib.pyplot as plt #for plotting functions
from DigiCommPy.passband_modulations import msk_mod,msk_demod
from DigiCommPy.channels import awgn
from scipy.special import erfc

N = 100000 # Number of symbols to transmit
EbN0dB = np.arange(start=-4,stop = 11,step = 2) # Eb/N0 range in dB for simulation
fc = 800 # carrier frequency in Hertz
OF = 32 # oversampling factor, sampling frequency will be fs=OF*fc

BER = np.zeros(len(EbN0dB)) # For BER values for each Eb/N0

a = np.random.randint(2, size=N) # uniform random symbols from 0's and 1's
result = msk_mod(a,fc,OF,enable_plot=True) # MSK modulation
s = result['s(t)']

for i,EbN0 in enumerate(EbN0dB):
    # Compute and add AWGN noise
    r = awgn(s,EbN0,OF) # refer Chapter section 4.1

    a_hat = msk_demod(r,N,fc,OF) #receiver

    BER[i] = np.sum(a!=a_hat)/N # Bit Error Rate Computation

theoreticalBER = 0.5*erfc(np.sqrt(10**(EbN0dB/10))) # Theoretical bit error rate

#-------------Plots--------------------------
fig, ax = plt.subplots(nrows=1,ncols = 1)
ax.semilogy(EbN0dB,BER,'k*',label='Simulated') # simulated BER
ax.semilogy(EbN0dB,theoreticalBER,'r-',label='Theoretical')
ax.set_xlabel(r'$E_b/N_0$ (dB)')
ax.set_ylabel(r'Probability of Bit Error - $P_b$')
ax.set_title(['Probability of Bit Error for MSK modulation'])
ax.legend();fig.show();
```

2.3 Квадратурна реалізація GMSK модулятора

Квадратурна реалізація GMSK (Gaussian Minimum Shift Keying) модулятора використовує комплексний сигнал для генерації сигналу GMSK. Наведемо приклад Python коду для реалізації квадратурного GMSK модулятора:

```
import numpy as np


def gaussian_filter(t, BT):
    """
    Gaussian filter function for GMSK modulation.
```

```
    t: Time array
    BT: BT product (bandwidth-time product)
    """
    return np.exp(-np.pi * t**2 / np.log(2) / BT**2)

def gmsk_mod(bits, samples_per_symbol, BT):
    """
    GMSK modulator.
    bits: Input bit sequence (0 or 1)
    samples_per_symbol: Number of samples per GMSK symbol
    BT: BT product (bandwidth-time product)
    """
    # Define Gaussian filter parameters
    t = np.linspace(-5, 5, samples_per_symbol)  # Time array for Gaussian filter
    h = gaussian_filter(t, BT)  # Gaussian filter coefficients

    # Initialize complex signal
    signal = np.zeros(len(bits) * samples_per_symbol, dtype=complex)

    # Generate GMSK modulation
    phase = 0
    for i, bit in enumerate(bits):
        if bit == 0:
            phase -= np.pi / 2
        else:
            phase += np.pi / 2
        signal[i*samples_per_symbol:(i+1)*samples_per_symbol] = np.exp(1j * phase) * h

    return signal
```

У цьому коді функція `gaussian_filter` визначає фільтр Гауса для GMSK модуляції, а функція `gmsk_mod` здійснює модуляцію GMSK на основі послідовності бітів вхідного сигналу. Результатом є комплексний сигнал, який може бути використаний для подальшого аналізу або передачі через канал зв'язку. Отже, квадратурний формат є одним з способів реалізації GMSK модулятора. В квадратурній реалізації GMSK використовуються комплексні сигнали для модуляції. Основна ідея полягає в тому, що бітова послідовність кодується відповідними фазовими змінами комплексного сигналу. Наведемо один можливий підхід для реалізації квадратурного GMSK модулятора за допомогою Python:

```
import numpy as np

def gaussian_filter(t, BT):
    """
    Gaussian filter function for GMSK modulation.
    t: Time array
    BT: BT product (bandwidth-time product)
    """
    return np.exp(-np.pi * t**2 / np.log(2) / BT**2)

def gmsk_mod_quad(bits, samples_per_symbol, BT):
    """
    Quadrature GMSK modulator.
    bits: Input bit sequence (0 or 1)
    samples_per_symbol: Number of samples per GMSK symbol
    BT: BT product (bandwidth-time product)
    """
    # Define Gaussian filter parameters
    t = np.linspace(-5, 5, samples_per_symbol)  # Time array for Gaussian filter
    h = gaussian_filter(t, BT)  # Gaussian filter coefficients
```

```
    # Initialize complex signals for I and Q channels
    signal_I = np.zeros(len(bits) * samples_per_symbol)
    signal_Q = np.zeros(len(bits) * samples_per_symbol)

    # Generate GMSK modulation
    phase = 0
    for i, bit in enumerate(bits):
        if bit == 0:
            phase -= np.pi / 2
        else:
            phase += np.pi / 2
        signal_I[i*samples_per_symbol:(i+1)*samples_per_symbol] = np.cos(phase) * h
        signal_Q[i*samples_per_symbol:(i+1)*samples_per_symbol] = np.sin(phase) * h

    return signal_I, signal_Q
```

У цьому коді функція `gmsk_mod_quad` створює квадратурні сигнали для каналів I та Q, які потім можуть бути використані для подальшої обробки або передачі через канал зв'язку. Квадратурний модулятор для GMSK, показаний на рисунку 2.3, легко отримати з представлення безперервної фазової модуляції [8].

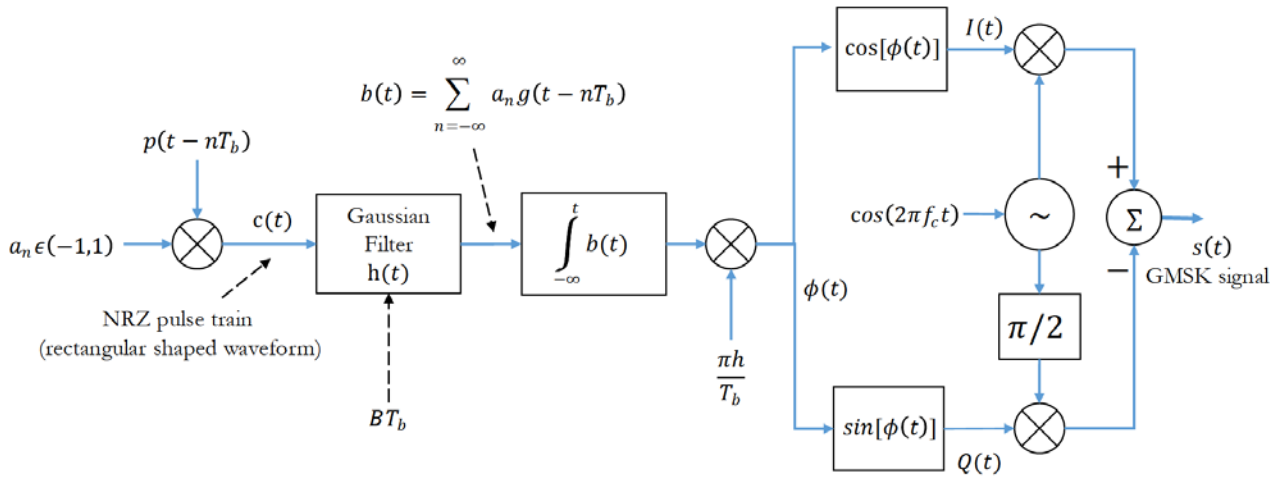

Рисунок 2.3 – Квадратурна реалізація GMSK модулятора

```
def gmsk_mod(a,fc,L,BT,enable_plot=False):
    """
    Function to modulate a binary stream using GMSK modulation
    Parameters:
        BT : BT product (bandwidth x bit period) for GMSK
        a : input binary data stream (0's and 1's) to modulate
        fc : RF carrier frequency in Hertz
        L : oversampling factor
        enable_plot: True = plot transmitter waveforms (default False)
    Returns:
        (s_t,s_complex) : tuple containing the following variables
            s_t : GMSK modulated signal with carrier s(t)
            s_complex : baseband GMSK signal (I+jQ)
    """
    from scipy.signal import upfirdn,lfilter

    fs = L*fc; Ts=1/fs;Tb = L*Ts; # derived waveform timing parameters
    c_t = upfirdn(h=[1]*L, x=2*a-1, up = L) #NRZ pulse train c(t)

    k=1 # truncation length for Gaussian LPF
    h_t = gaussianLPF(BT,Tb,L,k) # Gaussian LPF with BT=0.25
    b_t = np.convolve(h_t,c_t,'full') # convolve c(t) with Gaussian LPF to get b(t)
    bnorm_t = b_t/max(abs(b_t)) # normalize the output of Gaussian LPF to +/-1

    h = 0.5;
    # integrate to get phase information
    phi_t = lfilter(b = [1], a = [1,-1], x = bnorm_t*Ts) * h*np.pi/Tb

    I = np.cos(phi_t)
    Q = np.sin(phi_t) # cross-correlated baseband I/Q signals

s_complex = I - 1j*Q # complex baseband representation
t = Ts* np.arange(start = 0, stop = len(I)) # time base for RF carrier
sI_t = I*np.cos(2*np.pi*fc*t); sQ_t = Q*np.sin(2*np.pi*fc*t)
s_t = sI_t - sQ_t # s(t) - GMSK with RF carrier
```

```
if enable_plot:
    fig, axs = plt.subplots(2, 4)
    axs[0,0].plot(np.arange(0,len(c_t))*Ts,c_t);
    axs[0,0].set_title('c(t)');axs[0,0].set_xlim(0,40*Tb)
    axs[0,1].plot(np.arange(-k*Tb,k*Tb+Ts,Ts),h_t);
    axs[0,1].set_title('$h(t): BT_b$='+str(BT))
    axs[0,2].plot(t,I,'--');axs[0,2].plot(t,sI_t,'r');
    axs[0,2].set_title('$I(t)cos(2 \pi f_c t)$');axs[0,2].set_xlim(0,10*Tb)
    axs[0,3].plot(t,Q,'--');axs[0,3].plot(t,sQ_t,'r');
    axs[0,3].set_title('$Q(t)sin(2 \pi f_c t)$');axs[0,3].set_xlim(0,10*Tb)
    axs[1,0].plot( np.arange(0,len(bnorm_t))*Ts,bnorm_t);
    axs[1,0].set_title('b(t)');axs[1,0].set_xlim(0,40*Tb)
    axs[1,1].plot(np.arange(0,len(phi_t))*Ts, phi_t);
    axs[1,1].set_title('$\phi(t)$')
    axs[1,2].plot(t,s_t);axs[1,2].set_title('s(t)');
    axs[1,2].set_xlim(0,20*Tb)
    axs[1,3].plot(I,Q);axs[1,3].set_title('constellation')
    fig.show()
return (s_t,s_complex)
```

Функція gmsk mod реалізує квадратурний модулятор (для h = 0,5). Змодельовані осцилограми в різних точках модулятора наведено на рисунку 2.4 [9].

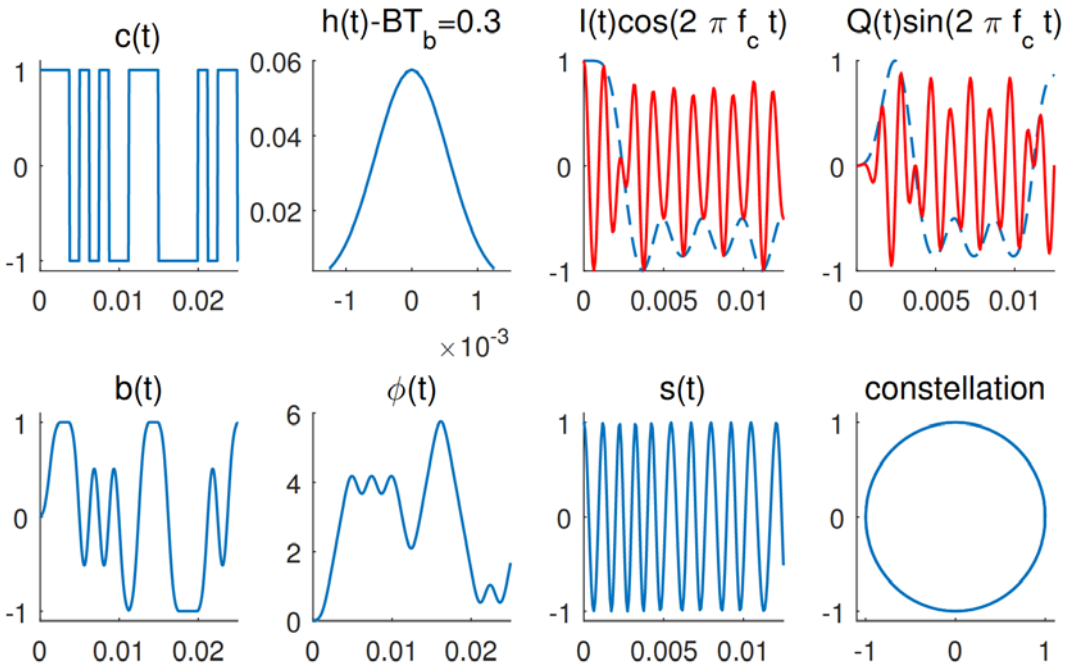

Рисунок 2.4 – Форми сигналів базової смуги в різних точках GMSK модулятора

2.4 Модулятор FSK сигналів

Дискретно-часова еквівалентна модель для генерування ортогонального BFSK сигналу показана на рисунку 2.5. Для генерації ортогонального BFSK сигналу у дискретно-часовій області, потрібно спочатку визначити параметри сигналу, такі як частоти несучих сигналів (f1 і f2), частота дискретизації (fs), кількість дискретних відліків у бітовому періоді (L) та тривалість сигналу. Далі можна використовувати ці параметри для генерації відповідних сигналів для кожного біта, використовуючи наприклад бінарний вигляд даних (0 або 1) для визначення, який сигнал виходить. Наведемо спрощений лістинг Python для генерації ортогонального BFSK сигналу:

