Міністерство освіти і науки України
Національний університет «Запорізька політехніка»
Кафедра радіотехніки та телекомунікацій

МЕТОДИЧНІ ВКАЗІВКИ

до курсового проєкту з дисципліни
«Широкосмугові технології телекомунікацій»
для студентів спеціальності
172 «Електронні комунікації та радіотехніка»
всіх форм навчання

Частина ІІ

2024

Методичні вказівки до курсового проєкту з дисципліни «Широкосмугові технології телекомунікацій» для студентів спеціальності 172 «Телекомунікації та радіотехніка» всіх форм навчання. Частина ІІ / Укл. В.С. Кабак, Г.В. Мороз, Г.М. Сидоренко. – Запоріжжя: НУ «Запорізька політехніка», 2024. – 72 с.

Укладачі: В.С.Кабак, доцент, к.т.н.,
Г.В. Мороз, ст. викладач,
Г.М. Сидоренко, зав. лаб.

Рецензент: С.В. Морщавка, доцент, к.т.н.

Відповідальний.
за випуск: Г.М. Сидоренко, зав. лаб.

Затверджено:
на засіданні кафедри
радіотехніки та телекомунікацій
Протокол № 5 від 05.04.2024 р.

Рекомендовано до видання НМК
факультету інформаційної безпеки та
електронних комунікацій
Протокол № 6 від 24.04.2024 р.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОНАКИ

БПН	–	без повернення до нуля
ЕОМ	–	електронна обчислювальна машина
ПЗ	–	пояснювальна записка
ПВП	–	псевдовипадкова послідовність
ТЗ	–	технічне завдання
BER	–	(Bit Error Rate) коефіцієнт бітової помилки
BPSK	–	(Binary Phase Shift Keying) двійкова фазова маніпуляція
CDMA	–	(Code Division Multiple Access) множинний доступ з кодовим розділенням каналів
DSSS	–	(Direct Sequence Spread Spectrum) широкосмугова модуляція з прямим розширенням спектра
DS-CDMA	–	системи зв’язку з використанням методу доступу CDMA з розширенням спектра прямою послідовністю
FHSS	–	(Frequency Hopping Spread Spectrum) розширення спектра з швидкою псевдовипадковою перебудовою частоти
GMSK	–	(Gaussian Minimum Shift Keying) гаусівська маніпуляція з мінімальним зсувом
PSK	–	(Phase Shift Keying) фазова маніпуляція
M-QAM	–	(M-level Quadrature Amplitude Modulation) М-рівнева квадратурна амплітудна модуляція
NRZ	–	(Non Return to Zero) без повернення до нуля
OFDM	–	(Orthogonal Frequency Division Multiplexing) мультиплексування ортогонально розділених несучих
PSK	–	(Phase Shift Keying) фазова маніпуляція
QAM	–	(Quadrature Amplitude Modulation) квадратурна амплітудна модуляція
QPSK	–	(Quadrature Phase Shift Keying) квадратурна фазова маніпуляція

1 ЗАСТОСУВАННЯ ПСЕВДОВИПАДКОВИХ ПОСЛІДОВНОСТЕЙ У КАНАЛІ ЗВЯЗКУ З ВИКОРИСТАННЯМ DS-CDMA

У синхронних DS-CDMA системах кожен з користувачів використовує свій власний адресний код для розширення спектру інформаційного сигналу.

У кожному з терміналів користувачів інформаційні дані спочатку підлягають одному з видів цифрової модуляції (наприклад, BPSK або QPSK). Потім модульовані біти підлягають процедурі розширення спектра за допомогою адресного коду або псевдовипадкової послідовності (ПВП), зокрема, М-послідовності, послідовності Голда або інших послідовностей з визначеними псевдовипадковими властивостями.

Сигнал з розширеним спектром від усіх користувачів одночасно передається до базової станції. Базова станція відокремлює інформаційний сигнал від кожного користувача шляхом кореляційної оцінки прийнятого сигналу з адресною кодовою послідовністю, що призначається кожному користувачеві.

В моделі каналу зв’язку як радіочастотну модуляцію пропонується використовувати метод QPSK.

Власне сама модель подається у вигляді програмного комплексу, який складається з головної програми і ряду підпрограм, які забезпечують функціонування головної програми. За основу програмного комплексу покладені алгоритми і програми, які наведені в [1].

Набір підпрограм для реалізації модуляції QPSK з визначенням їх функціонального призначення детально розглянуто у [2].

Базові визначення, принцип роботи і основні характеристики систем з розширенням спектра розглянуті в [3]

У першому розділі розглядається необхідний набір підпрограм для генерації ПВП і оцінки їх кореляційних властивостей

1.1 Програмна реалізація псевдовипадкових послідовностей

1.1.1 Генерація М-послідовностей

Як було визначено у [2], М-послідовності генеруються окремим лінійним регістром зсуву. Для утворення послідовності максимальної довжини ($N=2^n-1$) необхідне застосування n-каскадного регістру зсуву з лінійним зворотним зв'язком. Відповідно степінь генеруючого полінома рівняється n.

В моделі каналу зв'язку генерація М-послідовностей реалізується у підпрограмі **msec.m**. Параметрами підпрограми виступають кількість розрядів регістру зсуву **stg**, місце підключення зворотного зв'язку (виходи відповідних регістрів) **taps,** початковий вміст розрядів регістра **inidata**, кількість можливих послідовностей **n.** Згенерована підпрограмою М-послідовність зберігається у змінній **mout.**

Роботу підпрограми продемонструємо на такому прикладі. Припустимо, що необхідно сформувати М-послідовність на підставі генеруючого поліному $h(x)=x^3+x+1$. Виходячи з виду поліному зворотний зв'язок вводиться між виходом регістра зсуву і виходом першого розряду. Тобто, схема регістру зсуву набуває вигляду рис.1.1.

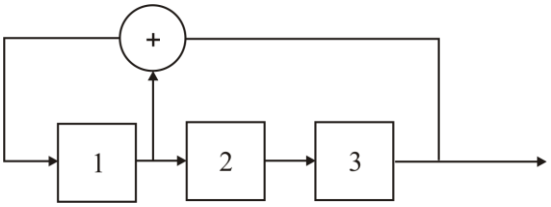

Рисунок 1.1 – Трьохрозрядний регістр зсуву зі зворотним зв'язком

Припустимо , що початковому стану регістра зсуву відповідають одиниці в усіх розрядах. Тоді в процесі тактування регістру і організацією зсуву на один розряд отримаємо наступні комбінації:

111 011 101 010 001 100 110 111.

Очевидно, що восьма комбінація, повторює першу комбінацію, тобто, довжина послідовності або її період рівняється 7.

Виходом схеми, який формує М-послідовність, є вихід останнього розряду. Відповідно для даного варіанту М-послідовність має такий вид:**1110100.**

Згідно з умовою балансу для ПВП кількість одиниць і нулів не повинна відрізнятися більше ніж на одиницю. Тобто, умова балансу для цієї послідовності виконується.

Друга умова випадковості послідовності є умова циклічності, де під циклом розуміють неперервну послідовність однакових двійкових символів. Бажано, щоб у кожному фрагменті послідовності приблизно половину від кількості циклів складали цифри обох типів довжиною 1, приблизно чверть – довжиною 2, приблизно одну восьму – довжиною 3 і так далі.

Для наведеної послідовності кількість циклів одиниць і нулів рівняється 2, половина з яких як для нулів і одиниць має довжину 1 і по одному циклу одиниць довжиною 3 і нулів довжиною 2. Очевидно, що через коротку дожину послідовності подальшу перевірку циклічності провести неможливо, оскільки отримати чверть ii одну восьму для такої кількості просто неможливо.

Щоб більш наочно продемонструвати циклічність ПВП, яка реалізується за таким алгоритмом збільшимо кількість регістрів до 4 (рис. 1.2). Для початкового стану регістра також приймемо одиниці в усіх розрядах.

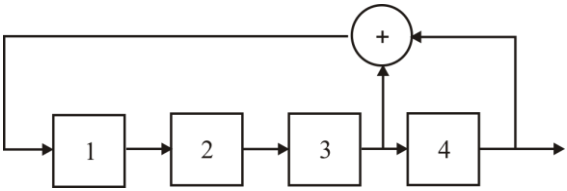

Рисунок 1.2 – Формування М-послідовності за допомогою 4- розрядного регістру зсуву

Відповідно отримаємо наступні стани регістру зсуву:
1111 0111 0011 0001 1000 0100 0010 1001
1100 0110 1011 0101 1010 1101 1110 1111

Очевидно, що у даному випадку довжина послідовності $L=2^4-1=15$, а сама М-послідовність має такий вид:

1 1 1 1 0 0 0 1 0 0 1 1 0 1 0,

з якого видно, що вона містить 8 одиниць і 7 нулів.

Для такої послідовності маємо чотири циклів нулів і одиниць. З чотирьох циклів нулів (000, 00, 0, 0)отримали половину циклів довжиною 1, чверть циклів довжиною 2. Подібна ж картина спостерігається і для циклів одиниць. Тому, можна відзначити, що послідовності, які генеруються за таким алгоритмом відносяться до послідовностей максимальної довжини і задовольняють умовам збалансованості і циклічності.

Саме такий алгоритм роботи реалізовано у підпрограмі mseq.m. Для генерації М-послідовності, для розглянутого у першому прикладі трьохрозрядного регістру зсуву (рис.1.1), необхідно у головній програмі задати команду:

m1=mseq(3, (1,3), (1, 1, 1));,

де задано кількість розрядів stg=3, місце підключення зворотного зв'язку taps=[1,3] , початковий стан регістра inidata=[1,1,1].

Як результат виконання програми mseq.m генерується М-послідовність у вигляді вектора [1,1,1,0,1,0,0]. На рис. 1.3 зображено епюру сформованої послідовності у вигляді цифрових відліків.

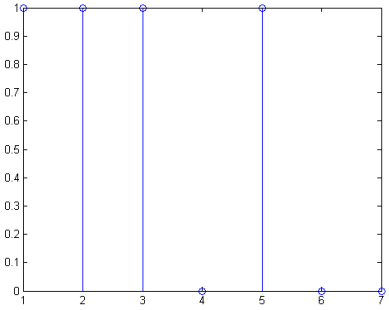

Рисунок 1.3 – Сформована М-послідовність

У підпрограмі mseq.m передбачено використання четвертого аргументу, який визначає кількість виходів схеми. Якщо кількість виходів N задається у підпрограмі, то можна отримати N М-

послідовностей, які зсунуті між собою на один символ. Наприклад, інші М-послідовності може бути згенеровано за допомогою команди:

m2=mseq(3, (1,3), (1, 1, 1), 3).

По закінченні виконання програми отримаємо як результат три М-послідовності зсунуті на один чіп:

ans =
1,1,1,0,0,1,0
0,1,1,1,0,0,1
1,0,1,1,1,0,0.

Для реалізації зсуву у підпрограмі mseq.m у свою чергу використовується підпрограма shift.m, яка задає зсув заданої кількості бітів для вектору або матриці, що задається користувачем.

1.1.2 Генерація послідовностей Голда

Схема генерації послідовностей Голда з використанням трьохкаскадного регістру зсуву зображена на рис.1.4.