```
import numpy as np

def generate_bfsk_signal(bits, fs, f1, f2, L, duration):
    """
    Generate orthogonal BFSK signal.

    bits: Input bit sequence (0 or 1)
    fs: Sampling frequency
    f1: Frequency of the first carrier
    f2: Frequency of the second carrier
    L: Number of samples per bit period
    duration: Duration of the signal (in seconds)
    """
    # Calculate time array
    t = np.linspace(0, duration, int(fs*duration))

    # Initialize BFSK signal
    signal = np.zeros(len(t))
```

```
# Generate BFSK modulation
for i, bit in enumerate(bits):
    if bit == 0:
        signal[i*L:(i+1)*L] = np.cos(2 * np.pi * f1 * t[i*L:(i+1)*L])
    else:
        signal[i*L:(i+1)*L] = np.cos(2 * np.pi * f2 * t[i*L:(i+1)*L])

return t, signal
```

Цей код генерує ортогональний BFSK сигнал для заданих параметрів. Потім його можна використовувати для симуляції та аналізу різних аспектів роботи системи зв'язку.

Функція bfsk mod реалізує дискретно-часову еквівалентну модель для генерації BFSK, показану на рисунку 2.5 [10].

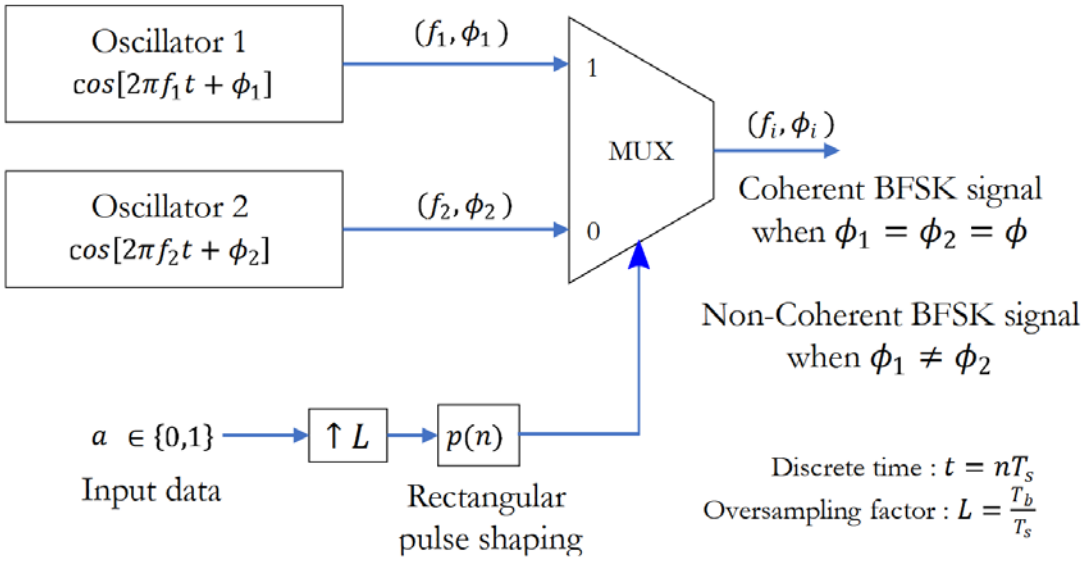

Рисунок 2.5 – Дискретна еквівалентна модель для BFSK модуляції

Вона підтримує генерацію як когерентної, так і некогерентної форми BFSK-сигналу. Якщо вибрано когерентне BFSK, функція повертає згенеровану початкову випадкову фазу f на додаток до згенерованого сигналу BFSK. Ця інформація про фазу має вирішальне значення для когерентного детектування на

приймачі. Якщо на модуляторі обрано некогерентну BFSK, то фазовий аргумент, що повертається функцією, не має значення на приймачі. Більше того, когерентно модульований BFSK сигнал також може бути некогерентно демодульований.

```
def bfsk_mod(a,fc,fd,L,fs,fsk_type='coherent',enable_plot = False):
    """
    Function to modulate an incoming binary stream using BFSK
    Parameters:
        a : input binary data stream (0's and 1's) to modulate
        fc : center frequency of the carrier in Hertz
        fd : frequency separation measured from Fc
        L : number of samples in 1-bit period
        fs : Sampling frequency for discrete-time simulation
        fsk_type : 'coherent' (default) or 'noncoherent' FSK generation
        enable_plot: True = plot transmitter waveforms (default False)
    Returns:
        (s_t,phase) : tuple containing following parameters
            s_t : BFSK modulated signal
            phase : initial phase generated by modulator, applicable only for
↪   coherent FSK. It can be used when using coherent detection at Rx
    """
    from scipy.signal import upfirdn
    a_t = upfirdn(h=[1]*L, x=a, up = L) #data to waveform

t = np.arange(start=0,stop=len(a_t))/fs #time base
if fsk_type.lower() == 'noncoherent':
    # carrier 1 with random phase
    c1 = np.cos(2*np.pi*(fc+fd/2)*t+2*np.pi*np.random.random_sample())
    # carrier 2 with random phase
    c2 = np.cos(2*np.pi*(fc-fd/2)*t+2*np.pi*np.random.random_sample())
else: #coherent is default
    # random phase from uniform distribution [0,2pi)
    phase=2*np.pi*np.random.random_sample()
    c1 = np.cos(2*np.pi*(fc+fd/2)*t+phase) # carrier 1 with random phase
   c2 = np.cos(2*np.pi*(fc-fd/2)*t+phase) # carrier 2 with the same random phase
s_t = a_t*c1 +(-a_t+1)*c2 # BFSK signal (MUX selection)

if enable_plot:
    fig, (ax1,ax2)=plt.subplots(2, 1);ax1.plot(t,a_t);ax2.plot(t,s_t);fig.show()
return (s_t,phase)
```

2.5 Висновки до розділу 2

Розроблені моделі надають можливість проводити детальний та ефективний аналіз різних алгоритмів модуляції. Це дозволяє вибрати оптимальні алгоритми

для конкретного застосування з урахуванням вимог щодо швидкості передачі, спектральної ефективності та мінімізації спотворень. Моделі дозволяють оцінити вплив різних алгоритмів модуляції на якість передачі даних в телекомунікаційних системах. Це допомагає попередньо прогнозувати якість зв'язку та вибирати найкращі рішення ще до їх впровадження.

Розроблені моделі дозволяють моделювати реальні умови передачі сигналу, включаючи шум, затухання та інші спотворення, що дозволяє проводити більш реалістичні тестування. Використання моделей дозволяє економити ресурси на тестуванні та вдосконаленні алгоритмів модуляції, оскільки це можна робити в віртуальному середовищі без потреби в реальних пристроях.

Отримані результати свідчать про важливість розробки моделей цифрових модуляторів для оптимізації телекомунікаційних систем та покращення їх функціональності та ефективності.

3 МОДЕЛЮВАННЯ ЦИФРОВИХ ПЕРЕДАВАЧІВ МОБІЛЬНИХ СИСТЕМ

3.1 Реалізація цифрових модемів за допомогою об'єктно-орієнтованого програмування

Визначимо загальний базовий клас `Modem`, який буде мати загальні методи для модуляції та демодуляції сигналів. Після цього можемо створити похідні класи для кожного конкретного методу модуляції. Наведемо приклад лістингу на Python для створення базового класу `Modem`:

```
class Modem:
    def __init__(self, symbol_rate, carrier_freq, modulation_index):
        self.symbol_rate = symbol_rate
        self.carrier_freq = carrier_freq
        self.modulation_index = modulation_index

    def modulate(self, data):
        pass

    def demodulate(self, signal):
        pass
```

Тепер ми можемо створити похідні класи для кожного методу модуляції, реалізуючи методи `modulate` і `demodulate` згідно з конкретними правилами модуляції для кожного методу.

Для прикладу, можна реалізувати клас для імпульсної амплітудної модуляції (PAM):

```
class PAM(Modem):
    def __init__(self, symbol_rate, carrier_freq, modulation_index):
        super().__init__(symbol_rate, carrier_freq, modulation_index)
```

```
    def modulate(self, data):
        # Implement PAM modulation
        pass

    def demodulate(self, signal):
        # Implement PAM demodulation
        pass
```

Аналогічно можна створити класи для інших методів модуляції (PSK, QAM, FSK), замінивши відповідні методи `modulate` і `demodulate` згідно з правилами цих методів. Такий підхід дозволяє легко розширювати систему для підтримки нових методів модуляції в майбутньому.

Функція-член modulate базового класу Modem реалізує загальну структуру для вищезгаданих модуляцій. Функція просто приймає вхідні інформаційні символи і вибирає відповідні модульовані символи з еталонного сузір'я для обраної техніки модуляції. Функція є спільною для методів модуляції PAM, PSK, QAM і FSK. Тому реалізації похідних класів для модемів PAM, PSK, QAM та FSK можуть просто успадковувати реалізацію від базового класу Modem, без необхідності окремої реалізації. Отже, визначимо базовий клас `Modem`, який буде містити загальні методи для модуляції та демодуляції сигналів. Ось як може виглядати цей клас:

```
class Modem:
    def __init__(self, symbol_rate, carrier_freq):
        self.symbol_rate = symbol_rate
        self.carrier_freq = carrier_freq

    def modulate(self, data):
        raise NotImplementedError("Method 'modulate' must be implemented in subclasses.")
```

def demodulate(self, signal):

raise NotImplementedError("Method 'demodulate' must be implemented in subclasses.")

Цей клас має два основні методи: `modulate`, який приймає дані для модуляції, та `demodulate`, який приймає сигнал для демодуляції. Обидва методи піднімають помилку `NotImplementedError`, оскільки вони будуть реалізовані в підкласах. Після чого ми можемо створити конкретні підкласи для кожного методу модуляції, які успадковують базовий клас `Modem` і реалізовують методи `modulate` і `demodulate` згідно з відповідними правилами модуляції та демодуляції [11].

```
import numpy as np
import abc
import matplotlib.pyplot as plt

class Modem:
    __metadata__ = abc.ABCMeta
    # Base class: Modem
    # Attribute definitions:
    #    self.M : number of points in the MPSK constellation
    #    self.name: name of the modem : PSK, QAM, PAM, FSK
    #    self.constellation : reference constellation

 #    self.coherence : only for 'coherent' or 'noncoherent' FSK
 def __init__(self,M,constellation,name,coherence=None): #constructor
     if (M<2) or ((M & (M -1))!=0): #if M not a power of 2
         raise ValueError('M should be a power of 2')
     if name.lower()=='fsk':
        if (coherence.lower()=='coherent') or (coherence.lower()=='noncoherent'):
             self.coherence = coherence
         else:
            raise ValueError('Coherence must be \'coherent\' or \'noncoherent\'')
     else:
         self.coherence = None
     self.M = M # number of points in the constellation
     self.name = name # name of the modem : PSK, QAM, PAM, FSK
     self.constellation = constellation # reference constellation
```

```
def plotConstellation(self):
    """
    Plot the reference constellation points for the selected modem
    """
    if self.name.lower()=='fsk':
        return 0 # FSK is multi-dimensional difficult to visualize

    fig, axs = plt.subplots(1, 1)
    axs.plot(np.real(self.constellation),np.imag(self.constellation),'o')
    for i in range(0,self.M):
        axs.annotate("{0:0{1}b}".format(i,self.M),
         ↪  (np.real(self.constellation[i]),np.imag(self.constellation[i])))

    axs.set_title('Constellation');
    axs.set_xlabel('I');axs.set_ylabel('Q');fig.show()

def modulate(self,inputSymbols):
    """
    Modulate a vector of input symbols (numpy array format) using the
    chosen modem. Input symbols take integer values in the range 0 to M-1.
    """
    if isinstance(inputSymbols,list):
        inputSymbols = np.array(inputSymbols)

    if  not (0 <= inputSymbols.all() <= self.M-1):
        raise ValueError('inputSymbols values are beyond the range 0 to M-1')

    modulatedVec = self.constellation[inputSymbols]
    return modulatedVec #return modulated vector
```

3.2 Лістинг програми формування M-PAM модуляції

Для модуляції M-PAM амплітуда сигналу залежить від значення символу, а не від фази. Отже, нам не потрібно використовувати квадратурну складову. Давайте врахуємо це в нашому класі`MPAMModem`:

```
import numpy as np

class MPAMModem(Modem):
    def __init__(self, symbol_rate, carrier_freq, levels):
        super().__init__(symbol_rate, carrier_freq)
        self.levels = levels

    def modulate(self, data):
```

```
        symbols = np.array(data)
        amplitude_levels = np.linspace(-1, 1, self.levels)
        modulated_signal = amplitude_levels[symbols] * np.sin(2 * np.pi * self.carrier_freq * np.arange(len(symbols)))
        return modulated_signal

    def demodulate(self, signal):
        amplitude_levels = np.linspace(-1, 1, self.levels)
        demodulated_signal = np.zeros(len(signal), dtype=int)
        for i, sample in enumerate(signal):
            closest_symbol = np.argmin(np.abs(sample - amplitude_levels))
            demodulated_signal[i] = closest_symbol
        return demodulated_signal
```

У цьому коді ми використовуємо `amplitude_levels`, щоб розділити амплітудні рівні на `levels` частин. Потім ми вибираємо амплітуду для кожного символу з відповідного рівня `amplitude_levels` і модулюємо сигнал цією амплітудою. При демодуляції ми визначаємо, який амплітудний рівень найближчий до кожного вхідного зразка сигналу.

Створимо підклас для модема імпульсно-амплітудної модуляції (M-PAM) на основі базового класу `Modem`. У цьому класі ми реалізуємо методи модуляції та демодуляції для M-PAM.