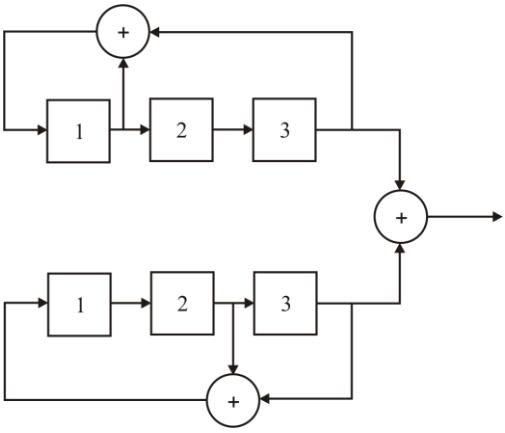

Рисунок 1.4 – Апаратна реалізація послідовності Голда

Як було визначено у [3], для генерації послідовності Голда необхідно сформувати дві М-послідовності з наступним їх додаванням

за модулем 2. Відповідно апаратна частина повинна містити два лінійних регістри зсуву зі зворотним зв'язком.

Кількість послідовностей Голда, що генеруються за такою схемою рівняється 2^n-1 [3]. Вони отримуються шляхом зміни початкового значення регістру і додаванням двох М-послідовностей при використанні n-каскадного регістру зсуву.

Для послідовності Голда, що генерується за схемою рис.1.4 з використанням кращої пари М-послідовностей, значення вектора крос-кореляціної функції визначаються трьома значеннями , а саме: [-1, -t(n,), t(n)-2],де t(n) визначається за (2.7) [3].

Алгоритм формування послідовності Голда продемонструємо на прикладі схеми з рис.1.4.

Як перший регістр зсуву будемо використовувати результати прикладу з параграфу 1.1.1. Тобто на виході першого регістра зсуву отримаємо послідовність [1,1,1,0,1,0,0].

Для другого регістру зсуву змінимо місце підключення зворотного зв'язку (між другим і третім розрядом). Початкові значення регістрів приймемо також рівними 1.

У такому разі стани розрядів при циклічному зсуві для такого варіанту регістра зсуву матимуть вигляд:

111 011 001 100 010 101 110 111 …, .

Відповідно сама М-послідовність на виході другого регістру зсуву приймає такий вид: 1 1 1 0 0 1 0 (рис.1.5).

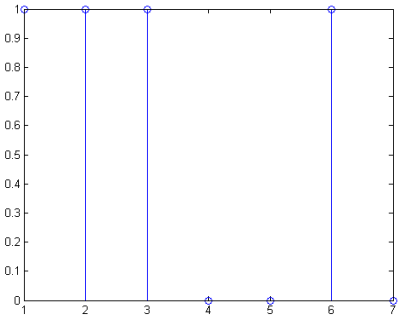

Рисунок 1.5 – Сформована М-послідовність на виході другого регістру зсуву

Результат додавання цих послідовностей за модулем 2 утворює Голд-послідовність:

$$\begin{array}{r} 1\ 1\ 1\ 0\ 1\ 0\ 0 \\ \oplus \\ 1\ 1\ 1\ 0\ 0\ 1\ 0 \\ \hline 0\ 0\ 0\ 0\ 1\ 1\ 0 \end{array} \quad ..$$

Генерація послідовності Голда за наведеним алгоритмом реалізована у підпрограмі goldseq.m. Аргументами функції виступають дві М-послідовності, що обираються як краща пара. Наприклад, для генерації трьохкаскадної Голд послідовності для розглянутого прикладу необхідно виконати такі команди:

m1=mseq(3, (1,3), (1, 1, 1));

m2=mseq(3, (2,3), (1, 1, 1));

g1=goldseq (m1, m2) ;.

В результаті обчислень отримуємо трьохкаскадну послідовність Голда у вигляді вектора [0, 0, 0, 0, 1, 1, 0] з довжиною 7.

Відповідна епюра сформованої послідовності Голда зображена на рис.1.6.

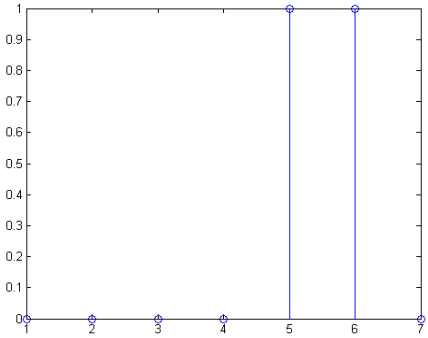

Рисунок 1.6 – Послідовність Голда, що реалізована за схемою рис.1.4

Змінюючи початковий вміст регістрів можна отримати різні послідовності Голда, наприклад:

m3=mseq(3, (1,3), (1, 0, 0));

m4=mseq(3, (2,3), (1, 0, 1));

g2=goldseq (m3, m4) ;.

Відповідно на рис.1.7 і 1.8 зображено М-послідовності для нових початкових значень регістрів, а на рис.1.8 –послідовність Голда як результат додавання М-послідовностей за модулем 2.

У підпрограмі goldseq.m також передбачено використання третього аргументу, який визначає кількість виходів схеми. Якщо задається кількість виходів N, то можна отримати N Голд-послідовностей, які зсунуті між собою на один символ.

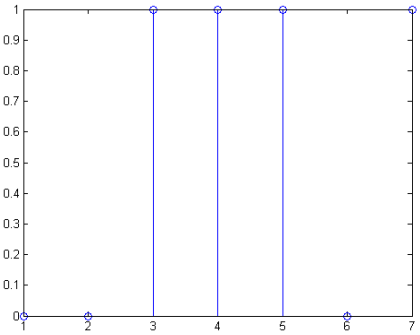

Рисунок 1.7 – Сформована М-послідовність на виході першого регістру

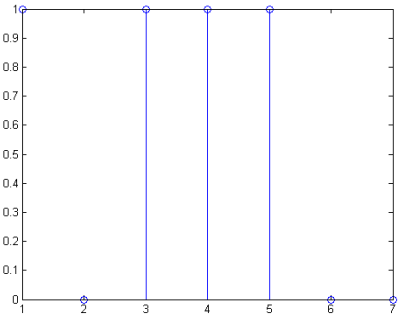

Рисунок 1.8 – Сформована М-послідовність на виході другого регістру

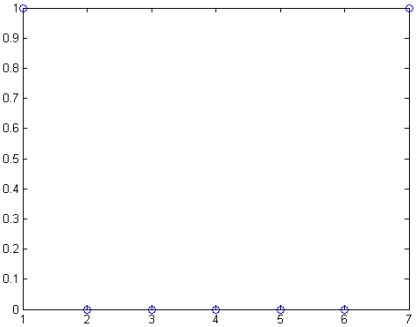

Рисунок 1.9 – Утворена послідовність Голда

Наприклад, інші Голд -послідовності можуть бути згенеровані за допомогою команди:

g2=goldseq(m1, m2, 3);.

По закінченні виконання програми отримаємо як результат три Голд-послідовності, що зсунуті на один символ:

ans=

0, 0 ,0, 0, 1, 1, 0

1, 0, 0, 1, 1, 0, 1

0, 1, 0, 1, 0, 0 ,0.

1.1.3 Програмна оцінка кореляційних властивостей ПВП

Для роботи корелятора у приймальній частині каналу зв’язку з використанням методу DSSS необхідно реалізувати оцінку автокореляційної функції і взаємної кореляційної функції між різними ПВП.

Алгоритм визначення авто – і кроскореляційної функцій ПВП реалізується достатньо просто. Як було показано у [3] кореляційні властивості послідовностей можна визначити шляхом обчислення кількості збігів і розбіжностей під час дискретних зсувів, що кратні тривалості символу. Тобто відбувається посимвольне (побітове) перемноження для зсунутих послідовностей, тільки для автокореляційної функції проводиться зсув однієї й тієї ж

послідовності, а для взаємної кореляційної функції реалізується зсув між різними послідовностями.

Процес розширення спектра для методу DS-CDMA у передавальному тракті передбачає операцію перемноження чіпів ПВП з символами, що отримані в результаті QPSK модуляції і які подаються у форматі NRZ. Оскільки, кодові послідовності, що згенеровані у підпрограмах msec,m, gold.sec.m складаються тільки з елементів, що належать ансамблю {0,1}, то ці ПВП також перетворюються на двохполярну послідовність, тобто формується кодова послідовність, елементи якої обираються з ансамблю {-1,1}.

Тоді алгоритм обчислення кореляційної функції може бути реалізовано простим посимвольним перемноженням зсунутих послідовностей з наступним додаванням отриманих результатів, тобто сума автоматично буде визначати різницю A-D збігів і розбіжностей під час посимвольного порівняння.

Відповідно оцінка автокореляційної функції за таким алгоритмом реалізована у підпрограмі autocorr.m. Аргументами цієї функції виступають ім'я послідовності і кількість періодів коду для яких необхідно обчислити автокореляційну функцію. Наприклад, для обчислення авто кореляційної функції кодової послідовності X(t)=[1, 1, 1, -1, -1, 1, -1] необхідно виконати таку процедуру:

```
X=[ 1, 1, 1, -1, -1, 1, -1];
```

R_{xx}=autocorr(X);

Так, для отриманої у параграфі 3.1.1 трьохкаскадної М-послідовності автокореляційна функція може бути обчислена за допомогою таких операторів:

```
m1=m1*2-1;
autocorr(m1);
```

В результаті обчислень отримуємо вектор значень автокореляційної функції [7,-1,-1,-1.,-1,-1,-1] (рис.1.10), що повністю відповідає виразу (4.6) [3].

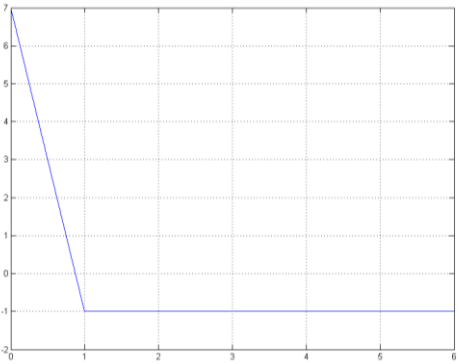

Рисунок 1.10 – Автокореляційна функція М-послідовності

Взаємна кореляційна функція розраховується за таким же алгоритмом. У моделі каналу функція взаємної кореляції або крос-кореляційна функція обчислюється за допомогою підпрограми crosscorr.m. Аргументами цієї підпрограми виступають імена послідовностей і кількість періодів коду для яких необхідно обчислити кореляційну функцію. Наприклад, для обчислення крос - кореляційної функції для кодових послідовностей X(t)=[1, 1, 1, -1, -1, 1, -1] і Y(t)=[1, -1, 1, -1, 1, -1, 1] необхідно виконати наступну процедуру:

```
X(t)=[ 1, 1, 1, -1, -1, 1, -1];
Y(t)=[ 1, -1, 1, -1, 1, -1, 1];
Rxy=crosscorr(X,Y);
```

У параграфі 1.1.1 було отримано дві М-послідовності m1 і m2 (рис.1.3 і рис.1.5). Тоді взаємна кореляційна функція між отриманими М-послідовностями m1 m2(1,:) може бути обчислена за допомогою операторів:

```
m1=m1*2-1;
m2=m2*2-1;
crosscorr(m1, m2(1, :));.
```

Результат розрахунків надає вектор значень крос-кореляційної функції [3, -1, 3, -1, -1, -5, 3] (рис.1.11).