```
import numpy as np

class MPAMModem(Modem):
    def __init__(self, symbol_rate, carrier_freq, levels):
        super().__init__(symbol_rate, carrier_freq)
        self.levels = levels

    def modulate(self, data):
```

```
        symbols = np.array(data)
        modulated_signal = symbols * np.sin(2 * np.pi * self.carrier_freq * np.arange(len(symbols)))
        return modulated_signal

    def demodulate(self, signal):
        demodulated_signal = np.sign(signal)
        return demodulated_signal
```

У цьому класі ми використовуємо синусоїдальну хвилю для модуляції, помножуючи дані на частоту несучої хвилі. Для демодуляції ми просто використовуємо функцію sign для визначення напрямку сигналу.

Необхідно зауважити, що ми передаємо параметр `levels`, який вказує на кількість рівнів сигналу M-PAM. Наприклад, якщо `levels = 2`, це буде реалізація модема BPSK. Для модема QPSK потрібно `levels = 4`, і т.д.

Значення M залежить від параметра k - кількості бітів, які ми хочемо втиснути в один символ M-PAM. Наприклад, якщо в один символ передачі втиснути 3 біти (k = 3), то $M = 2^K = 2^3 = 8$, що призводить до конфігурації 8-PAM. Для сигналу M-PAM точки сузір'я розташовані в точках ±1; ±3;:::; ± (M -1), як показано на рисунку 3.3 [12].

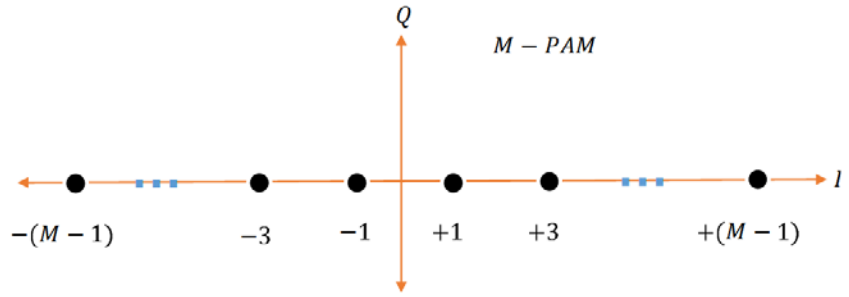

Рисунок 3.1 – Сузір'я M-PAM

```
class PAMModem(Modem):
    # Derived class: PAMModem
    def __init__(self, M):
        m = np.arange(0,M) #all information symbols m={0,1,...,M-1}
        constellation = 2*m+1-M + 1j*0  # reference constellation
        Modem.__init__(self, M, constellation, name='PAM') #set the modem attributes
```

3.3 Модем з фазовою маніпуляцією (M-PSK) сигналів

Модем з фазовою маніпуляцією (M-PSK) є пристроєм зв'язку, який використовує фазову маніпуляцію для передачі даних. У сигналів M-PSK інформація кодується шляхом зміни фази несучої хвилі. У даному випадку "M" відноситься до кількості можливих фазових станів, які може приймати несуча хвиля. У цьому модемі фазовий кут несучої хвилі змінюється відповідно до передаваних даних. Наприклад, якщо M=4, то можливі фазові стани будуть розділені на 4 рівні, зазвичай 0, 90, 180 та 270 градусів. Цей тип модема широко використовується у бездротових системах зв'язку, таких як Wi-Fi, мобільний зв'язок, радіозв'язок, де важливо швидко та ефективно передавати дані через канал зв'язку. Його використання дозволяє забезпечити ефективність передачі даних при обмеженні ресурсів каналу та мінімізації спотворень сигналу.

Модулятор M-PSK використовує фазову маніпуляцію для передачі даних. Для будь-якого значення M (кількість фазових станів) фази розділяються на M рівнів, рівномірно розподілені навколо кола в комплексній площині. Кожен символ даних відображається на одному з цих фазових станів. Наприклад, для M = 4, можна використовувати фази 0°, 90°, 180° та 270°. Для M = 8, фази можуть бути розміщені кожні 45°. Для M = 16, фази розміщуються кожні 22.5°.

Ідеальне сузір'я для таких модуляцій показує, як рівномірно розподілені фазові стани навколо кола в комплексній площині (рис. 3.2) [1].

Це спрощена постановка. У реальних системах ідеальна рівномірність розподілу може бути порушена через різні фактори, такі як спотворення каналу, шум, помилки у вимірюваннях тощо.

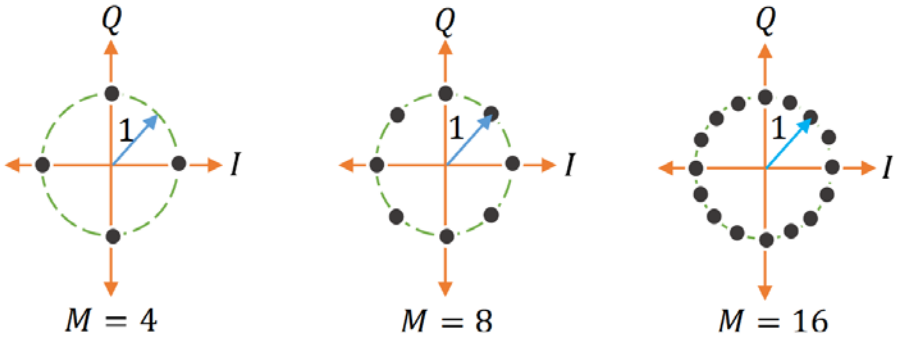

Рисунок 3.2 – Сузір'я M-PSK

Отже, PSK-модеми можна класифікувати як один з класів модемів, які використовують фазову маніпуляцію для передачі даних. Фазова маніпуляція зазвичай використовується для модуляції цифрових даних в аналогових сигнали для передачі через канал зв'язку. PSK-модеми можуть використовуватися для широкого спектру застосувань, включаючи бездротовий зв'язок, супутникові комунікації, мобільний зв'язок, інтернет через радіо та багато інших. Вони надають ефективний спосіб передачі цифрових даних, використовуючи властивості фазових змін сигналу. Крім PSK, існують інші класи модемів, такі як Amplitude Shift Keying (ASK) і Frequency Shift Keying (FSK), які використовують зміни амплітуди або частоти сигналу відповідно для кодування даних.

```
class PSKModem(Modem):
    # Derived class: PSKModem
    def __init__(self, M):
        #Generate reference constellation
        m = np.arange(0,M) #all information symbols m={0,1,...,M-1}
        I = 1/np.sqrt(2)*np.cos(m/M*2*np.pi)
        Q = 1/np.sqrt(2)*np.sin(m/M*2*np.pi)
        constellation = I + 1j*Q #reference constellation
        Modem.__init__(self, M, constellation, name='PSK') #set the modem attributes
```

3.3 Модем з квадратурною амплітудною модуляцією (M-QAM)

У модуляції M-QAM (Quadrature Amplitude Modulation), інформаційні біти кодуються як зміни як амплітуди, так і фази сигналу. Модулятор M-QAM передає послідовність інформаційних символів, які вибрані зі множини. Щоб обмежити помилкові рішення приймача до однобітових помилок, інформаційні символи кодуються за кодом Грея, в якому сусідні символи відрізняються лише одним бітом. Це допомагає знизити ймовірність помилкового сприйняття відомостей приймачем. Модуляція M-QAM широко використовується у бездротових комунікаційних системах, таких як Wi-Fi, LTE, WiMAX, та в інших сучасних цифрових системах передачі даних. Вона дозволяє ефективно використовувати частотний спектр та забезпечує високу швидкість передачі даних.

Існують різні форми сузір'їв, і деякі з них можуть бути більш ефективними в різних ситуаціях. Прямокутні (квадратні) сузір'я, такі як QAM, часто використовуються через їхню простоту в реалізації модуляції та демодуляції.

Прямокутні QAM-сузір'я можна розглядати як комбінацію амплітудної модуляції змінної фази (наприклад, PSK) та фазової модуляції змінної амплітуди (наприклад, ASK). Наприклад, для стандартного сузір'я 16-QAM точки можуть бути згенеровані з двох сигналів 4-PAM, а для 64-QAM - з двох сигналів 8-PAM.

Інші форми сузір'їв, такі як кругові або трикутні, можуть мати свої переваги, такі як краща енергетична ефективність або краща стійкість до помилок. Однак вони можуть вимагати більш складних алгоритмів модуляції та демодуляції. Вибір форми сузір'я залежить від конкретних вимог до системи зв'язку, таких як швидкість передачі даних, вимоги до шуму та інші обмеження.

Модулятор M-QAM реалізовано як похідний клас від базового класу Modem. Реалізація модулятора динамічно генерує точки сузір'я M-QAM на основі карти Карно з обходом коду Грея типу 1. Отримані ідеальні сузір'я для кодів Грея 16-QAM і 64-QAM показані на рисунку 3.3 [2].

Сузір'я сигнального простору для 16-QAM і 64-QAM можна зобразити на комплексній площині, де вісь X відповідає ін-фазній (In-phase) компоненті сигналу, а вісь Y - квадратурній (Quadrature) компоненті. Кожна точка в цьому просторі представляє собою один символ, і його координати визначаються

амплітудою і фазою сигналу. Для 16-QAM, у сузір'ї є 16 точок, які рівномірно розподілені в комплексній площині. Вони можуть бути організовані у вигляді 4 рядків по 4 точки або будь-якої іншої конфігурації. Для 64-QAM, сузір'я містить 64 точки, що також рівномірно розподілені в комплексній площині. Вони можуть бути організовані, наприклад, у вигляді 8 рядків по 8 точок. Такі сузір'я дозволяють ефективно використовувати частотний спектр для передачі даних, дозволяючи одночасно передавати багатобітові символи. Однак, вони можуть бути більш чутливими до шуму та спотворень, тому важливо використовувати ефективні методи демодуляції та корекції помилок для отримання надійного прийому сигналу [3].

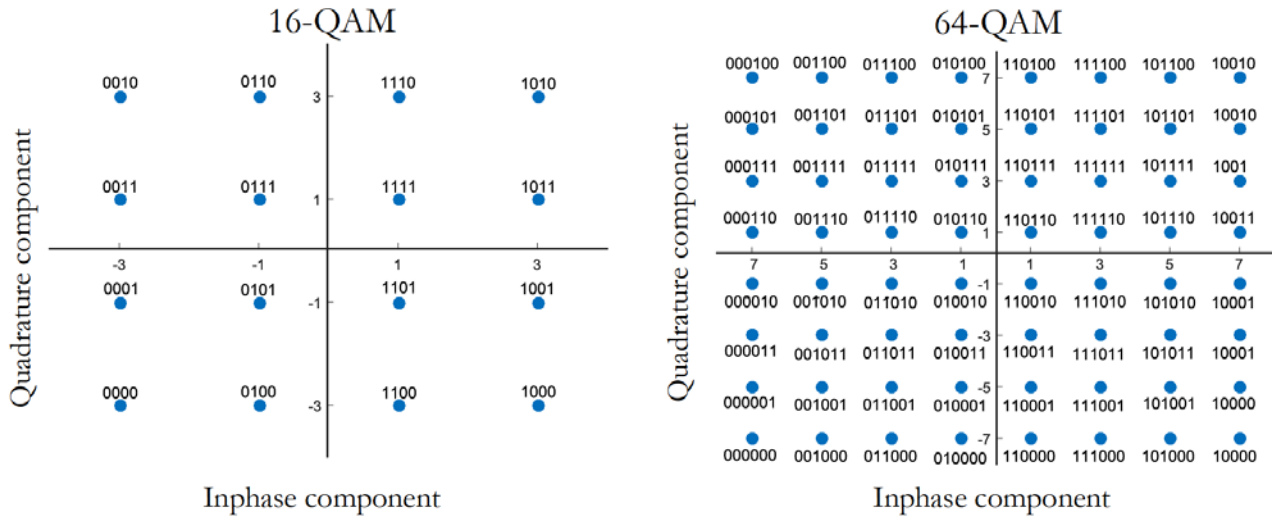

Рисунок 3.3 - Сузір'я сигнального простору для 16-QAM і 64-QAM

QAM-модем можна реалізувати як похідний клас від базового класу модему. В об'єктно-орієнтованому програмуванні такий підхід дозволяє успадковувати загальні функції та властивості базового класу та додавати до них нові функції та властивості, специфічні для QAM-модему.

Наприклад, базовий клас модему може містити загальні методи для модуляції та демодуляції сигналів, керування параметрами зв'язку та інші загальні функції. Похідний клас QAM-модему може додати специфічні методи для генерації сузір'їв сигнального простору, обробки QAM-сигналів та інші функції, які характерні саме для QAM-модуляції. Такий підхід дозволяє

структурувати код, робить його більш зрозумілим та легшим для розширення і підтримки в майбутньому. Крім того, він дозволяє використовувати загальні рішення для всіх класів модемів, забезпечуючи зручну роботу з різними видами модуляції.