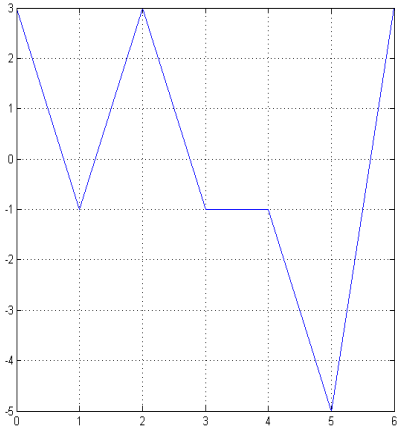

Рисунок 1.11 – Взаємна кореляційна функція згенерованих М-послідовностей

Необхідно відзначити, що значення вектору формуються з трьох цифр [-1, -t(n), t(n)-2] де t(n)=5, як це виходить за (2.7) [3]. Відповідно m1, m2(1, :) мають характеристики кращої пари.

Можна відзначити, що кількість М-послідовностей, що задовольняють цій вимозі, дуже мала. У таблиці 1.1 наводиться інформація про включення зворотного зв'язку для багатокакадної М-послідовності і визначена кількість найкращих пар.

Оцінка кореляційних властивостей Голд-послідовностей проводиться повністю тотожно з використанням підпрограм autocorr.m і crosscorr.m.

Так для розглянутої у параграфі 1.1.2 послідовності Голда для обчислення автокореляційної функції необхідно реалізувати такі команди:

```
g1=g1*2-1;
autocorr(g1(1,:));.
```

Таблиця 1.1 – Визначення місця включення зворотного зв'язку для найкращих пар

Кількість розрядів	Період	Кількість М-послідовностей	Місце підключення зворотного зв'язку	Кількість найкращих пар
3	7	2	(1,3) (2,3)	2
4	15	2	(1,4)	0
5	31	6	(2,5) (2,3,4,5)(1,2,4,5)	3
6	63	6	(1,6) (1,2,3,7) (2,3,4,7)	2
7	129	18	(4,7) (1,2,5,6) (2,3,5,6) (1,7) (1,3,6,7) (2,4,6,7)	6
8	255	16	(2,3,4,8) (3,5,6,8) (2,5,6,8) (1,3,5,8)	0
9	511	48	(4,9) (3,4,5,9) (4,5,8,9) (1,4,8,9) (2,3,5,9)	

В результаті обчислення отримуємо вектор значень автокореляційної функції [7,3,-1,-1.,-1, 3]. Графік автокореляційної функції Голд-послідовності для додатних зсувів зображено на рис.1.12.

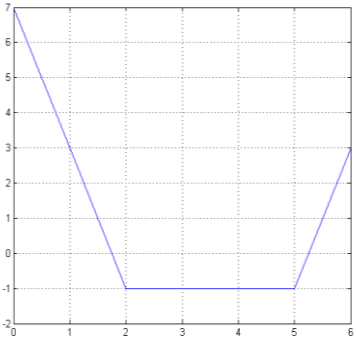

Рисунок 1.12 – Автокореляціна функція Голд-послідовності

Автокореляційна функція має максимальне значення у точці синхронізації, а для усіх інших значень часу вона має значення, що флуктуюють.

Взаємна кореляційна функція між отриманими послідовностями g1(1,:) g1 (2,:) визначиться як:

crosscorr(g1(1,:), g1(2, :)).

Результат розрахунків надає вектор значень крос-кореляційної функції [3,-1,-5, -1, 3, -1,-1] (рис.1.13).

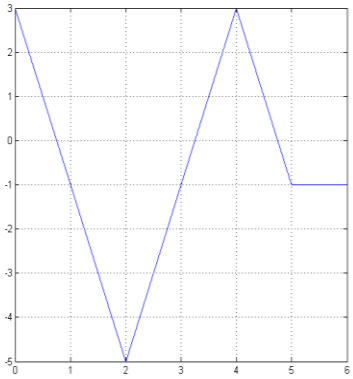

Рисунок 1.13 – Взаємна кореляційна функція для Голд-послідовностей

Значення вектору формуються з трьох цифр [-1, -t(n), t(n)-2], де t(n)=5, як це виходить за (2.17) [2].

Ортогональна Голд-послідовність реалізована шляхом додавання нульового біту на початку або кінці Голд-послідовності. Це можна виконати за допомогою оператора:

code = [goldseq(m3, m4, user),zeros(user, 1)];.

На рис.1.6 було зображено Голд-послідовність, яку було створено за допомогою двох М-послідовностей m1і m2. На рис. 1.14 зображено цю ж послідовність, але у форматі NRZ.

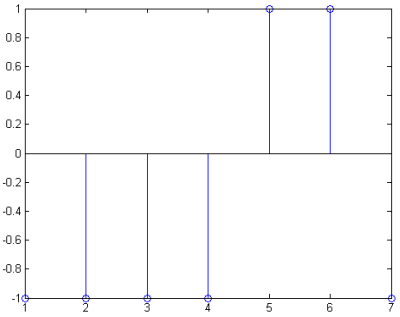

Рисунок 1.14 – Голд-послідовність у форматі NRZ

Відповідно на рис.1.15 зображено утворену ортогональну Голд-послідовність з восьми чіпів, як результат додавання нуля на кінці послідовності Голд-послідовності.

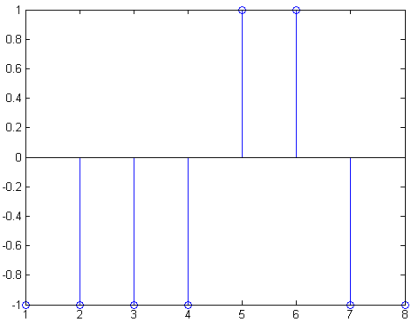

Рисунок 1.15 – Ортогональна Голд-послідовність

Автокореляційна функція для ортогональної Голд-послідовності зображена на рис.1.16.

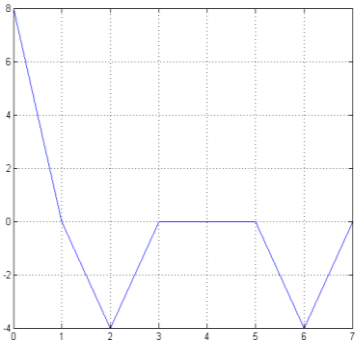

Рисунок 1.16 – Автокореляційна функція ортогональної Голд-послідовності

Якщо за допомогою сформованих раніше чотирьох М-послідовностей m1, m2, m3, m4 утворити дві Голд-послідовності, а потім за запропонованим алгоритмом перейти до двох ортогональних Голд-послідовностей, то можна обчислити взаємну кореляційну функцію. Результат обчислення взаємної кореляційної функції відображено на рис.1.17.

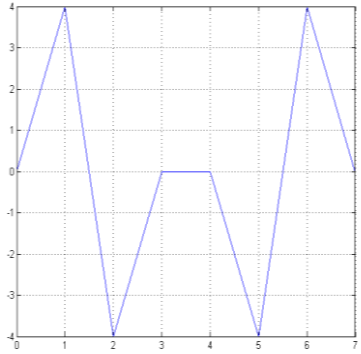

Рисунок 1.17 – Взаємна кореляційна функція ортогональної Голд-послідовності

Аналізуючи взаємні кореляційні функції для М-послідовностей (рис. 1.11), Голд-послідовностей (рис. 1.13) і ортогональних Голд-послідовностей (рис. 1.17) необхідно відзначити важливу особливість ортогональних Голд-послідовностей – на відміну від М-послідовностей та Голд- послідовностей, взаємна кореляційна функція для ортогональних Голд-послідовностей має нульове значення у точці синхронізації.

Повні лістинги підпрограм msec.m., goldseq.m., autocorr.m, crosscorr.m., а також shift.m – підпрограми, якаі реалізує циклічний зсув на один розряд наведено у додатку А.

2 СТРУКТУРА ГОЛОВНОЇ ПРОГРАМИ

У другому розділі розглядається побудова моделі каналу зв'язку з використанням методу DS-CDMA. Відповідно до технології DSSS у модуляторі відбувається операція розширення спектра прямою послідовністю з наступним виконанням зворотної операції стиску спектра на приймальному боці.

Блок-схема алгоритму формування моделі каналу зв'язку з використанням методу DSSS наведена на рис. 2.1. Відповідно усі етапи формування і обробки сигналів за цією моделлю реалізовані у головній програмі dscdma.m. Для виводу графічних залежностей BER =f(E_b/N_0) застосовується інша головна програма ber_cdma, яка практично повторює програму dscdma.m, але містить додатковий цикл за параметром E_b/N_0, що, власне, і дозволяє отримати графічні залежності.

По-перше, у головній програмі вводяться початкові дані для моделювання. Введемо такі позначення:

sr [кбіт/с] – швидкість передачі символів;

br [кбіт/с] – швидкість передачі інформаційних бітів;

ml – кількість бітів, що передаються за один символ;

nd – кількість переданих символів, для яких буде проводиться розрахунок ймовірності помилкового прийому за одну реалізацію випадкового процесу (один цикл моделювання);

ebn0 – початкове значення енергії, що приходиться на один біт E_b, до спектральної щільності шуму N_0 у визначеній смузі частот;

IPOINT – порядок передискритизації;

irfn –кількість ліній затримки у формуючому фільтр;

alfs – коефіцієнт закруглення (параметр формуючого фільтра, який визначає розширення смуги частот порівняно з ідеальною АЧХ).

Швидкість передавання символів задана рівною 256000 біт/с, або 256 кбіт/с, кількість бітів ml, що переносяться за один радіосимвол для QPSK **ml=2**, відповідно бітова швидкість **br** визначається як результат перемноження символьної швидкості на параметр **ml**, що у випадку QPSK означає, що бітова швидкість удвічі більша за символьну швидкість.

Кількість символів, що підлягають аналізу для однієї реалізації випадкового процесу прийнята рівною **nd=100**, початкове значення параметру $E_b/N=3$.

Коефіцієнти імпульсної характеристики фільтра формуючого фільтра у кожному з каналів розраховуються за допомогою підпрограми **hrollfcoef**(irfn, IPOINT,sr,alfs,1/0). Параметрами підпрограми є кількість ліній затримки фільтра irfn, раніше визначені порядок передискретизації IPOINT і символьна швидкість sr, а також коефіцієнт закруглення alfs. Значення останнього п'ятого параметра функції **hrollfcoef**, яке може приймати значення або 1, або 0, визначає приналежність фільтра до передавача каналу зв'язку (відповідний параметр рівняється 1), або до приймача (параметр прийнято рівним 0).

Для моделювання кількість ліній затримки прийнята рівною **irfn=21**, а значення коефіцієнта округлення **alfs=0.5**.

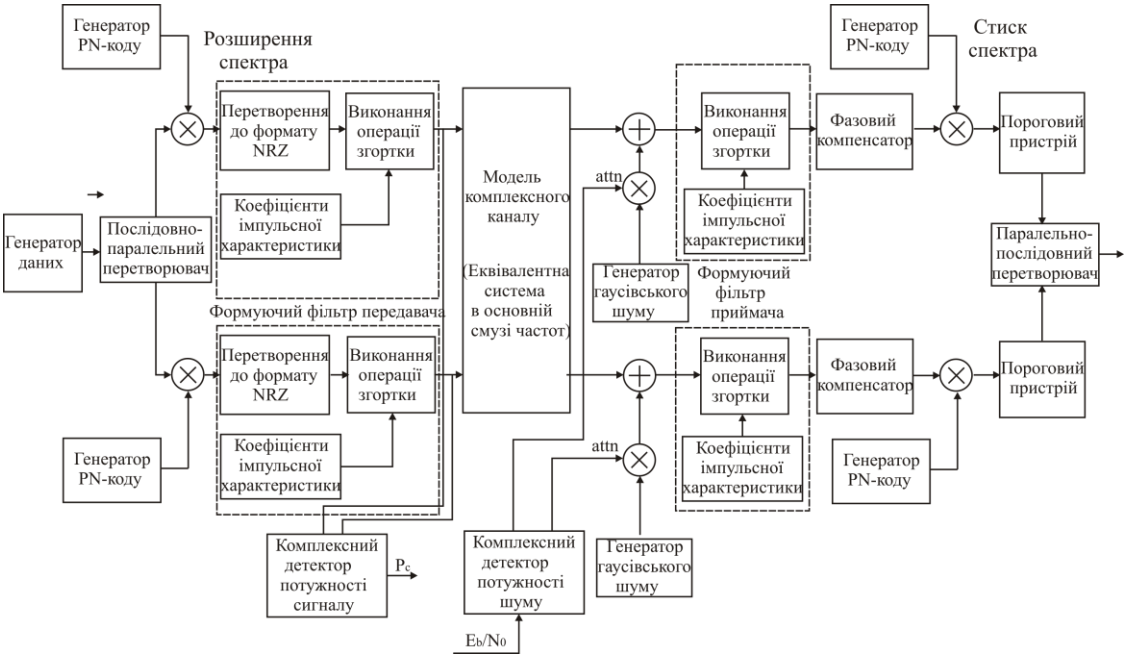

Рисунок 2.1 – Блок-схема каналу зв’язку з використанням DS-CDMA технології

В термінах алгоритмічної мови програмування середовища Matlab блок введення початкових даних для методу DSSS набуває наступного вигляду:

```
% dscdma.m
% Simulation program to realize DS-CDMA transmission system
%**********************Preparation part**************
 sr=256000.0;     % Symbol rate
ml=2;                        % ml : Number of modulation levels
br=sr.*ml;          % bit rate
nd=100;             % Number of symbols
ebn0=3;             % Eb/N0
IPOINT=8;                  % Number of oversamples
irfn=21;% Number of taps
alfs=0.5;           % Rolloff factor
[xh]=hrollfcoef(irfn, IPOINT,sr,alfs,1);
          % Tranamitter filter coefficients
[xh2]=hrollfcoef(irfn, IPOINT,sr,alfs,0);
          % Receiver filter coefficients
```

Різниця між програмами які застосовуються безпосередньо для методу QPSK [1] і реалізацією такої модуляції у синхронних DS-CDMA системах полягає у параметрах, які використовуються для визначення кількості користувачів і генерації PN-коду.

Як відзначалося, у програмному комплексі пропонується реалізувати генерацію трьох видів розширюючого коду – М-послідовностей, послідовностей Голда і ортогональних Голд-послідовностей. Вибір тієї чи іншої послідовності пропонується здійснити за допомогою операторів **switch, case** і змінної **sec**. Так, задаючи значення змінної sec = 1 реалізується генерація М-послідовності, для sec = 2 відбувається генерація Голд-послідовності і, відповідно, для sec = 3 формується головною програмою ортогональна Голд-послідовність.

В синхронних DS-CDMA системах кількість кодів, що повинна бути призначена різним користувачам повинна рівнятися довжині кодів, що використовуються. Відповідно, довжину кодової послідовності необхідно обирати більшою за кількість користувачів.

Наприклад, якщо використовується трьохрозрядний регістр зсуву, то кількість користувачів, яка буде підлягати аналізу (змінна **user**) не повинна перевищувати семи (7).

Для генерації кодової послідовності необхідно задати кількість розрядів (змінна stg), позиції розташування зворотного зв'язку (змінна ptap) і початковий вміст регістрів (змінні **regi1, regi2** відповідно).

Як було визначено, для генерації Голд-послідовностей і ортогональних Голд-послідовностей необхідно сформувати дві М-послідовності. Сформовані ПВП зберігаються як змінна **code.**

В термінах середовища Matlab блок завдання параметрів для моделювання PN-кодів набуває такого вигляду:

```
user=1;
sec=1;
stage=3;
ptap==[1 3];
ptap==[2 3];
regi1=[1 1 1];
regi2=[1 1 1];
```

Безпосередньо генерація трьох видів коду реалізується за допомогою наступного блоку:

Безпосередньо генерація трьох видів коду реалізується за допомогою наступного блоку:

```
switch seq
case 1                    % M-sequence
   code=msec(stage, ptap1, regi1, user);

case 2                    % Gold-sequence

m1=msec(stage, ptap1, regi1);
m2=msec(stage, ptap2, regi2);
 code=goldsec(m1, m2, user);


case 3                    % Orthogonal Gold-sequence
```

m1=msec(stage, ptap1, regi1);
m2=msec(stage, ptap2, regi2);
code=[goldsec(m1, m2, user), (user,1];

Відповідно після виконання цього блоку реалізується генерація коду і згенерований код зберігається як змінна **code**. Змінна **code** практично представляє матрицю, порядок якої визначається результатом перемноження кількості користувачів на довжину кодової послідовності. Генерація М-послідовності і Голд послідовності відбувається за допомогою функцій mseq.m і goldsec.m відповідно.

Оскільки згенровані послідовності складаються з ансамблю тільки нульових та одиничних значень, то наступною операцією є перетворення кодових послідовностей до формату NRZ без постійної складової з визначенням довжини сформованої ПВП:

code= code*2-1
clen=length(code)

Подалі починається основне тіло програми.

Введемо наступні позначення:

nloop – кількість циклів моделювання;

nod – кількість переданих бітів;

noe – кількість помилково декодованих бітів на приймальній стороні.

Для того, щоб адекватно визначити імовірнісні характеристики каналу зв'язку задамо кількість циклів моделювання (кількість реалізацій випадкового процесу для послідовності з 100 біт) достатньо великою – приймаємо значення **nloop** = 1000. Початкові значення кількості переданих (змінна **nod**) і помилково декодованих бітів (змінна **noe**) приймаються на початку циклу рівними нулю.

Змінну циклу позначимо через **iii,** а початок циклу задається оператором **for iii=1:nloop.** Відповідний фрагмент програми набуває наступного вигляду:

```
%*****************Start calculations**********************
nloop=1000;    % Number of simulation loops
noe=0;         % Number of  error data
nod=0;         % Number of transmitted data
for  iii=1:nloop
….
```

Тіло циклу

…

end

Далі у тілі циклу виконуються усі етапи обробки сигналу за блок-схемою з рис. 2.1.

Спочатку відбувається генерація випадкової послідовності інформаційних даних з однаковою ймовірністю з'явлення нулів і одиниць за допомогою вбудованої функції **rand**. Для сформованого інформаційного потоку реалізується робота I,Q – модулятора QPSK сигналів згідно з алгоритмом розглянутим е [], тобто відбувається формування послідовностей парних і непарних бітів відповідно до алгоритму формування канальних символів (рис. 2.2).

Подалі реалізується другий етап модуляції, а саме, операція розширення спектра, тобто відбувається перемноження отриманого модульованого сигналу і згенерованої кодової послідовності за допомогою підпрограми **spread.m.**

Подальша обробка сигналів проходить повністю подібно до звичайного методу QPSK. Відповідно до принципів цифрової обробки сигналів здійснюється операція передискретизації з порядком пердискретизації IPOINT = 8 і наступною фільтрацією у формуючому фільтрі Найквіста.

Реалізація передискретизації і фільтрації у програмному комплексі реалізується за допомогою підпрограм compoversamp2.m і compconv2.m відповідно, повний лістинг яких наведено у додатку.

Тобто, наведений нижче блок операторів головної програми практично формує вихідний сигнал передавача в основній смузі частот для запропонованої моделі:

```
%******************Transmitter**********************
%******************Data generation**********************
 data1=rand(user,nd*ml)>0.5;    % rand: built in function
%******************QPSK Modulation**********************
 [ich,qch]=qpskmod(data1,user,nd,ml);
%******************Spreading**********************
[ich1,qch1]=spread(ich,qch,code));   % spreading
%******************Oversampling**********************
[ich2,qch2]=compoversamp2(ich1,qch1, IPOINT);
```

%***************Convolution*******************
[ich3,qch3]=compconv (ich2,qch2,xh); % filter.**

На рис. 2.3 зображено перші двадцять відліків вхідної випадкової послідовності з однаковою ймовірністю з’явлення одиниць і нулів, як результат виконання першого з наведених операторів (функція **rand**).

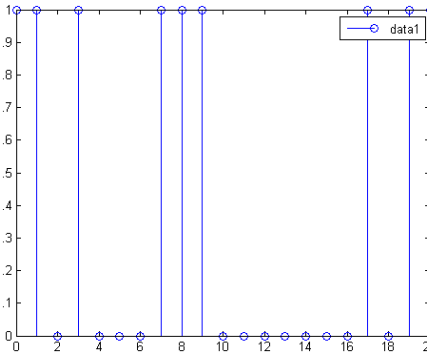

Рисунок 2.3 – Перші двадцять відліків вхідної інформаційної послідовності

На рис. 2.4 і рис. 2.5 наведені епюри перших десять канальних символів I, Q-модулятора, як результат роботи підпрограми qpskmod.m.

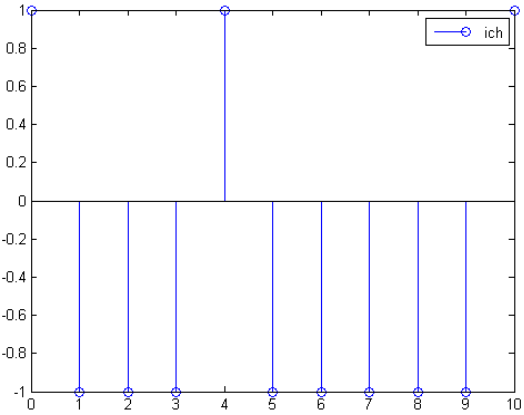

Рисунок 2.4 – Перші десять відліків послідовності канальних символів I-каналу

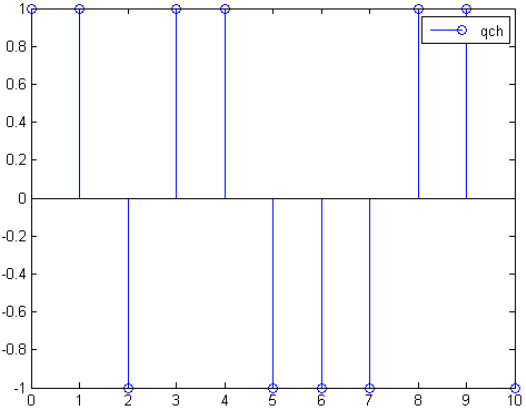

Рисунок 2.5 – Перші десять відліків послідовності канальних символів Q –каналу

На рис. 2.6 зображено один період згенерованої трьохкаскадної М-послідовності.