```
class QAMModem(Modem):
    # Derived class: QAMModem
    def __init__(self,M):
        if (M==1) or (np.mod(np.log2(M),2)!=0): # M not a even power of 2
           raise ValueError('Only square MQAM supported. M must be even power of 2')

        n = np.arange(0,M) # Sequential address from 0 to M-1 (1xM dimension)
        a = np.asarray([x^(x>>1) for x in n]) #convert linear addresses to Gray code
        D = np.sqrt(M).astype(int) #Dimension of K-Map - N x N matrix
        a = np.reshape(a,(D,D)) # NxN gray coded matrix
        oddRows=np.arange(start = 1, stop = D ,step=2) # identify alternate rows

a[oddRows,:] = np.fliplr(a[oddRows,:]) #Flip rows - KMap representation
nGray=np.reshape(a,(M)) # reshape to 1xM - Gray code walk on KMap

#Construction of ideal M-QAM constellation from sqrt(M)-PAM
(x,y)=np.divmod(nGray,D) #element-wise quotient and remainder
Ax=2*x+1-D # PAM Amplitudes 2d+1-D - real axis
Ay=2*y+1-D # PAM Amplitudes 2d+1-D - imag axis
constellation = Ax+1j*Ay
Modem.__init__(self, M, constellation, name='QAM') #set the modem attributes
```

3.4 Висновки до розділу 3

Моделювання дозволило вдосконалити процеси передачі даних у мобільних мережах. Це означає, що тепер можна передавати дані швидше та ефективніше, що покращує загальну продуктивність мобільних комунікацій. Моделювання дозволило ідентифікувати потенційні проблеми та вразливі місця в мобільних системах перед їх реальним впровадженням. Це допомагає забезпечити більш високу надійність мобільних комунікацій та зменшити ймовірність виникнення помилок. Моделювання дозволяє ефективно використовувати ресурси мобільних систем, такі як пропускна здатність мережі та потужність передавачів. Це дозволяє забезпечити максимальну ефективність роботи мобільних мереж та оптимізувати їхні ресурси.

Результати моделювання можуть служити основою для розвитку нових технологій та стандартів зв'язку у мобільних системах. Це може призвести до

появи нових функцій та можливостей для користувачів мобільних комунікацій. Загалом, моделювання цифрових передавачів у мобільних системах сприяє покращенню ефективності та надійності мобільних комунікацій, що є важливим для розвитку сучасних телекомунікаційних систем.

4 ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ ТЕЛЕКОМУНІКАЦІЙНИХ МОДЕМІВ

Об'єктно-орієнтоване програмування (ООП) є потужним підходом до розробки програмного забезпечення, що дозволяє організовувати код у вигляді об'єктів, які мають властивості та методи. Це особливо корисно для реалізації телекомунікаційних модемів, де можна використовувати класи для представлення різних типів модемів та їх функціональності. Розглянемо деякі можливі класи та їх взаємозв'язки для представлення телекомунікаційних модемів:

Базовий клас модему може містити загальні властивості та методи, які спільні для всіх типів модемів, такі як підключення до мережі, передача даних, отримання даних тощо. Похідні класи для конкретних типів модемів. Кожен з цих класів може містити специфічні властивості та методи, які характерні для конкретного типу модему. Класи для роботи з підключенням та комунікацією для взаємодії з мережею, передачі та отримання даних через модем. Інші допоміжні класи можуть включати класи для кодування та декодування даних, обробки помилок, керування параметрами зв'язку [4].

Наведемо приклад простого коду на Python для реалізації базового класу модему:

```
class Modem:
    def __init__(self):
        self.connected = False

    def connect(self):
        # Код для підключення до мережі
        self.connected = True

    def disconnect(self):
        # Код для відключення від мережі
```

```
        self.connected = False

    def send_data(self, data):
        # Код для передачі даних
        pass

    def receive_data(self):
        # Код для отримання даних
        pass
```

Можна розширити цей базовий клас та створити похідні класи для реалізації різних типів модемів з більш конкретною функціональністю.

4.1 Інсталяція модемів у Python

Розглянемо створення класів для різних типів модемів у Python:

```
class Modem:
    def __init__(self, model):
        self.model = model
        self.connected = False

    def connect(self):
        # Код для підключення до мережі
        self.connected = True

    def disconnect(self):
        # Код для відключення від мережі
        self.connected = False
    def send_data(self, data):
        # Код для передачі даних
```

```
        pass

    def receive_data(self):
        # Код для отримання даних
        pass

class DSLModem(Modem):
    def __init__(self, model):
        super().__init__(model)
        self.dsl_speed = None

    def set_speed(self, speed):
        self.dsl_speed = speed

class CableModem(Modem):
    def __init__(self, model):
        super().__init__(model)
        self.channel = None

    def set_channel(self, channel):
        self.channel = channel

class MobileModem(Modem):
    def __init__(self, model):
        super().__init__(model)
        self.signal_strength = None

    def check_signal_strength(self):
        # Код для перевірки сили сигналу мобільної мережі
        pass
```

```
# Приклад використання:

dsl_modem = DSLModem("DSL-123")
dsl_modem.connect()
dsl_modem.set_speed("10 Mbps")
print(dsl_modem.connected)  # True
print(dsl_modem.dsl_speed)  # 10 Mbps
dsl_modem.disconnect()
print(dsl_modem.connected)  # False

cable_modem = CableModem("Cable-456")
cable_modem.connect()
cable_modem.set_channel(42)
print(cable_modem.connected)  # True
print(cable_modem.channel)    # 42
cable_modem.disconnect()
print(cable_modem.connected)  # False

mobile_modem = MobileModem("Mobile-789")
mobile_modem.connect()
mobile_modem.check_signal_strength()
print(mobile_modem.connected)  # True
```

У цьому прикладі ми створили базовий клас `Modem`, а також похідні класи `DSLModem`, `CableModem` та `MobileModem`. Кожен з цих класів спеціалізується на певному типі модему і має свою власну функціональність, яка може включати унікальні методи та властивості.

```python
import numpy as np
import abc
import matplotlib.pyplot as plt

class Modem:
    __metadata__ = abc.ABCMeta
    # Base class: Modem
    # Attribute definitions:
    #    self.M : number of points in the MPSK constellation
    #    self.name: name of the modem : PSK, QAM, PAM, FSK
    #    self.constellation : reference constellation
    #    self.coherence : only for 'coherent' or 'noncoherent' FSK
    def __init__(self,M,constellation,name,coherence=None): #constructor
        if (M<2) or ((M & (M -1))!=0): #if M not a power of 2
            raise ValueError('M should be a power of 2')
        if name.lower()=='fsk':
           if (coherence.lower()=='coherent') or (coherence.lower()=='noncoherent'):
                self.coherence = coherence
            else:
               raise ValueError('Coherence must be \'coherent\' or \'noncoherent\'')
        else:
            self.coherence = None
        self.M = M # number of points in the constellation
        self.name = name # name of the modem : PSK, QAM, PAM, FSK
        self.constellation = constellation # reference constellation

    def plotConstellation(self):
        """
        Plot the reference constellation points for the selected modem
        """
        from math import log2
        if self.name.lower()=='fsk':
            return 0 # FSK is multi-dimensional difficult to visualize

     fig, axs = plt.subplots(1, 1)
     axs.plot(np.real(self.constellation),np.imag(self.constellation),'o')
     for i in range(0,self.M):
         axs.annotate("{0:0{1}b}".format(i,int(log2(self.M))),
          ↪  (np.real(self.constellation[i]),np.imag(self.constellation[i])))

     axs.set_title('Constellation');
     axs.set_xlabel('I');axs.set_ylabel('Q');fig.show()

 def modulate(self,inputSymbols):
     """
         Modulate a vector of input symbols (numpy array format) using the chosen
          modem. The input symbols take integer values in the range 0 to M-1.
     """
     if isinstance(inputSymbols,list):
         inputSymbols = np.array(inputSymbols)

     if  not (0 <= inputSymbols.all() <= self.M-1):
        raise ValueError('Values for inputSymbols are beyond the range 0 to M-1')

     modulatedVec = self.constellation[inputSymbols]
     return modulatedVec #return modulated vector
```

```
class PAMModem(Modem):
    # Derived class: PAMModem
    def __init__(self, M):
        m = np.arange(0,M) #all information symbols m={0,1,...,M-1}
        constellation = 2*m+1-M + 1j*0  # reference constellation
        Modem.__init__(self, M, constellation, name='PAM') #set the modem attributes

class PSKModem(Modem):
    # Derived class: PSKModem
    def __init__(self, M):
        #Generate reference constellation
        m = np.arange(0,M) #all information symbols m={0,1,...,M-1}
        I = 1/np.sqrt(2)*np.cos(m/M*2*np.pi)
        Q = 1/np.sqrt(2)*np.sin(m/M*2*np.pi)
        constellation = I + 1j*Q #reference constellation
        Modem.__init__(self, M, constellation, name='PSK') #set the modem attributes

class QAMModem(Modem):
    # Derived class: QAMModem
    def __init__(self,M):
        if (M==1) or (np.mod(np.log2(M),2)!=0): # M not a even power of 2
            raise ValueError('Only square MQAM supported. M must be even power of 2')

        n = np.arange(0,M) # Sequential address from 0 to M-1 (1xM dimension)
        a = np.asarray([x^(x>>1) for x in n]) #convert linear addresses to Gray code
        D = np.sqrt(M).astype(int) #Dimension of K-Map - N x N matrix
        a = np.reshape(a,(D,D)) # NxN gray coded matrix
        oddRows=np.arange(start = 1, stop = D ,step=2) # identify alternate rows
        a[oddRows,:] = np.fliplr(a[oddRows,:]) #Flip rows - KMap representation
        nGray=np.reshape(a,(M)) # reshape to 1xM - Gray code walk on KMap

        #Construction of ideal M-QAM constellation from sqrt(M)-PAM
        (x,y)=np.divmod(nGray,D) #element-wise quotient and remainder
        Ax=2*x+1-D # PAM Amplitudes 2d+1-D - real axis
        Ay=2*y+1-D # PAM Amplitudes 2d+1-D - imag axis
        constellation = Ax+1j*Ay
        Modem.__init__(self, M, constellation, name='QAM') #set the modem attributes

class FSKModem(Modem):
    # Derivied class: FSKModem

 def __init__(self,M,coherence='coherent'):
     if coherence.lower()=='coherent':
         phi= np.zeros(M) # phase=0 for coherent detection
     elif coherence.lower()=='noncoherent':
         phi = 2*np.pi*np.random.rand(M) # M random phases in the (0,2pi)
     else:
         raise ValueError('Coherence must be \'coherent\' or \'noncoherent\'')
     constellation = np.diag(np.exp(1j*phi))
     Modem.__init__(self, M, constellation,
      ↪  name='FSK',coherence=coherence.lower()) #set the base modem attributes
```

Класи, визначені у файлі modem.py, є лише логічними сутностями, які визначають поведінку (методи) та властивості (атрибути) різних модемів. Необхідний клас модему потрібно створити екземпляр, перш ніж його можна буде використовувати у подальшому програмуванні. Під час створення

екземплярів класів модемів створюється об'єкт модему, який є сутністю зі специфічною характеристикою або функцією. Наступний код демонструє екземпляр об'єкта модему 16-PSK і те, як можна викликати його функції-члени для виконання модуляції та демодуляції [5].