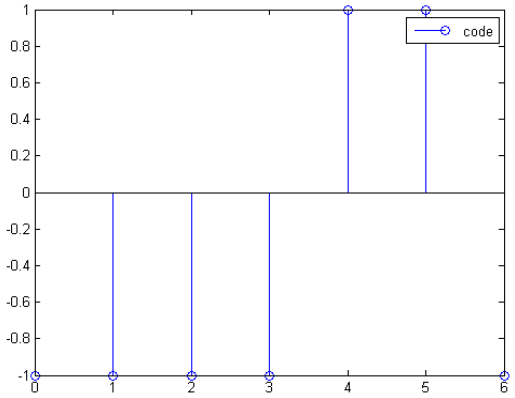

Рисунок 2.6 – Згенерована М-послідовність

На рис. 2.7, рис. 2.8 продемонстровано операцію розширення спектра у часовій області для обох каналів (підпрограма **spread.m**) для перших десяти символів кожного з каналів модулятора.

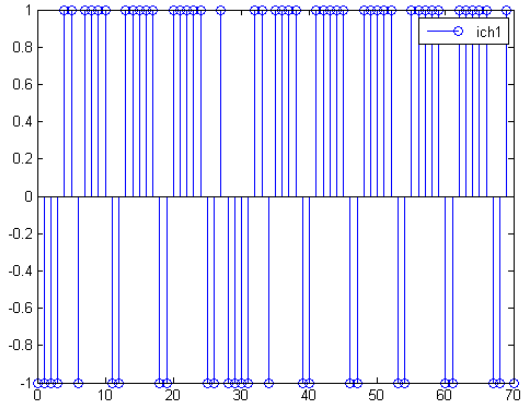

Рисунок 2.7 – Результат виконання операції розширення спектра в І-каналі

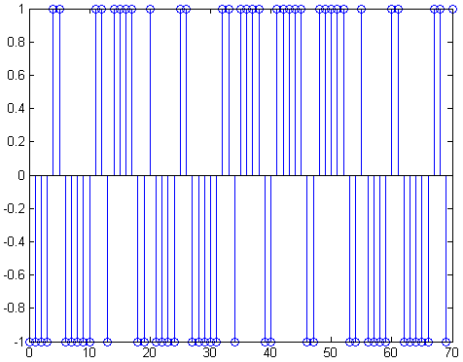

Рисунок 2.8 – Результат виконання операції розширення спектра в Q-каналі

Більш детально цей процес проілюстровано на рис. 2.9, рис. 2.10, де для обох каналів взяті тільки два перших канальних символи, яким у закодованому вигляді відповідають 14 дискретів (чіпів).

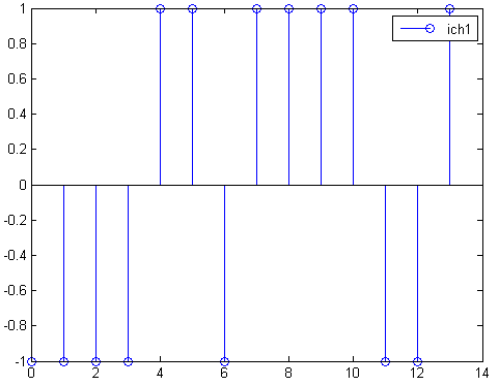

Рисунок 2.9 – Операція розширення спектра для двох перших символів I-каналу

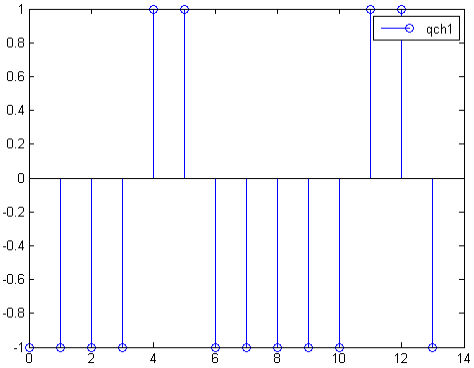

Рисунок 2.10 – Операція розширення спектра для двох перших символів Q-каналу

Наступним кроком є виконання операції передискретизації. Для більшої наочності фізичного сенсу цієї операції процес передискретизації продемонструємо тільки для одного інформаційного біта, якому відповідають закодована послідовність з 7 чіпів. Відповідно на рис. 2.11, рис. 2.12 відображено процес передискретизації в обох каналах з порядком передискретизації IPOINT = 8.

Якщо цю операцію провести для перших десяти канальних символів у кожному каналі модулятора, то епюри у розрізі саме операції передискретизації виходять неінформативними (рис. 2.13, рис. 2.14).

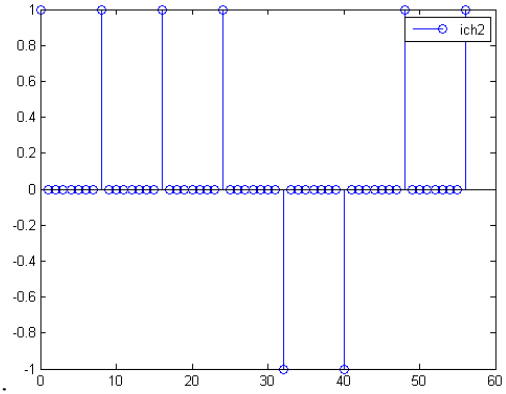

Рисунок 2.11 – Операція пере дискретизації для одного інформаційного біта I-каналу, якому відповідають сім закодованих чіпів

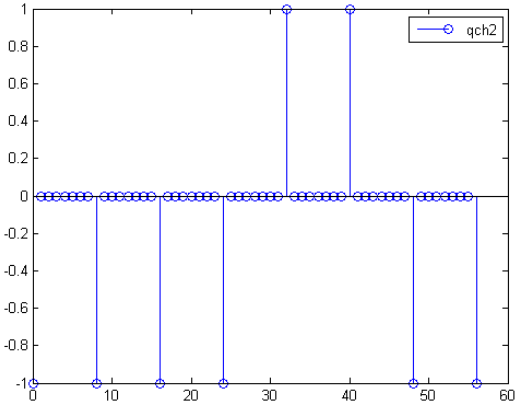

Рисунок 2.12 – Операція передискретизації для одного інформаційного біта I-каналу, якому відповідають сім закодованих чіпів.

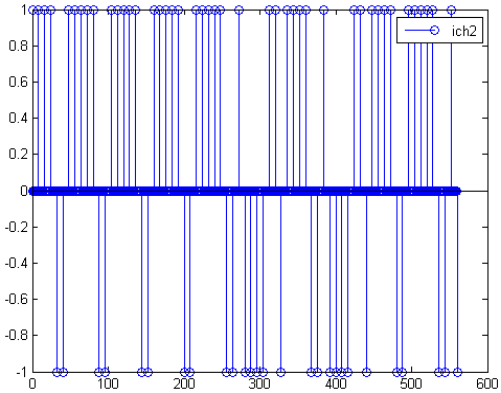

Рисунок 2.13 – Операція передискретизації для десяти закодованих канальних символів I-каналу

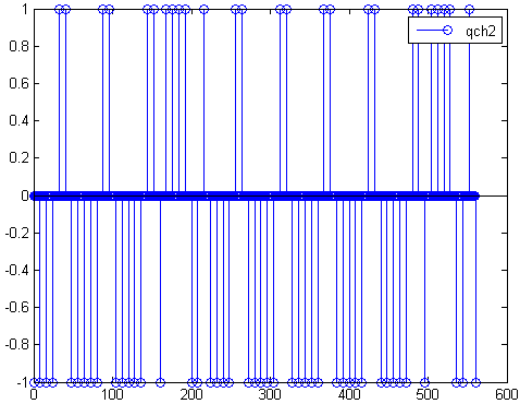

Рисунок 2.14 – Операція передискретизації для десяти закодованих канальних символів Q-каналу

Далі у кожному каналі відбувається фільтрація формуючим фільтром Найквіста. Результати фільтрації у кожному каналі як результат виконання операції згортки продемонстровано на рис. 2.15, рис. 2.16.

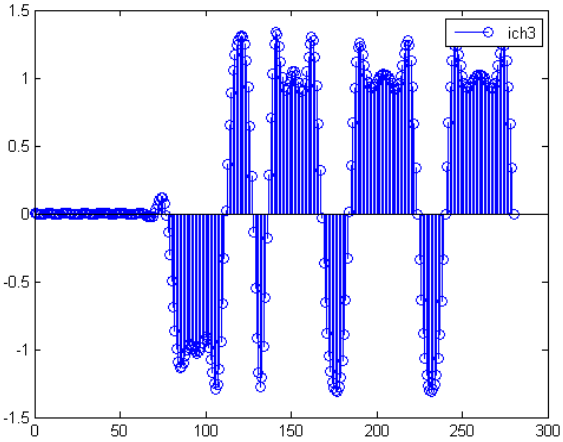

Рисунок 2.15 – Епюри на виході формуючого фільтра I-каналу

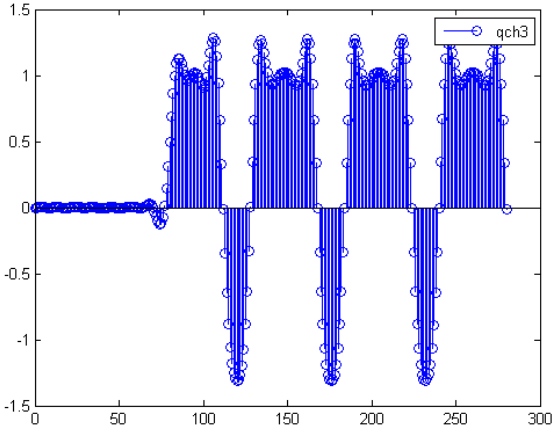

Рисунок 2.16 – Епюри на виході формуючого фільтра I-каналу

Для здійснення модуляції BPSK у кожному каналі модулятора в реальних системах необхідно від цифрових вибірок перейти до аналогового подання модулюючих I, Q символів, тобто необхідно цифрові вибірки подати до цифро-аналогового перетворювача. Результат виконання цієї операції зображено на рис. 2.17, рис. 2.18.

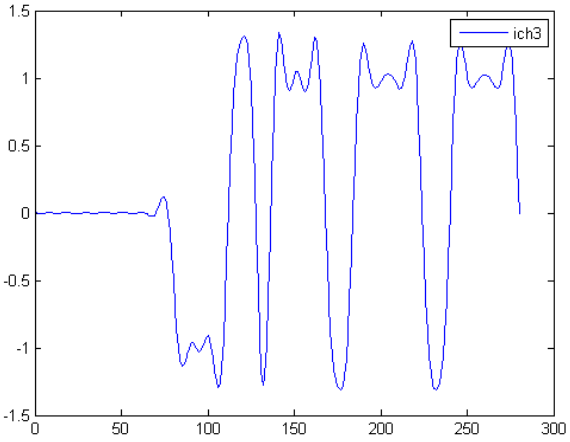

Рисунок 2.17 – Модулюючі символи I-каналу

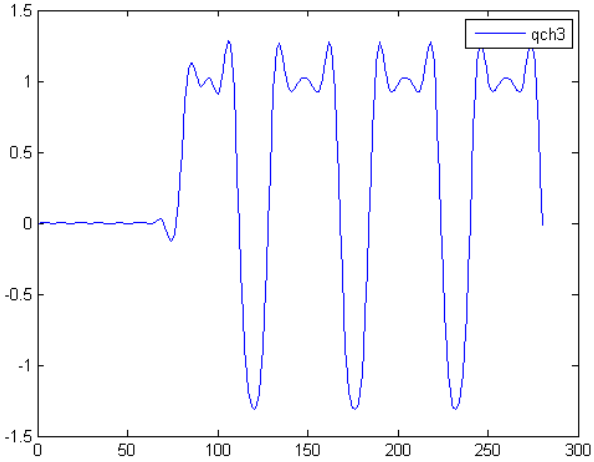

Рисунок 2.18 – Модулюючі символи Q-каналу

Оскільки процеси на радіочастоті в моделі не розглядаються, то отримані символи можна вважати вихідними сигналами кожного з

каналів модулятора, які передаються через еквівалентний канал зв'язку в основній смузі частот.