```
>> import sys
>> sys.path.append('path_where_digicommpy_resides') #search path
>> from DigiCommPy.modem import PSKModem #import the PSKModem class from modem.py
>> M = 16 #16 points in the constellation
>> pskModem = PSKModem(M) #create a 16-PSK modem object
>> pskModem.plotConstellation() #plot ideal constellation for this modem
>> import numpy as np # for numerical computing
>> nSym = 10 #10 symbols as input to PSK modem
>> inputSyms = np.random.randint(low=0, high = M, size=nSym) # uniform random
 ↪  symbols from 0 to M-1
array([10, 14,  1,  0,  0,  1, 10,  0, 14,  5])
>> modulatedSyms = pskModem.modulate(inputSyms) #modulate
array([-0.5-0.5j, 0.5-0.5j , 0.65+0.27j, 0.707+0.j, 0.707+0.j, 0.65+0.27j,
 ↪  -0.5-0.5j,  0.707+0.j, 0.5-0.5j, -0.27+0.65j])
>> detectedSyms = pskModem.demodulate(modulatedSyms) #demodulate
array([10, 14,  1,  0,  0,  1, 10,  0, 14,  5], dtype=int64)
```

Наведемо код для створення моделі модему 16-PSK із застуванням його функції-членів для модуляції та демодуляції:

```
# Імпортуємо модуль modem.py, який містить визначені класи модемів
from modem import Modem16PSK

# Створюємо екземпляр об'єкта модему 16-PSK
modem_16psk = Modem16PSK()

# Викликаємо функції-члени для модуляції та демодуляції
modulated_signal = modem_16psk.modulate(data)
demodulated_data = modem_16psk.demodulate(received_signal)
```

У цьому коді ми спочатку імпортуємо клас `Modem16PSK` з файлу `modem.py`, який містить визначення класу для модему 16-PSK. Потім ми створюємо екземпляр цього класу, `modem_16psk`, і викликаємо його функції-члени `modulate` та `demodulate` для модуляції та демодуляції сигналу

відповідно. Цей приклад демонструє, як можна використовувати класи, визначені у файлі `modem.py`, для створення об'єктів модемів і виклику їх методів для виконання певних операцій з передачі та отримання даних.

4.2 Програмне моделювання ефективності цифрових передавачів

Виконаємо лістинг, який реалізує уніфікований підхід до моделювання різних типів модуляції (MPSK, MQAM, MPAM або MFSK). Він використовує метод Монте-Карло для симуляції передачі та прийому сигналу з вибраною модуляцією.

Алгоритм лістингу:

1. Код перевіряє змінну, яка вказує на тип модуляції, і вибирає відповідну схему модуляції (PAM, PSK, QAM або FSK).

2. За допомогою обраної схеми модуляції генеруються випадкові символи або деякий заданий потік даних для передачі.

3. Створюються модульовані сигнали згідно з обраною схемою модуляції.

4. Шум може бути доданий до модульованого сигналу для моделювання каналу передачі.

5. Симулюється процес прийому сигналу та його демодуляція, яка включає виявлення та декодування сигналу.

6. Порівнюється отриманий результат з переданим для визначення частоти помилок символів.

Це загальний огляд функціоналу запропонованого лістингу на мові Python:

```
import numpy as np

class ModulationSimulator:
    def __init__(self, modulation_type):
        self.modulation_type = modulation_type

    def generate_symbols(self, num_symbols):
```

```
        # Генеруємо символи для модуляції залежно від типу модуляції
        if self.modulation_type == 'MPSK':
            symbols = # Генерація символів для MPSK
        elif self.modulation_type == 'MQAM':
            symbols = # Генерація символів для MQAM
        elif self.modulation_type == 'MPAM':
            symbols = # Генерація символів для MPAM
        elif self.modulation_type == 'MFSK':
            symbols = # Генерація символів для MFSK
        else:
            raise ValueError("Unsupported modulation type")

        return symbols

    def simulate_transmission(self, num_symbols, snr):
        # Генеруємо символи для передачі
        symbols = self.generate_symbols(num_symbols)

        # Додаємо шум до сигналу
        noisy_symbols = self.add_noise(symbols, snr)

        # Демодуляція та оцінка помилок

    def add_noise(self, symbols, snr):
        # Додаємо шум до сигналу
        noise_power = 10 ** (-snr / 10)
        noise = np.random.normal(0, np.sqrt(noise_power), symbols.shape)
        noisy_symbols = symbols + noise
        return noisy_symbols

# Приклад використання:
```

```
simulator = ModulationSimulator('MPSK')
simulator.simulate_transmission(1000, 10)
```

У прикладі ми визначаємо клас `ModulationSimulator`, який приймає тип модуляції при створенні. У методі `simulate_transmission` ми генеруємо символи для передачі, додаємо шум до сигналу та виконуємо демодуляцію для оцінки помилок. Також необхідно враховувати, що це лише загальна структура, і реалізація генерації символів, додавання шуму та демодуляції буде залежати від конкретних алгоритмів та форматів модуляції, які ви хочете моделювати.

Визначення ефективності модуляції в каналі з випадковим білим гауссівським шумом (AWGN) часто вимірюється за допомогою таких метрик, як BER (Bit Error Rate) або SER (Symbol Error Rate). Ці метрики вказують на частоту помилок бітів або символів під час передачі через канал. Для оцінки ефективності модуляції в AWGN можна використовувати симуляційні методи, такі як метод Монте-Карло, який ви описали. У таких симуляціях генеруються випадкові символи або біти для передачі, модулюються, додається шум, а потім сигнал демодулюється, і обчислюється кількість помилок. Цей процес дозволяє оцінити ефективність різних типів модуляції в каналі з шумом і визначити, яка модуляція є найбільш ефективною для ваших конкретних умов передачі.

Програма моделювання автоматично вибирає обраний тип модуляції, виконує моделювання методом Монте-Карло, обчислює частоту помилок символів і порівнює її з теоретичною частотою помилок символів. Результати моделювання продуктивності, отримані для різних видів модуляції, показано на рисунку 4.1 [6].

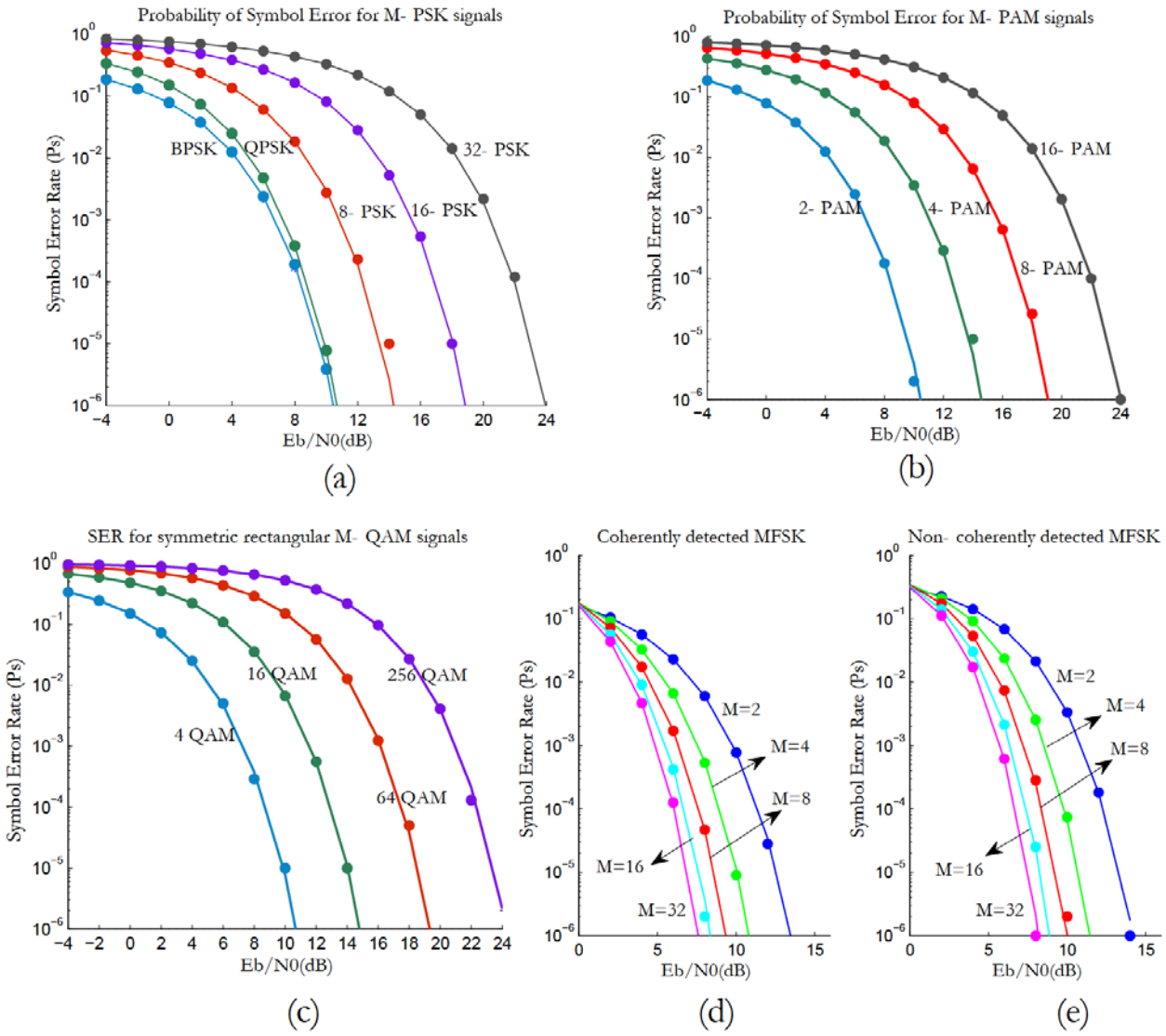

Рисунок 4.1 - Ефективність роботи цифрових передавачів в каналі AWGN

```
import numpy as np # for numerical computing
import matplotlib.pyplot as plt # for plotting functions
from matplotlib import cm # colormap for color palette
from scipy.special import erfc
from DigiCommPy.modem import PSKModem,QAMModem,PAMModem,FSKModem
from DigiCommPy.channels import awgn
from DigiCommPy.errorRates import ser_awgn

#---------Input Fields------------------------
nSym = 10**6 # Number of symbols to transmit
EbN0dBs = np.arange(start=-4,stop = 12, step = 2) # Eb/N0 range in dB for simulation
mod_type = 'FSK' # Set 'PSK' or 'QAM' or 'PAM' or 'FSK'
arrayOfM = [2,4,8,16,32] # array of M values to simulate
#arrayOfM=[4,16,64,256] # uncomment this line if MOD_TYPE='QAM'
```

```
coherence = 'coherent' #'coherent'/'noncoherent'-only for FSK

modem_dict = {'psk': PSKModem,'qam':QAMModem,'pam':PAMModem,'fsk':FSKModem}
colors = plt.cm.jet(np.linspace(0,1,len(arrayOfM))) # colormap
fig, ax = plt.subplots(nrows=1,ncols = 1)

for i, M in enumerate(arrayOfM):
    #-----Initialization of various parameters----
    k=np.log2(M)
    EsN0dBs = 10*np.log10(k)+EbN0dBs # EsN0dB calculation
    SER_sim = np.zeros(len(EbN0dBs)) # simulated Symbol error rates
    inputSyms = np.random.randint(low=0, high = M, size=nSym)
    # uniform random symbols from 0 to M-1

    if mod_type.lower()=='fsk':
        modem=modem_dict[mod_type.lower()](M,coherence)#choose modem from dictionary
    else: #for all other modulations
        modem = modem_dict[mod_type.lower()](M)#choose modem from dictionary
    modulatedSyms = modem.modulate(inputSyms) #modulate

    for j,EsN0dB in enumerate(EsN0dBs):
        receivedSyms = awgn(modulatedSyms,EsN0dB) #add awgn noise

        if mod_type.lower()=='fsk': #demodulate (Refer Chapter 3)
            detectedSyms = modem.demodulate(receivedSyms,coherence)
        else: #demodulate (Refer Chapter 3)
            detectedSyms = modem.demodulate(receivedSyms)