У приймачі до переданих закодованих бітів додається білий гаусівський шум відповідно до заданого відношення E_b/N_0. Потім прийняті дані проходять таку ж фільтрацію у формуючому фільтрі, як це відбувалося на передавальному боці. Додавання білого шуму до прийнятих даних реалізується за допомогою підпрограми comb2.m. Повний лістинг програми comb2.m. наведено у додатку.

```
spow=sum(rot90(ich3.^2+qch3.^2))/nd;
attn=sqrt(0.5*spow*sr/br*10^(-ebn0/10));
[ich6,qch6]=comb2(ich5,qch5,attn);
[ich7, qch7]=compconv2(ich6,qch6,xh2);
Sample=irfn*IPOINT+1;
ich8= ich7(: , sample: IPOINT: IPOINT*nd*clen+sample-1)
qch8= qch7(: , sample: IPOINT: IPOINT*nd*clen+sample-1)
```

Останні два оператори усувають передискретизацію, яка вводилася на передавальному боці. у I та Q каналах відповідно. Тобто, на цьому етапі отримуються сигнал з розширеним спектром який є сумою сигналів від усіх користувачів.

Наступною операцією є зворотний стиск спектра прийнятого сигналу у кореляторі, що дозволяє визначити саме сигнал від корисного користувача. Кореляційна обробка прийнятого сигналу (стиск спектру) реалізується у програмному комплексі підпрограмою **despread,m**:

```
[ich9 qch9]=despread[ich8,qch8,code;.
```

Після декореляції відбувається демодуляція прийнятого сигналу за алгоритмом роботи I,Q демодулятора.

Останнім блоком головної програми є визначення помилково прийнятих бітів для усіх користувачів з наступним визначенням залежності коефіцієнта бітової помилки від відношення сигнал/шум:

```
[demodata]=qpskdemod(ich9,qch9,user,nd,ml);
 %******************Bit Error Rate
(BER)**********************
 noe2=sum(sum(abs(data1-demodata)));
   % sum: built in function
```

```
nod2=user*nd*ml; %
noe=noe+noe2;
nod=nod+nod2;
fprintf('%d\t%e\n',iii,noe2/nod2);    % fprint: built in function

end     % for iii=1:nloop
 %******************Output  result **********************
 ber=noe/nod;
```

В результаті запуску головної програми dscdma.m відбувається формування епюрів сигналів для усіх етапів обробки сигналу у модуляторі і демодуляторі з побудовою фазового сузір'я для заданого значення E_b/N_0.

Моделювання залежності коефіцієнта бітової помилки BER для AWGN каналу зв'язку реалізовано у програмі BER_CDMA, яка відрізняється від програми dscdma введенням додаткового циклу за параметром E_b/N_0 з відповідним виведенням графічних залежностей $BER=f(E_b/N_0)$. Для проведення порівняльного аналізу на графіках також представлені залежності BER для AWGN каналу і крива релеївського порогу.

3 ПОРЯДОК ВИКОНАННЯ КУРСОВОГО ПРОЄКТУ

3.1 Генерація і оцінка кореляційних властивостей псевдовипадкових послідовностей

Варіанти технічного завдання приведені у таблиці 3.1. Відповідно до заданої у технічному завданні виду псевдовипадкової послідовності здійснити генерацію двох різних ПВП одного типу і обчислити автокореляційну і взаємну кореляційну функції.

Для генерації ПВП і обчислення кореляційних функцій необхідним є такий набір програмних модулів:

main.m – головна програма, в якій реалізується вибір типу послідовності, завдання кількості розрядів регістру, початкових станів розрядів і вивід результатів;

msec.m – підпрограма генерації М-послідовностей;

goldseq.m – підпрограма генерації Голд-послідовностей;

autucorr.m– підпрограма обчислення автокореляційної функції;

crosscor.m – підпрограма обчислення взаємної кореляційної функції;

shift.m. – підпрограма здійснення циклічного зсуву на один розряд.

3.2 Експериментальна частина

3.2.1 Відкрити середовище Matlab. Створити нову директорію CDMA, в яку необхідно розмістити основну програму, а також усі підпрограми, що необхідні для моделювання.

3.2.2 Скопіювати файл основної програми **main.m**, який знаходиться у додатку А методичних вказівок. Зберегти скопійований файл під ім’ям main.m у створеній директорії CDMA.

3.2.3 Скопіювати у додатку А підпрограму **msec.m**, і повторити процедуру створення нового М-файлу в директорії CDMA.

3.2.4 Скопіювати у додатку А підпрограму **golgsec.m**, і повторити процедуру створення нового М-файлу в директорії CDMA.

3.2.5 Скопіювати у додатку А підпрограму **shift.m**, що виконує операцію передискретизації, і повторити процедуру створення нового М-файлу в директорії CDMA.

Таблиця 3.1 – Завдання на курсовий проект

Варіант	Псевдовипадкова послідовність для проведення моделювання	Кількість розрядів регістру зсуву	Початкові стани першого і другого регістрів	Коефіцієнт округлення α	Порядок передискретизації
1	2	3	4	5	6
1	М-послідовність	3	[1 1 1], [1 1 0].	0,3	8
2	Голд-послідовність	3	[1 0 1], [1 1 0].	0,5	8
3	Ортогональна Голд-послідовність	3	[1 1 1], [1 0 0].	0,25	8
4	М-послідовність	3	[0 1 1], [1 1 1].	0,4	8
5	Голд-послідовність	3	[1 1 0], [1 1 1].	0,3	8
6	Ортогональна Голд-послідовність	3	[1 0 1], [0 11]	0,5	8
7	М-послідовність	3	[0 1 1], [1 0 1].	0,35	8
8	Голд-послідовність	3	[1 1 1], [1 0 0].	0,3	8
9	Ортогональна Голд-послідовність	3	[1 1 0], [0 1 1].	0,5	8

Кінець таблиці 3.1

1	2	3	4	5	6
10	М-послідовність	4	[1 1 1 1], [1 1 0 0].	0,25	8
11	Голд-послідовність	4	[1 0 1 1], [1 1 0 0].	0,25	8
12	Ортогональна Голд-послідовність	4	[1 1 1 0], [1 0 0 0].	0,5	8
13	М-послідовність	4	[0 1 1 1], [1 1 1 0].	0,3	8
14	Голд-послідовність	4	[1 1 0 1], [0 1 1 1].	0,4	8
15	Ортогональна Голд-послідовність	4	[1 1 0 1], [0 0 11]	0,3	8
16	М-послідовність	4	[0 0 1 1], [1 0 1 0].	0,5	8
17	Голд-послідовність	4	[1 1 1 1], [1 0 0 1].	0,25	8
18	Ортогональна Голд-послідовність	4	[1 1 0 1], [1 0 10].	0,3	8
19	М-послідовність	5	[1 1 1 1 1], [1 0 0 1 1]	0,5	8
20	Голд-послідовність	5	[1 1 0 1 1], [1 0 1 1 0]	0,25	8

3.2.6 Скопіювати у додатку А підпрограму **autocorr.m**, що обчислює автокореляційну функцію, і повторити процедуру створення нового М-файлу в директорії CDMA.

3.2.7 Скопіювати у додатку А підпрограму **crosscor.m** що обчислює автокореляційну функцію, і повторити процедуру створення нового М-файлу в директорії CDMA.

3.2.8 Вибрати тип послідовності згідно з варіантом завдання.

Для цього в головній програмі main необхідно задати значення змінної sec. Так, якщо задається значення змінної sec = 1, то реалізується генерація М-послідовності, для sec = 2 відбувається генерація Голд-послідовності і, відповідно, для sec = 3 формується головною програмою ортогональна Голд-послідовність.

Для завдання значення sec необхідно відкрити у редакторі програму **main** і встановити значення sec відповідно до завдання.

3.2.9 Задати місце включення зворотного зв'язку (підключення генератора парності). Для цього в головній програмі **main** задати значення операторів **ptap1, ptap2** згідно з таблицею 3.1.

3.2.10 Задати початкові стани чотирьох регістрів зсуву. Для цього в головній програмі **main** задати значення операторів **regi1, regi2 regi3, regi4**. Для перших двох регістрів початкові значення взяти з варіанту завдання. Для третього і четвертого регістру провести довільну зміну в одному з розрядів регістрів **regi1, regi2 (**неприпустимою є тільки комбінація з одних нулів).

Зберігти введені зміни операторі у програмі main.m.

3.2.11 У командній строчці вікна Command Window набрати назву основної програми **main** і запустити аналіз натисканням клавіші Enter.

За результатами моделювання отримати наступні епюри:

– епюри сформованих двох варіантів послідовності відповідно до заданої ПВП (один період);

– автокореляційну функцію для обох варіантів ПВП;

– взаємну кореляційну функцію для обраного типу ПВП.

3.3 Моделювання каналу зв'язку з використанням технології DS-CDMA

Для проведення моделювання DS-CDMA необхідним є наступний блок підпрограм:

dscdma.m – головна програма, в якій реалізуютьться усі процеси обробки сигналів відповідно до блок-схеми моделі (рис.2.1). а також вивід графічних залежностей;

ber_cdma – головна програма, в якій реалізуєтьться розрахунок залежності BER від параметра E_b/N_0 з виведенням відповідних графічних залежностей графічних залежностей;

spread.m – підпрограма яка реалізує процес розширення спектру у демодуляторі

despread.m – підпрограма, яка реалізує процес стиску спектру у модуляторі

msec.m – підпрограма генерації М-послідовностей;

goldseq.m – підпрограма генерації Голд-послідовностей;

autucorr.m– підпрограма обчислення автокореляційної функції;

crosscor.m – підпрограма обчислення взаємної кореляційної функції;

shift.m. – підпрограма здійснення циклічного зсуву на один розряд

Повні лістинги програм **dscdma.m** і **ber_cdma,** які, власне, формують модель каналу, а також підпрограм, що реалізують посимвольне перемноження символів ПВП та модулюючих символів на передавальному боці (операція розширення спектра) – **spread.m** і посимвольне перемноження прийнятої закодованої послідовності з ПВП приймача (операція стиску спектра) – **despread.m** наведені у додатку Б.