        SER_sim[j] = np.sum(detectedSyms != inputSyms)/nSym

    SER_theory = ser_awgn(EbN0dBs,mod_type,M,coherence) #theory SER
    ax.semilogy(EbN0dBs,SER_sim,color =
     ↪  colors[i],marker='o',linestyle='',label='Sim '+str(M)+'-'+mod_type.upper())
    ax.semilogy(EbN0dBs,SER_theory,color = colors[i],linestyle='-',label='Theory
     ↪  '+str(M)+'-'+mod_type.upper())

ax.set_xlabel('Eb/N0(dB)');ax.set_ylabel('SER ($P_s$)')
ax.set_title('Probability of Symbol Error for M-'+str(mod_type)+' over AWGN')
ax.legend();fig.show()
```

4.3 Висновки до розділу 4

Використання ООП дозволяє створювати чітку структуру коду, розділяючи його на логічні компоненти (об'єкти), що спрощує розуміння та роботу з програмним кодом. ООП дозволяє легко розширювати функціональність телекомунікаційних модемів шляхом створення нових класів або успадкування вже існуючих. Застосування ООП полегшує підтримку програмного коду, оскільки зміни в одній частині системи можуть бути внесені без впливу на решту коду. ООП дозволяє захищати дані від несанкціонованого доступу та створювати високорівневі абстракції, що полегшує розробку та роботу з програмним кодом.

Використання ООП дозволяє прискорити процес розробки телекомунікаційних модемів, зменшуючи час на написання та тестування коду. Отже, використання об'єктно-орієнтованого програмування для розробки телекомунікаційних модемів може значно полегшити розробку, підтримку та розширення функціональності таких систем.

5 ОХОРОНА ПРАЦІ

Статтею 13 Закону України «Про охорону праці» передбачені обов'язки роботодавця щодо забезпечення працівників комфортними та безпечними умовами праці, а також права працівників на такі умови. Цей закон встановлює основні положення, які стосуються реалізації конституційного права працівників на охорону їхнього життя та здоров'я у процесі праці, на належні, безпечні та здорові умови праці. Крім того, він регулює відносини між роботодавцем і працівником з питань безпеки, гігієни праці та виробничого середовища, залучаючи відповідні органи державної влади, і встановлює єдиний порядок організації охорони праці в Україні.

Під час конфігурування бази даних IP розділених телекомунікаційних мережна працівника могли мати вплив такі небезпечні та шкідливі виробничі фактори:

1. Фізичні:

- підвищена запиленість та загазованість повітря робочої зони;
- підвищений рівень шуму на робочому місці;
- підвищена чи понижена вологість повітря;
- підвищений рівень електромагнітного випромінювання;
- підвищена чи понижена іонізація повітря;
- недостатня освітленість робочої зони;
- підвищена яскравість світла; понижена контрастність;
- пряма і відбита блисткість.

2. Психофізіологічні: статичне перевантаження та розумове перевантаження.

Відповідно до наведених факторів здійснюємо розробку заходів щодо безпечного виконання поставленого завдання.

5.1 Технічні рішення щодо безпечного виконання роботи

Вимоги щодо організації та обладнання робочих місць: площа, відведена на одне робоче місце має становити не менше 6 кв. м., а об'єм – не менше 20 куб. м. Конструкція робочого місця повинна забезпечувати підтримання оптимальної робочої пози (тобто такої, яка дозволяє працівникові виконувати роботу з мінімальним напруженням тіла, і яка дозволяє уникнути перевтоми в ході і після закінчення робочого процесу). Раціональна робоча поза має важливе значення для збереження здоров'я працівника, оскільки тривале перебування його в незручній і напруженій позі може призвести до таких захворювань, як сколіоз (викривлення хребта), варикозне розширення вен, плоскостопість тощо. Установлено, що робота в зігнутому положенні збільшує затрати енергії на 20%, а при значному нахиленні — на 45% порівняно з прямим положенням корпуса [14].

За потреби особливої концентрації уваги під час виконання робіт суміжні робочі місця операторів необхідно відділяти одне від одного перегородками висотою 1,5 - 2 м.

Робочі місця слід розташовувати відносно джерела природного світла (вікон) таким чином, щоб світло падало збоку, переважно зліва. Також робоче місце має відповідати сучасним вимогам ергономіки:

- стіл повинен мати висоту поверхні 680 - 800 мм., ширину 600 - 1400 мм. і глибину 800 - 1000 мм. (такі параметри забезпечують можливість виконання операцій в зоні досяжності працівника);

- робочий стілець робочий стілець має бути підйомно-поворотним, з можливістю регулювання висоти, бажано зі стаціонарними або змінними підлікотниками і напівм'якою нековзкою поверхнею сидіння, що легко чиститься і не електризується;

- екран комп'ютера має розташовуватися на оптимальній відстані від користувача, що становить 600 – 700 мм., але не менше за 600 мм. з урахуванням літерно-цифрових знаків і символів.

Приміщення, де здійснювалося конфігурування бази даних IP розділених телекомунікаційних мережза небезпекою ураження електричним струмом належить до приміщень без підвищеної небезпеки (сухе, мало запорошене, з

нормальною температурою повітря, ізольованими підлогами і малим числом заземлених приладів).

Персональні комп'ютери, периферійні пристрої, інше устаткування (апарати управління, контрольно-вимірювальні прилади, світильники), електропроводи та кабелі за виконанням і ступенем захисту мають апаратуру захисту від струму короткого замикання та інших аварійних режимів. Під час монтажу та експлуатації ліній електромережі необхідно повністю унеможливити виникнення електричного джерела загоряння внаслідок короткого замикання та перевантаження проводів, обмежувати застосування проводів з легкозаймистою ізоляцією і, за можливості, застосовувати негорючу ізоляцію.

Лінія електромережі для живлення персональних комп'ютерів і периферійних пристроїв виконується як окрема групова трипровідна мережа шляхом прокладання фазового, нульового робочого та нульового захисного провідників. Нульовий захисний провідник використовується для заземлення (занулення) електроприймачів.

Усі провідники відповідають номінальним параметрам мережі та навантаження, умовам навколишнього середовища, умовам розподілу провідників, температурному режиму та типам апаратури захисту.

Заземлені конструкції, що знаходяться в приміщеннях, де розміщені робочі місця операторів (батареї опалення, водопровідні труби, кабелі із заземленим відкритим екраном), надійно захищені діелектричними щитками або сітками з метою недопущення потрапляння людини під напругу.

Персональні комп'ютери і периферійні пристрої підключаються до електромережі тільки за допомогою справних штепсельних з'єднань і електророзеток заводського виготовлення. У штепсельних з'єднаннях та електророзетках, крім контактів фазового та нульового робочого провідників, мають бути спеціальні контакти для підключення нульового захисного провідника. Їхня конструкція має бути такою, щоб приєднання нульового захисного провідника відбувалося раніше, ніж приєднання фазового та нульового робочого провідників. Порядок роз'єднання при відключенні має бути зворотним.

5.2 Технічні рішення з гігієни праці та виробничої санітарії

5.2.1 Мікроклімат

Стан повітря робочої зони у виробничому приміщенні називають мікрокліматом або метеорологічними умовами. Мікроклімат або метеорологічні умови виробничих приміщенні, визначаються за такими параметрами:

температурою повітря у приміщенні, С;

відносною вологістю повітря, %;

рухливістю повітря, м/с;

тепловим випромінюванням, Вт/м3.

Всі ці параметри поодинці, а також у комплексі впливають на фізіологічну функцію організму його терморегуляцію і визначають самопочуття.

Конфігурування бази даних IP розділених телекомунікаційних мереж за енерговитратами відноситься до категорії I а (енерговитрати до 139Дж/с) [15]. Допустимі параметри мікроклімату для цієї категорії згідно ДСН 3.3.6.042-99 [16] наведені в табл. 5.1.

Таблиця 5.1 – Параметри мікроклімату

Період року	Допустимі		
	t,°C	W, %	V, м/с
Теплий	22-28	55	0,1-0,2
Холодний	21-25	75	0,1

Для створення і автоматичної підтримки в приміщенні незалежно від зовнішніх умов допустимих значень температури, вологості, чистоти і швидкості руху повітря обладнані системами опалення та кондиціонування повітря. Систематично проводиться вологе прибирання.

5.2.2 Склад повітря робочої зони

Оточуюче нас повітря (атмосфера) є найважливішим фактором забезпечення життя. В природних умовах повітря, як правило, не забруднене отруйними речовинами і життю людини не загрожує. Органи чутливості людини не дозволяють з достатньою точністю визначати якість повітря і запобігати загрозі отруєння.

В приміщенні, де здійснюється конфігурування бази даних IP розділених телекомунікаційних мереж, у повітрі можуть перевищувати ГДК такі речовини як вуглекислий газ, пил та озон. Джерелами цих речовин є офісна техніка. Пил потрапляє у приміщення ззовні через відкриті вікна та заноситься на одязі і взутті працівниками.

ГДК шкідливих речовин, які знаходяться в досліджуваному приміщені, наведені в таблиці 5.2.

Таблиця 5.2 – ГДК шкідливих речовин у повітрі

Назва речовини	ГДК, мг/м^3		Клас небезпечності
	Максимально разова	Середньо добова	
Вуглекислий газ	3	1	4
Пил нетоксичний	0,5	0,15	4
Озон	0,16	0,03	4

Рівні позитивних і негативних іонів у повітрі мають відповідати санітарно-гігієнічним нормам (табл. 5.3).

Таблиця 5.3 – Рівні іонізації повітря приміщень при роботі на ПК

Рівні	Кількість іонів в 1 см^3	
	n+	n-
Мінімально необхідні	400	600
Оптимальні	1500-3000	3000-5000

Максимально необхідні	50000	50000

Для підтримки допустимих значень мікроклімату та концентрації позитивних та негативних іонів необхідно передбачати установки або прилади зволоження або штучної іонізації, кондиціювання повітря.

5.2.3 Виробниче освітлення

Освітлення відіграє важливу роль у житті людини. Біля 90% інформації сприймається через зоровий канал, тому правильно виконане раціональне освітлення має важливе значення для виконання всіх видів робіт. Недостатня освітленість або її надмірна кількість знижують рівень збудженості центральної нервової системи і, природно, активність усіх життєвих процесів. Раціональне освітлення є важливим фактором загальної культури виробництва. Неможливо забезпечити чистоту та порядок у приміщені, в якому напівтемрява, світильники брудні або в занедбаному стані.

Приміщення, в яких встановлені персональні комп’ютери, повинні мати природне та штучне освітлення відповідно до ДБН В.2.5-28-2018 [17]. Норми освітленості при штучному освітленні та КПО (для ІІІ пояса світлового клімату) при природному та сумісному освітленні зазначені у таблиці 5.4:

Таблиця 5.4 - Норми освітленості в приміщенні

Характеристика зорової роботи	Найменший розмір об'єкта розрізнювання	Розряд зорової	Підрозряд зорової роботи	Контраст об'єкта розрізнення з	Характеристика фона	Освітленість, лк	КПО, $e_н$, %	
						Штучне освітлення	Природнє освітлення	Сумісне освітлення

						Комбіноване	Загальне	Верхнє або верхнє	Бокове	Верхнє або верхнє	Бокове
Дуже високої точності	Від 0,15 до 0,3	II	г	великий	світлий	1000	300	7	2,5	4,2	1,5

Штучне освітлення в досліджуваному приміщенні здійснюється системою загального рівномірного освітлення. У разі переважної роботи з документами, допускається застосування системи комбінованого освітлення (крім системи загального освітлення додатково встановлюються світильники місцевого освітлення).

Для забезпечення достатнього освітлення передбачені такі заходи:

1) Максимальне використання бічного природного освітлення.

2) Систематичне очищення скла від бруду – не рідше двох разів на рік.

3) Штучне освітлення в приміщенні забезпечується світильниками типу РСП08×250 (однолампові) з лампами ДРЛ-250.

5.2.4 Виробничий шум

Виробничий шум – це сукупність різних за гучністю і тоном звуків, які виникають у повітряному середовищі. В досліджуваному приміщенні наявний як постійний, так і непостійний шуми. Нормування непостійного шуму, а також орієнтовна оцінка загального рівня постійного шуму здійснюється скоректованим за частотою загальним рівнем звукового тиску – так званим рівнем звуку, який вимірюється в дБА за шкалою «А» шумоміра.

Непостійний шум характеризується еквівалентним рівнем звуку LA екв., що являє собою середньоквадратичний рівень звуку непостійного шуму, який має такий самий вплив на людину, як і постійний шум.

Для умов виконання роботи допустимі рівні звукового тиску не повинні перевищувати 50 дБА (табл. 5.5).

Таблиця 5.5 – Допустимі рівні звукового тиску і рівні звуку для постійного широкополосного шуму (згідно ДСН 3.3.6.037-99 [18])

Характер робіт	Допустимі рівні звукового тиску (дБ) в стандартизованих октавних смугах зі середньогеометричними частинами (Гц)									Допуст-мий рівень звуку, дБА
	32	63	125	250	500	1000	2000	4000	8000	
Виробничі приміщення	86	71	61	54	49	45	42	40	38	50

Рівень шуму в приміщенні не перевищує допустимих значень.

5.2.5 Виробничі випромінювання

Розрізняють природні та штучні джерела електромагнітних полів (ЕМП). У процесі еволюції біосфера постійно перебуває під впливом ЕМП природного походження (природний фон): електричне та магнітне поля Землі, космічні ЕМП, передусім ті, що генеруються Сонцем. У період науково-технічного прогресу людство створило і все ширше використовує штучні джерела ЕМП. У теперішній час ЕМП антропогенного походження значно перевищують природний фон і є тим несприятливим чинником, чий вплив на людину з року в рік зростає.

Значення напруженості електростатичного поля на робочих місцях (як у зоні екрана дисплея, так і на поверхнях обладнання, клавіатури, друкувального пристрою) мають не перевищувати гранично допустимих за ГОСТ 12.1.045-84 [19]. Значення напруженості електромагнітних полів на робочих місцях з ВДТ мають відповідати нормативним значенням (ДСанПіН 3.3.6-2002 [20], ГОСТ 12.1.045-84 [19]). Інтенсивність потоків інфрачервоного випромінювання має не перевищувати допустимих значень відповідно до ГОСТ 12.1.005-88 [21] (табл. 5.6).

Таблиця. 5.6 - Допустимі параметри електромагнітних неіонізуючих випромінювань і електростатистичного поля

Види поля	Допустима поверхнева щільність потоку енергії, Вт/кв.м
Електромагнітне поле оптичного діапазону в ультрафіолетовій частині спектру УФ-С (220 — 280 мм)	0,001
Електромагнітне поле оптичного діапазону в ультрафіолетовій частині спектру УФ-В (280 — 320 мм)	0,01
Електромагнітне поле оптичного діапазону в ультрафіолетовій частині спектру УФ-А (320 — 400 мм)	10,0
Електромагнітне поле оптичного діапазону в видимій частині спектру 400 — 760 мм	10,0
Електромагнітне поле оптичного діапазону в інфрачервоній частині спектру 0,76 — 10,0 мкм	35,0 — 70,0
Напруженість електричного поля відеодисплейного терміналу	20кВ/м

Для дотримання наведених нормативів слід використовувати офісну техніку з сертифікатом якості та дотримуватися встановлених режимів праці та відпочинку з ПК.

5.3 Пожежна безпека

Пожежна безпека – стан об’єкта, при якому з регламентованою ймовірністю відкидається можливість виникнення та розвиток пожежі, і впливу на людей її небезпечних факторів, а також забезпечується захист матеріальних цінностей.

З урахуванням ДСТУ Б В.1.1-36:2016 [22] приміщення, де здійснюється конфігурування бази даних IP розділених телекомунікаційних мереж відноситься до категорії вибухопожежонебезпеки В (тверді горючі та

важкогорючі речовини і матеріали, за умови, що приміщення, в яких вони знаходяться, не відносяться до категорій А, Б і питома пожежне навантаження для твердих і рідких легкозаймистих та горючих речовин на окремих ділянках площею не менше 10 м^2 кожна перевищує 180 МДж/м^2).

Клас приміщення і зон з вибухо- і пожежонебезпеки – П-IIа (приміщення, у якому знаходяться тверді горючі речовини та матеріали) [23].

5.3.1 Технічні рішення системи запобігання пожежі

Усі заходи системи запобігання пожежі в приміщенні можна поділити на організаційні, технічні, режимні та експлуатаційні.

Організаційні заходи пожежної безпеки передбачають: організацію пожежної охорони на об’єкті, проведення навчань з питань пожежної безпеки (включаючи інструктажі та пожежно-технічні мінімуми), застосування наочних засобів протипожежної пропаганди та агітації, проведення перевірок, оглядів стану пожежної безпеки приміщень, будівель, об’єкта в цілому та ін.

До технічних заходів належать: суворе дотримання правил і норм, визначених чинними нормативними документами при реконструкції приміщень, будівель та об’єктів, технічному переоснащенні виробництва, експлуатації чи можливому переобладнанні електромереж, опалення, вентиляції, освітлення і т. ін.

Заходи режимного характеру передбачають заборону куріння та застосування відкритого вогню в недозволених місцях, недопущення появи сторонніх осіб у вибухонебезпечних приміщеннях чи об'єктах, регламентацію пожежної безпеки при проведенні вогневих робіт тощо.

Експлуатаційні заходи охоплюють своєчасне проведення профілактичних оглядів, випробувань, ремонтів технологічного та допоміжного устаткування, а також інженерного господарства (електромереж, електроустановок, опалення, вентиляції)

5.3.2 Технічні рішення системи протипожежного захисту

Система протипожежного захисту – це сукупність організаційних заходів а також технічних засобів, спрямованих на запобігання впливу на людей небезпечних чинників пожежі та обмеження матеріальних збитків від неї.

До основних засобів пожежогасіння, що використовуються в приміщеннях такого класу відносять вогнегасники, розміщення яких в приміщеннях позначається встановленим поруч вказівним знаком

Згідно категорії пожежовибухонебезпеки будівлі, класу приміщення і за вибухо- і пожежонебезпекою П-IIа та Наказу Міністерства внутрішніх справ України «Про затвердження Правил експлуатації та типових норм належності вогнегасників», в приміщенні має бути встановлено 2 порошкових або вуглекислотних вогнегасники із зарядом речовини від 3 до 5 кг. [24]

В цілому приміщення по категорії вибухо- і пожежонебезпечності та ступеню вогнестійкості відповідає нормам, але особливу увагу потрібно звернути на утримання в справному стані засобів протипожежного захисту та своєчасне інформування пожежної охорони про несправність пожежної техніки, впровадження систем протипожежного захисту.

ВИСНОВКИ

Програмне моделювання роботи телекомунікаційних передавачів цифрових сигналів має значимість у практиці з ряду причин. Моделювання дозволяє досліджувати та вдосконалювати різноманітні аспекти передачі даних, такі як алгоритми модуляції, кодування, корекції помилок та управління каналом, що сприяє покращенню якості та ефективності передачі. Моделювання дозволяє визначити оптимальні параметри передавача для максимізації використання ресурсів мережі та забезпечення кращої пропускної здатності та якості обслуговування. Використання програмних моделей дозволяє тестувати та верифікувати нові технології та алгоритми перед їхнім впровадженням у реальних системах, що зменшує ризик непередбачених помилок та негативних наслідків. В порівнянні з фізичними експериментами або випробуваннями, програмне моделювання є економічнішим та швидшим способом вивчення та оптимізації роботи передавачів. Програмне моделювання дозволяє студентам та дослідникам отримати практичний досвід у роботі з цифровими телекомунікаційними системами, що сприяє підвищенню рівня їхньої кваліфікації та розвитку в цій області.

Отже, практична значимість програмного моделювання роботи телекомунікаційних передавачів цифрових сигналів полягає у забезпеченні покращення якості, ефективності та надійності телекомунікаційних систем, а також у зменшенні витрат та часу на їхнє розроблення та впровадження.

Розроблені комп'ютерні моделі дозволяють проводити аналіз та тестування телекомунікаційних систем без необхідності залучення реальних пристроїв. Це значно зменшує витрати часу та ресурсів, необхідних для тестування нових алгоритмів та технологій. Результати розробки моделей цифрових модуляторів дозволяють ефективно аналізувати алгоритми модуляції та їх вплив на якість передачі даних. Це допомагає вибирати оптимальні алгоритми та зменшує ризик непередбачуваних проблем при реальному впровадженні.

Результати моделювання дозволяють вдосконалити процеси передачі даних у мобільних мережах, що призводить до покращення ефективності та надійності

мобільних комунікацій. Використання об'єктно-орієнтованого програмування спрощує розробку та управління телекомунікаційними модемами. Чітка структура коду, легка розширюваність та підтримка полегшують розробку та забезпечують ефективне управління.

Особлива увага до питань охорони праці в контексті телекомунікаційних систем є важливою для забезпечення безпеки та здоров'я працівників у даній галузі. Отримані результати сприяють покращенню функціональності, ефективності та безпеки телекомунікаційних систем, що є важливим для розвитку цієї галузі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Frenzel, L.E. (2016). Principles of Electronic Communication Systems. New York: McGraw- Hill Education.

2. Quadrature Amplitude Modulation (QAM) (2023). What is it? Electrical4U. https://www.electrical4u.com/quadrature- amplitude- modulation- qam/ (accessed 14 February 2023).

3. Beard, C., Stallings, W., and Tahiliani, M.P. (2015). Wireless Communication Networks and Systems, Global Edition. Pearson.

4. Salvi, S. and Geetha, V. (2019). From light to Li-F i: research challenges in modulation, MIMO, deployment strategies and handover. International Conference on Data Science and Engineering (ICDSE), Patna, India (26–28 September 2019), pp. 107–119.

5. Васильківський, М. В. Програмні технології в інфокомунікаційних системах. Навчальний посібник для студентів спеціальності 172 «Телекомунікації та радіотехніка» : електронний навчальний посіб-ник комбінованого (локального та мережного) використання [Електронний ресурс] / Васильківський М. В., Бортник Г. Г., Кичак В. М. – Вінниця : ВНТУ, 2023. – 141 с.

6. Основи програмування (Python, Java) : лабораторний практикум / Смотр О., Придатко О., Малець І. – Львів : ЛДУ БЖД, 2019. – 134 с.

7. Програмування мовою Python / О.М. Васильєв. — Тернопіль: Навчальна книга – Богдан, 2019. — 504 с.; іл.

8. Васильківський, М., Коломієць, А., & Грабчак, Н. (2022). Дослідження функціональних параметрів інфокомунікаційних мереж 6G. Вісник Хмельницького національного університету, (6), 46–52. https://www.doi.org/10.31891/2307-5732-2022-315-6-46-52

9. Шарадкін Д.М., Субач І.Ю., Микитюк А.В. Інструментальні засоби Python для моделювання та си-стемного аналізу часових рядів при вирішенні задач кіберзахисту інформаційно-комунікаційних си-стем: навч. пос. / Шарадкін Д.М.,

Субач І.Ю., Микитюк А.В.; ІСЗЗІ КПІ ім. Ігоря Сікорського. – Київ : КПІ ім. Ігоря Сікорського, 2023.- 139 с.

10. Березовський В. Є. Чисельні методи з прикладами реалізації мовою Python / В. Є. Березовський, Л. Є.Ковальов, М. О.Медведєва : навчальний посібник. Умань : ВПЦ «Візаві», 2023. 88 с.

11. Васильківський, М., Прикмета, А., Олійник, А., & Нікітович, Д. (2023). Оптимізація інтелектуальних телекомунікаційних мереж. Вісник Хмельницького національного університету, Технічні науки. – 2023. – № 1. (317). – С. 33–41. doi: 10.31891/2307-5732-2023-317-1-33-41

12. М. Васильківський, О. Городецька, Б. Климчук, і В. Говорун, «Стратегії технологічного розвитку апаратного забезпечення інфокомунікаційних радіомереж», ІТКІ, вип. 56, вип. 1, с. 83–91, Бер 2023.

13. ДСТУ OHSAS 18002:2015. Системи управління гігієною та безпекою праці. Основні принципи виконання вимог OHSAS 18001:2007 (OHSAS 18002:2008, IDT). К. : ГП «УкрНИУЦ», 2016. 21 с.

14. НПАОП 0.00-7.15-18 Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями. URL: http://sop.zp.ua/norm_npaop _0 _ 00-7_ 15-18_01_ua.php.

15. Гігієнічна класифікація праці (за показниками шкідливості і небезпеки факторів виробничого середовища від 12.08.1986 № 4137-86. - [Електронний ресурс] - Режим доступу: http://zakon4.rada.gov.ua/laws/show/v4137400-86

16. ДСН 3.3.6.042-99 Санітарні норми мікроклімату виробничих приміщень. - [Електронний ресурс] - Режим доступу: http://mozdocs.kiev.ua/view.php?id=1972

17. ДБН В.2.5-28-20018 Природне і штучне освітлення - [Електронний ресурс] - Режим доступу: http://document.ua/prirodne-i-shtuchne-osvitlennja-nor8425.html

18. ДСН 3.3.6.037-99 Санітарні норми виробничого шуму, ультразвуку та інфразвуку. - [Електронний ресурс] - Режим доступу: http://document.ua/sanitarni-normi-virobnichogo-shumu-ultrazvuku-ta-infrazvuku-nor4878.html

19. ДСНіП3.3.6.096-2002. Державні санітарні норми і правила при роботі з джерелами електромагнітних полів. URL: http://zakon2.rada.gov.ua/laws/show/ z0203-03.

20. ДСанПіН 3.3.6-2002 Державні санітарні норми і правила при роботі з джерелами електромагнітних полів - [Електронний ресурс] - Режим доступу: http://zakon.nau.ua/doc/?code=z0203-03

21. НПАОП 0.00-7.11-12. Загальні вимоги стосовно забезпечення роботодавцями охорони праці працівників. URL: http://zakon.rada.gov.ua/laws/show/ z0226-12.

22. ДСТУ Б В.1.1-36:2016 Визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпек. URL: https://dbn.co.ua/load/normativy/dstu/dstu_b_v_1_1_36/5-1-0-1759.

23. ДБН В.1.1-7:2016 Пожежна безпека об`єктів будівництва. Загальні вимоги. URL: http://www.poliplast.ua/doc/dbn _v.1.1-7-2002.pdf.

24. Наказ Міністерства внутрішніх справ України «Про затвердження Правил експлуатації та типових норм належності вогнегасників». URL: https://zakon.rada.gov.ua/laws/show/z0225-18#Text.

ДОДАТКИ

Додаток А

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

ПРОГРАМНА МОДЕЛЬ ЦИФРОВОГО ПЕРЕДАВАЧА ПРОГРАМНО-КЕРОВАНОЇ ТЕЛЕКОМУНІКАЦІЙНОЇ СИСТЕМИ

назва бакалаврської дипломної роботи

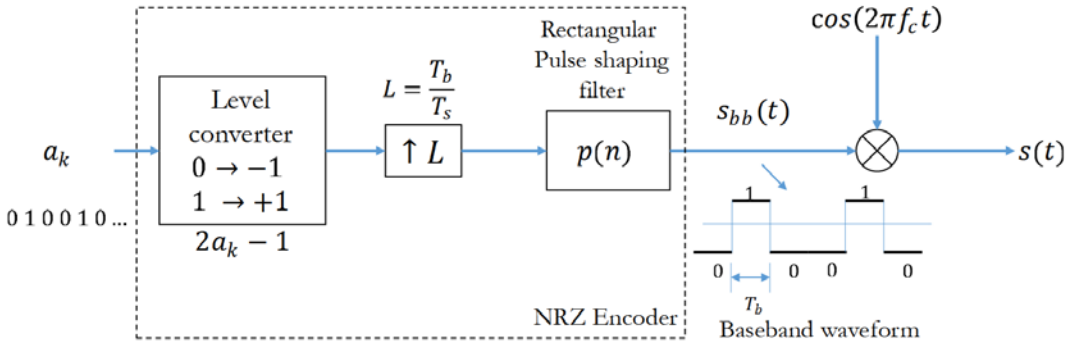

Рисунок 1 – Модель передавача BPSK

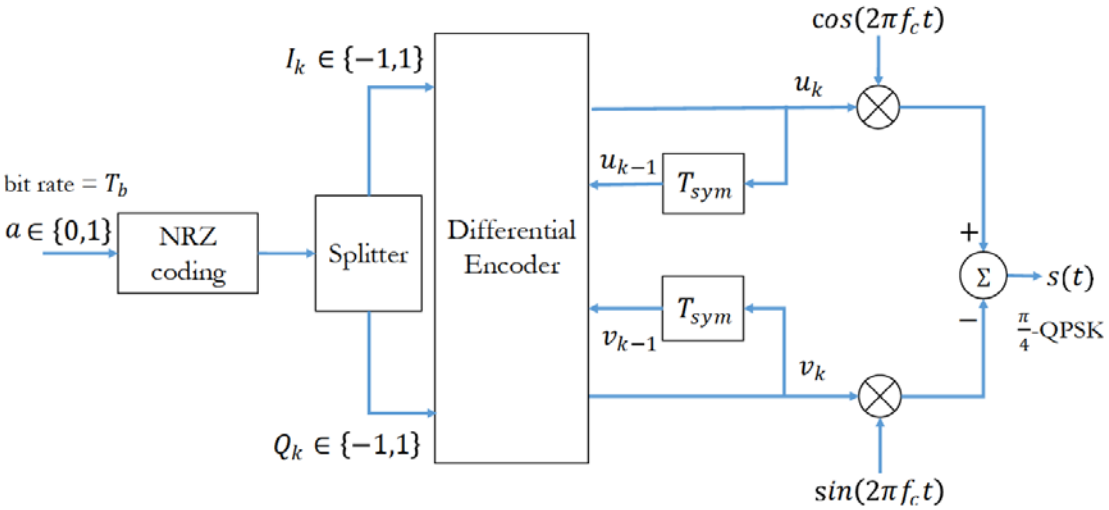

Рисунок 2 – Структурна схема π/4-DQPSK модулятора сигналів

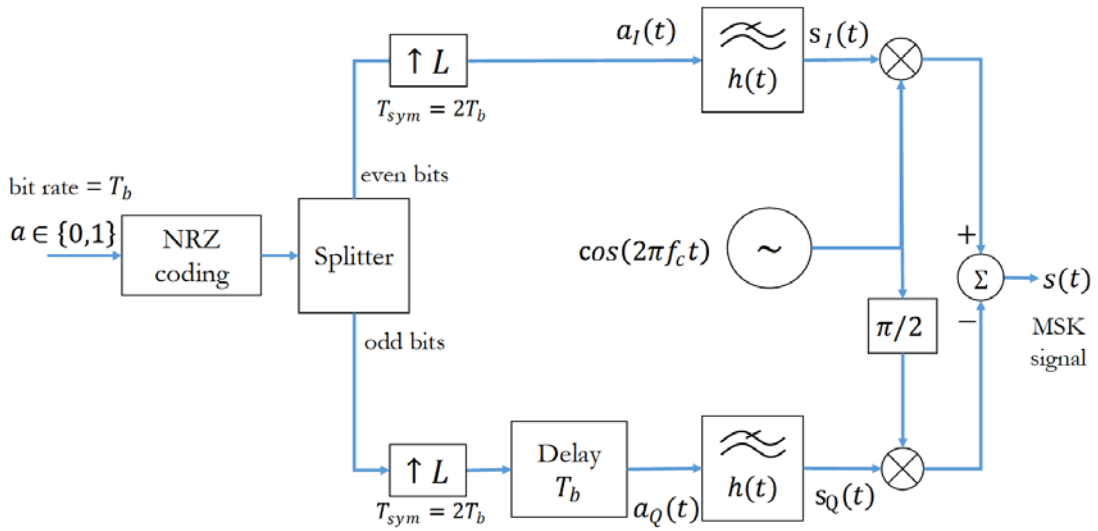

Рисунок 3 – Структурна модель MSK модулятора еквівалентного з OQPSK модуляцією

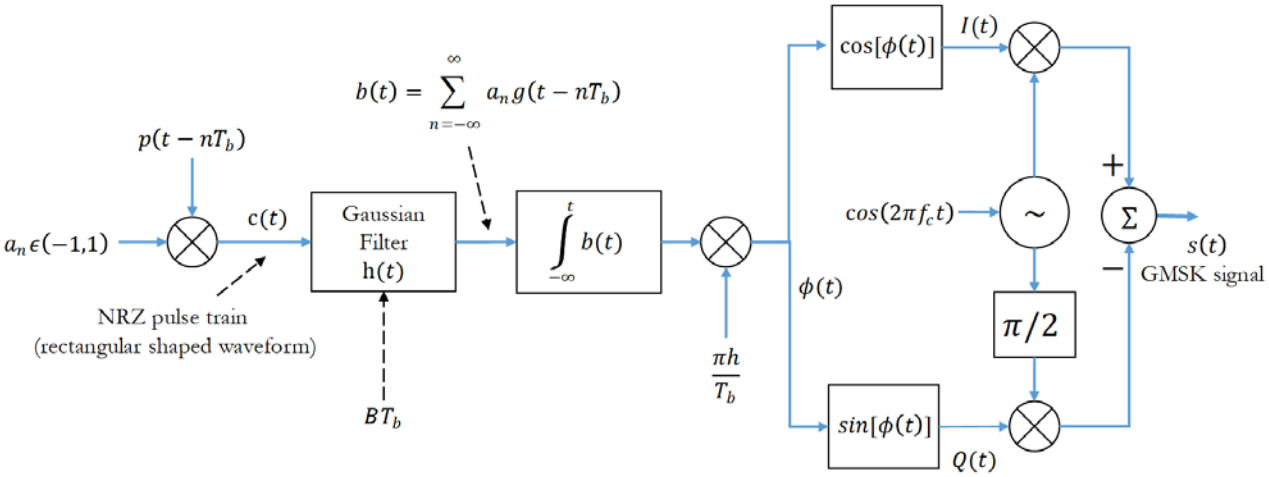

Рисунок 4 - Квадратурна реалізація GMSK модулятора

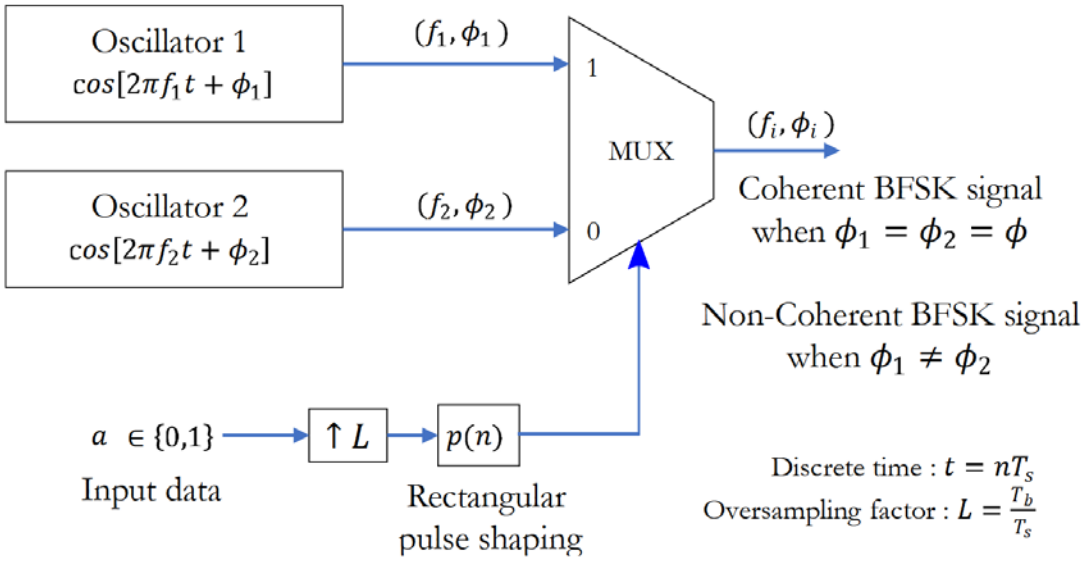

Рисунок 5 - Дискретна еквівалентна модель для BFSK модуляції