Також необхідним є набір підпрограм, які необхідні для реалізації модема QPSK:

hrollcoef.m – коефіцієнти імпульсної характеристики фільтра Найквіста;

qpskmod.m – формування канальних символів для методу QPSK;

qpskdemod.m – алгоритм роботи QPSK демодулятора;

compoversamp2.m – передискретизація в обох каналах I, Q модулятора;

compconv2.m – операція згортки в обох каналах I, Q модулятора;

comb2.m – додавання гаусівського шуму до обох каналів I, Q демодулятора.

3.3.1 Створити нову директорію DSCDMA, в яку необхідно розмістити основну програму, а також усі підпрограми, що необхідні для моделювання.

3.3.2 Скопіювати файл основної програми **dscdma.m**, який знаходиться у додатку Б методичних вказівок. Зберегти скопійований файл під ім'ям **dscdma**.**m** у створеній директорії DSCDMA.

3.3.3 Скопіювати у додатку А підпрограму **msec.m**, і повторити процедуру створення нового М-файлу в директорії DSCDMA.

3.3.4 Скопіювати у додатку А підпрограму **golgsec.m**, і повторити процедуру створення нового М-файлу в директорії DSCDMA.

3.3.5 Скопіювати у додатку А підпрограму **shift.m**, що виконує операцію передискретизації, і повторити процедуру створення нового М-файлу в директорії DSCDMA.

3.3.6 Скопіювати у додатку Б підпрограму **spread.m**, і повторити процедуру створення нового М-файлу в директорії CDMA.

3.3.7 Скопіювати у додатку Б підпрограму **despread.m**, і повторити процедуру створення нового М-файлу в директорії CDMA.

3.3.8 Скопіювати у додатку В підпрограму **qpskmod.m**, що моделює алгоритм роботи модулятора для методу QPSK і повторити процедуру створення нового М-файлу в директорії CDMA.

3.3.9 Скопіювати у додатку В підпрограму **hrollcoeff.m**, що виконує операцію обчислення коефіцієнтів імпульсної характеристики формуючого фільтра, і повторити процедуру створення нового М-файлу в директорії CDMA.

3.3.10 Скопіювати у додатку В підпрограму **compconv2.m**, що виконує операцію згортки імпульсної характеристики формуючого фільтра і вхідної послідовності, і повторити процедуру створення нового М-файлу в директорії CDMA.

3.3.11 Скопіювати у додатку В підпрограму **compoversamp2.m**, що виконує операцію передискретизації, і повторити процедуру створення нового М-файлу в директорії CDMA.

3.3.12 Скопіювати у додатку В підпрограму **qpskdemod.m**, що моделює алгоритм роботи демодулятора для методу QPSK і повторити процедуру створення нового М-файлу в директорії CDMA.

3.3.13 Провести моделювання каналу DS-CDMA у випадку роботи одного користувача і заданого виду ПВП.

Для моделювання роботи модему необхідно відкрити у редакторі головну програму dscdma.m і задати значення параметрів відповідно до технічного завдання:

– значення змінної **sec**;

– значення **ptap1, ptap2,** які визначають місця підключення зворотного зв'язку у регістрі зсуву;

– початкові значення регістрів **regi1, regi2 regi3, regi4**;

– встановити змінну **user=1;** (кількість користувачів, що працюють одночасно).

3.3.14 У командній строчці вікна Command Window набрати назву основної програми **dscdma** і запустити аналіз натисканням клавіші Enter.

За результатами моделювання отримати наступні епюри:

1) вхідну послідовність даних у вигляді цифрових відліків;

2) підпослідовність цифрових символів I-каналу;

3) підпослідовність цифрових символів Q-каналу;

4) один період згенерованого PN-коду;

5) два кодованих за методом DSSS символи I-каналу;

6) два кодованих за методом DSSS символи Q-каналу;

7) десять закодованих за методом DSSS символів I-каналу;

8) десять закодованих за методом DSSS символів Q-каналу;

9) один символ з розширеним спектром в I-каналі після операції передискретизації;

10) один символ з розширеним спектром в Q-каналі після операції передискретизації;

11) десять символів з розширеним спектром в I-каналі після операції передискретизації;

12) десять символів з розширеним спектром в Q-каналі після операції передискретизації;

13) результат фільтрації формуючим фільтром Найквіста сигналу з розширеним спектром в I-каналі до ЦАП;

14) результат фільтрації формуючим фільтром Найквіста сигналу з розширеним спектром в Q-каналі до ЦАП;

15) вихідний сигнал I-каналу модулятора після ЦАП;

16) вихідний сигнал Q-каналу модулятора після ЦАП;

17) сумарний сигнал від роботи декількох користувачів, що працюють одночасно, в I-каналі після ЦАП (у випадку одного користувача повторюють епюру за пунктом 15);

18) сумарний сигнал від роботи декількох користувачів, що працюють одночасно, в Q-каналі після ЦАП (у випадку одного користувача повторюють епюру за пунктом 16);

19) епюру сигналу на вході I-каналу демодулятора до АЦП;

20) епюру сигналу на вході Q-каналу демодулятора до АЦП;

21) епюру дискретизованого сигналу на вході I-каналу демодулятора після АЦП;

22) епюру дискретизованого сигналу на вході Q-каналу демодулятора після АЦП;

23) епюру сигналу на виході формуючого фільтра I-каналу демодулятора;

24) епюру сигналу на виході формуючого фільтра Q-каналу демодулятора;

25) епюру сигналу I-каналу демодулятора після усунення операції передискретизації;

26) епюру сигналу Q-каналу демодулятора після усунення операції передискретизації;

27) епюру сигналу I-каналу демодулятора після операції зворотного стиску спектра;

28) епюру сигналу Q-каналу демодулятора після операції зворотного стиску спектра;

29) фазове сузір'я як результат роботи модему для каналу з CDMA.

3.3.15 Збільшити кількість користувачів, що працюють одночасно, до 2. Для цього відкрити головну програму cdma і встановити значення змінної **user=2;**. Зменшити кількість епюр, що виводяться у вигляді графічних залежностей (оператори figure). З блоку виводу графіків залишити тільки оператори, які відповідають п.13,14, 15,16, 23,24 . Для усіх інших операторів проставити на

початку оператора символ коментаря. Зберегти модифікований файл. Запустити аналіз.

3.3.16 Збільшити кількість користувачів, що працюють одночасно, до 5. Для цього відкрити головну програму cdma і встановити значення змнної **user=5**;. Зберігати модифікований файл. Запустити аналіз.

3.4 Моделювання залежності BER від відношення Eb/N0 для різної кількості користувачів

3.4.1 Відкрити у редакторі програму **ber_cdma**. Задати значення змінної **user = 1;**. Зберегти файл. У командній строчці вікна Command Window набрати назву основної програми **ber_cdma** і запустити аналіз натисканням клавіші Enter. Результатом роботи програми виступає графічна залежність BER від відношення Eb/N0.

3.4.2 Відкрити у редакторі програму **ber_cdma**. Задати значення змінної user=4;. Зберегти файл. У командній строчці вікна Command Window набрати назву основної програми **ber_cdma** і запустити аналіз натисканням клавіші Enter. Отримати графічну залежність BER у випадку одночасної роботи чотирьох користувачів.

3.4.3 Відкрити у редакторі програму **ber_cdma**. Задати значення змінної user=7;. Зберегти файл. У командній строчці вікна Command Window набрати назву основної програми **ber_cdma** і запустити аналіз натисканням клавіші Enter. Отримати графічну залежність BER у випадку одночасної роботи семи користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Harada H. Simulation and software radio for mobile telecommunications [Текст] / H. Harada, R. Prasad. – Artech House, 2003. – 465 p.

2. Методичні вказівки до лабораторних робіт з дисципліни «Широкосмугові технології телекомунікацій» для студентів спеціальності 172 «Телекомунікації та радіотехніка» всіх форм навчання. Частина І / Укл. В.С. Кабак, Г.М. Сидоренко. – Запоріжжя: НУ «Запорізька політехніка», 2019. – 68 с.

3. Методичні вказівки до самостійної роботи з дисципліни «Широкосмугові технології телекомунікацій» для студентів спеціальності 172 «Електронні комунікації та радіотехніка» всіх форм навчання. / Укл. В.С. Кабак, Г.В. Мороз, Г.М. Сидоренко. – Запоріжжя: НУ «Запорізька політехніка», 2019. – 60 с.

ДОДАТОК А

```
% main.m
user = 1;              % number of users
seq =2;                % 1: M-sequence    2: Gold   3:Orthigonal Gold
 stage=4;              % number of stages
ptap1 =[2,3];         %position of taps for 1st
ptap2 =[2,3];         %position of taps for 2nd
ptap3=[2,3];
ptap4=[2,3];
regi1 =[0,1,1,1];     % initial value of register for 1st
regi2=[1,1,1,1];

   regi3=[1,1,1,0];
   regi4=[1,0,0,1];

  %**********Generation of the spreading code***************
switch seq
case 1
        code = msec(stage, ptap1, regi1,user);
        code1 = msec(stage, ptap2, regi2,user);
case 2
        m1 = msec(stage, ptap1, regi1);
        m2 = msec(stage, ptap2, regi2);

        m3 = msec(stage, ptap3, regi3);
        m4 = msec(stage, ptap4, regi4);

        code = [goldseq(m1, m2, user)];
        code1 = [goldseq(m3, m4, user)];
case 3
         m1 = msec(stage, ptap1, regi1);
        m2 = msec(stage, ptap2, regi2);
        m3 = msec(stage, ptap3, regi3);
        m4 = msec(stage, ptap4, regi4);
        code = [goldseq(m1, m2, user),zeros(user, 1)];
```

```
        code1 = [goldseq(m3, m4, user),zeros(user, 1)];
end
code=code*2-1;
code1=code1*2-1;
    %figure;%stem(0:6,code);
  %figure;%stem(0:6,code1);

  figure;stem(0:14,code);
  figure;stem(0:14,code1);
R1=autocorr(code);
R2=autocorr(code1);
R3=crosscorr(code,code1(1,:));
 figure;
 plot(0:14,R1);
 figure;
 plot(0:14,R2);
 figure;
 plot(0:14,R3);

 %  figure;   %plot(0:6,R1);
% figure;  % plot(0:6,R2);
 %%figure;   %plot(0:6,R3);
   %fprintf('%d/n',R1);
 %end


% mseq.m
% The generation function of M-sequence
%*******************************************************
% stg         : Number of stages
% taps        : Position of register feedback
% inidata     : Initial sequence
% n           : Number of output sequence
% mout        : output M-sequence
 function [mout]=msec(stg,taps,inidata,n)
% *******************************************************
if nargin <4
            n=1;
```

```
end
 mout=zeros (n, 2^stg-1);
fpos=zeros(stg,1);
 fpos(taps) =1 ;

for ii=1:2^stg-1
        mout(1, ii)= inidata(stg); %storage of the output data
        num=mod(inidata*fpos,2); % calculation of the feedback data
         inidata (2:stg)= inidata (1: stg-1);   % one shifts the register
         inidata (1) = num;                          % return feedback data
end

if n>1
       for ii=2:n
            mout(ii, : )=shift(mout(ii-1, :),1,0);
       end
end
   % *****************End of file*****************************
```

```
% goldsec.m
  function [gout]=goldseq(m1,m2,n)
    % ****************** *****************************
% m1: M-sequence1
% m2: M-sequence2
% n: number of output sequences
% gout: output Gold sequence

   % ****************** *****************************
if nargin < 3
       n=1;
end
gout=zeros(n, length(m1));
       for ii=1:n
                  gout(ii,:)=xor(m1,m2);
                       m2=  shift(m2, 1, 0);
       end
```

```
   % ****************End of file*****************************

% shift.m
function [outregi]=shift(inregi, shiftr, shiftu)
   % ********************************************************
%inregi         :Vector or matrix
%shiftr         : The account of shift to the right
%shiftu          : The account of shift to the top
%outregi        : Register output
   % ********************************************************

[h,v]=size(inregi);
outregi=inregi;

shiftr=rem(shiftr,v);
shiftu=rem(shiftu,h);

if shiftr>0
      outregi (:, 1:shiftr)=inregi(: , v-shiftr+1: v);
      outregi (:, 1+shiftr:v)=inregi(:, 1: v-shiftr);
elseif shiftr<0
    outregi (:, 1: v+shiftr)=inregi(: , 1-shiftr : v);
    outregi (:,v+shiftr+1 :v)=inregi(: , 1 :-shiftr);
end
inregi=outregi;

if shiftu>0
   outregi (1:h-shiftu, :)=inregi(1+shiftu : h, : );
   outregi (h -shiftu +1 : h, :)= inregi(1:shiftu, :);
elseif shiftu<0
  outregi (1:-shiftu, :)=inregi(h+shiftu+1 : h, : );
  outregi (1-shiftu : h, :)=inregi (1 : h+shiftu, : );
end

   % ************End of file*****************************

% autucorr.m
```

```
function [out] = autocorr (indata, tn)
%*********************************
 % indata            : input sequence
% tn                 : number of period
% out                : autocorrelation data
%************************************************
 if nargin<2
            tn=1;
end
 ln=length(indata);
out=zeros(1, ln*tn);

for ii=0:ln*tn-1
            out (ii+1) = sum(indata.*shift(indata, ii ,0));
end
    % *****************End of file*****************************
```

```
%crosscor.m
% Crosscorrelation function of sequence
function [out] = crosscorr (indata1, indata2, tn)
%*************************************************************
***
% indata1           : input sequence 1
% indata2           : input sequence 2
% tn                 : number of period
% out                : crosscorrelation data
%*************************************************************
***
if nargin<3
            tn=1;
end

ln=length(indata1);
out=zeros(1, ln*tn);
for ii=0:ln*tn-1
```

```
out (ii+1) = sum(indata1.*shift(indata2, ii ,0));
end

   % *****************End of file*****************************
```