```
import numpy as np # for numerical computing
import matplotlib.pyplot as plt # for plotting functions
from matplotlib import cm # colormap for color palette
from scipy.special import erfc
from DigiCommPy.modem import PSKModem,QAMModem,PAMModem,FSKModem
from DigiCommPy.channels import awgn
from DigiCommPy.errorRates import ser_awgn

#---------Input Fields-----------------------
nSym = 10**6 # Number of symbols to transmit
EbN0dBs = np.arange(start=-4,stop = 12, step = 2) # Eb/N0 range in dB for simulation
mod_type = 'FSK' # Set 'PSK' or 'QAM' or 'PAM' or 'FSK'
arrayOfM = [2,4,8,16,32] # array of M values to simulate
#arrayOfM=[4,16,64,256] # uncomment this line if MOD_TYPE='QAM'
```

```
coherence = 'coherent' #'coherent'/'noncoherent'-only for FSK

modem_dict = {'psk': PSKModem,'qam':QAMModem,'pam':PAMModem,'fsk':FSKModem}
colors = plt.cm.jet(np.linspace(0,1,len(arrayOfM))) # colormap
fig, ax = plt.subplots(nrows=1,ncols = 1)

for i, M in enumerate(arrayOfM):
    #-----Initialization of various parameters----
    k=np.log2(M)
    EsN0dBs = 10*np.log10(k)+EbN0dBs # EsN0dB calculation
    SER_sim = np.zeros(len(EbN0dBs)) # simulated Symbol error rates
    inputSyms = np.random.randint(low=0, high = M, size=nSym)
    # uniform random symbols from 0 to M-1

    if mod_type.lower()=='fsk':
        modem=modem_dict[mod_type.lower()](M,coherence)#choose modem from dictionary
    else: #for all other modulations
        modem = modem_dict[mod_type.lower()](M)#choose modem from dictionary
    modulatedSyms = modem.modulate(inputSyms) #modulate

    for j,EsN0dB in enumerate(EsN0dBs):
        receivedSyms = awgn(modulatedSyms,EsN0dB) #add awgn noise

        if mod_type.lower()=='fsk': #demodulate (Refer Chapter 3)
            detectedSyms = modem.demodulate(receivedSyms,coherence)
        else: #demodulate (Refer Chapter 3)
            detectedSyms = modem.demodulate(receivedSyms)

        SER_sim[j] = np.sum(detectedSyms != inputSyms)/nSym

    SER_theory = ser_awgn(EbN0dBs,mod_type,M,coherence) #theory SER
    ax.semilogy(EbN0dBs,SER_sim,color =
     ↪  colors[i],marker='o',linestyle='',label='Sim '+str(M)+'-'+mod_type.upper())
    ax.semilogy(EbN0dBs,SER_theory,color = colors[i],linestyle='-',label='Theory
     ↪  '+str(M)+'-'+mod_type.upper())

ax.set_xlabel('Eb/N0(dB)');ax.set_ylabel('SER ($P_s$)')
ax.set_title('Probability of Symbol Error for M-'+str(mod_type)+' over AWGN')
ax.legend();fig.show()
```

Рисунок 6 – Лістинг програми оцінювання ефективності цифрових телекомунікаційних передавачів

Додаток Б

(обов’язковий)

Протокол перевірки кваліфікаційної роботи на наявність текстових запозичень

ПРОТОКОЛ

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Програмна модель цифрового передавача програмно-керованої телекомунікаційної системи

Тип роботи: Бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра інфокомунікаційних систем і технологій, факультет інформаційних електронних систем
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 91,58 % Схожість 8,42 %

Аналіз звіту подібності (відмітити потрібне):

■ 1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

☐ 2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

☐ 3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа відповідальна за перевірку _____ Васильківський М.В.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Кісіль А.Ю.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Бортник С.Г.
(підпис) (прізвище, ініціали)