ДОДАТОК Б

```
%dscdma.m
% Simulation program to realize DS-CDMA system
% ************************Preparation part*****************
sr=256000.0;        % Symbol rate
ml=2;           % ml : Number of modulation levels
br=sr.*ml;      % bit rate
nd=1000;        % Number of symbols
ebn0=10;        % Eb/N0
IPOINT=8;       % Number of oversamples
 %*******************Filter initialization***********************
irfn=21;    % Number of taps
alfs=0.5;   % Rolloff factor
[xh]=hrollfcoef(irfn,IPOINT,sr,alfs,1);  % Tranamitter filter coefficients
[xh2]=hrollfcoef(irfn,IPOINT,sr,alfs,0); % Receiver filter coefficients
 %****************Spreading code  initialization******************
user = 1;             % number of users
seq =1;               % 1: M-sequence    2: Gold   3:Orthigonal Gold
stage=3;             % number of stages
ptap1 =[1,3];       %position of taps for 1st
ptap2 =[2,3];       %position of taps for 2nd
regi1 =[1,1,1];    % initial value of register for 1st
 regi2=[1,1,1];
  %****************Generation of the spreading code****************
switch seq
case 1
        code = msec(stage, ptap1, regi1,user);
case 2
        m1 = msec(stage, ptap1, regi1);
        m2 = msec(stage, ptap2, regi2);
          code = [goldseq(m1, m2, user)];
case 3
        m1 = msec(stage, ptap1, regi1);
        m2 = msec(stage, ptap2, regi2);
        code = [goldseq(m1, m2, user),zeros(user, 1)];
```

```
end
code=code*2-1;
clen=length(code);
fprintf('%d',code)
  %******************Fading initialization**********************
 rfade=0;    % Rayleigh fading   0: nothing   1:  consider
itau=[0, 8];
dlvll=[0.0,40.0];   % attenuation level
n0=[6,7]; % number of waves togenerate fading
th1=[0.0,0.0]; % initial phase of delayed waves
itnd1=[3001, 4004];   %set fading counter
nowl=2;   % number of direct waves+ number of delayed waves
 tstp=1/sr/IPOINT/clen;   % time resolution
 fd=160; % maximum Doppler frequency
 flat=1; % flat fading
 itnde1=nd*IPOINT*clen*30;  % number of fading counter to skip
  %******************Start calculations**********************
nloop=200;  % Number of simulation loops
noe=0;     % Number of  error data
nod=0;     %Number of transmitted data
 for  iii=1:nloop
 %******************Transmitter*****************************
%******************Data generation*************************
 data1=rand(user,nd*ml)>0.5;    % rand: built in function
  %******************QPSK Modulation*************************
 [ich,qch]=qpskmod(data1,user,nd,ml);
%******************Spreading*******************************
 [ich1,qch1]=spread(ich,qch,code);   % spreading
 %******************Oversampling****************************
 [ich2,qch2]=compoversamp2(ich1,qch1, IPOINT);
 %******************Convolution*****************************
 [ich3,qch3]=compconv2(ich2,qch2,xh); % filter
                    if user==1
                        ich4=ich3;
                        qch4=qch3;
                    else
                        ich4=sum(ich3);
```

```
                qch4=sum(qch3);
            end

%******************Fading channel*********************
             if rfade==0
                ich5=ich4;
                qch5=qch4;
            else
[ich5,qch5]=sefade (ich4,qch4, itau, dlvll,th1,n0,itnd1,…
nowl, length(ich4), tstp, fd, flat);
                itnd1=itnd1+itnde1;
            end
  %******************Receiver*****************************
spow=sum(rot90(ich3.^2+qch3.^2))/(nd);    % sum: built in function
attn=0.5*spow*sr/br*10.^(-ebn0/10);
attn=sqrt(attn);    % sqrt: built in function
%***************Add White GaussianNoise (AWGN)**************
 [ich6,qch6]=comb2(ich5,qch5,attn); % add white Gaussian noise
[ich7,qch7]=compconv2(ich6,qch6,xh2);
sampl=irfn*IPOINT+1;
ich8=ich7(:, sampl:IPOINT:IPOINT*nd*clen+sampl-1);
qch8=qch7(:, sampl:IPOINT:IPOINT*nd*clen+sampl-1);
%***************Despreading***********************************
 [ich9,qch9]=despread(ich8, qch8,code);
  %******************QPSK Demodulation************************
 [demodata]=qpskdemod(ich9,qch9,user,nd,ml);
 %******************Bit Error Rate (BER)************************
 noe2=sum(sum(abs(data1-demodata)));    % sum: built in function
nod2=user*nd*ml; %
noe=noe+noe2;
nod=nod+nod2;
fprintf('%d\t%e\n',iii,noe2/nod2);    % fprint: built in function
end     % for iii=1:nloop
 %******************Output  result ************************
 ber=noe/nod;
fprintf('%d\t%d\t%d\t%e\n',ebn0,noe,nod,noe/nod);
fid=fopen('BERqpsk,dat','a');
```

```
fprintf(fid,'%d\t%e\t%f\t%f\t\n',ebn0,noe/nod,noe,nod);
fclose(fid);
%******************End **********************
 figure;   stem(0:20,data1(1:21),'DisplayName','data1');
 figure;   stem(0:10,ich(1:11),'DisplayName','ich');
figure;   stem(0:10,qch(1:11),'DisplayName','qch');
figure;   stem(0:6,code(1:7),'DisplayName','code');
 figure;   stem(0:13,ich1(1:14),'DisplayName','ich1');
 figure;   stem(0:13,qch1(1:14),'DisplayName','qch1');
figure;   stem(0:70,ich1(1:71),'DisplayName','ich1');
figure;   stem(0:70,qch1(1:71),'DisplayName','qch1');
  figure;   stem(0:56,ich2(1:57),'DisplayName','ich2');
figure;   stem(0:56,qch2(1:57),'DisplayName','qch2');
 figure;   stem(0:560,ich2(1:561),'DisplayName','ich2');
figure;   stem(0:560,qch2(1:561),'DisplayName','qch2');
figure;   stem(0:280,ich3(1:281),'DisplayName','ich3');
figure;   stem(0:280,qch3(1:281),'DisplayName','qch3');
figure;   plot(0:280,ich3(1:281),'DisplayName','ich3');
figure;   plot(0:280,qch3(1:281),'DisplayName','qch3');
figure;   plot(0:280,ich4(1:281),'DisplayName','ich4');
figure;   plot(0:280,qch4(1:281),'DisplayName','qch4');
 figure;   plot(0:280,ich6(1:281),'DisplayName','ich6');
 figure;   plot(0:280,qch6(1:281),'DisplayName','qch6');
figure;   stem(0:280,ich6(1:281),'DisplayName','ich6');
 figure;   stem(0:280,qch6(1:281),'DisplayName','qch6');
 figure;   stem(0:560,ich7(1:561),'DisplayName','ich7');
figure;   stem(0:560,qch7(1:561),'DisplayName','qch7');
figure;   stem(0:70,ich8(1:71),'DisplayName','ich8');
figure;   stem(0:70,qch8(1:71),'DisplayName','qch8');
figure;   stem(0:10,ich9(1:11),'DisplayName','ich9');
 figure;   stem(0:10,qch9(1:11),'DisplayName','qch9');
 figure;   plot(ich9,qch9,'DisplayName','phaser');
%******************End of file ***************************


%  spread.m
function [iout, qout]=spread(idata, qdata, code1)
```

```
%*********************************************************
% idata       : ich data sequence
% qdata       :qch data sequence
% iout        : ich output data sequence
% qout        : qch output data sequence
% code1         : spread code sequence
%*********************************************************
switch nargin
   case{0,1}
error ('lack of input argument');
case 2
       code1=qdata;
       qdata=idata;
end
[hn,vn] =size(idata);
[hc,vc] =size(code1);
if hn>hc
          error ('lack of spread code sequences');
end
iout=zeros(hn, vn*vc);
qout=zeros(hn, vn*vc);
for ii=1:hn
            iout(ii,:)=reshape(rot90(code1(ii,:),3)*idata(ii,:),1,vn*vc);
            qout(ii,:)=reshape(rot90(code1(ii,:),3)*qdata(ii,:),1,vn*vc);
end
%*********************end of file*****************************

%  despread.m
% Data despread function
function [iout, qout]=despread(idata, qdata, code1)
%*********************************************************
% idata       : ich data sequence
% qdata       :qch data sequence
% iout        : ich output data sequence
% qout        : qch output data sequence
% code1         : spread code sequence
%*********************************************************
```

```
switch nargin
case {0,1}
error ('lack of input argument');
case 2
        code1=qdata;
        qdata=idata;
end
[hn,vn] =size(idata);
[hc,vc] =size(code1);
 vn=fix(vn/vc);
iout=zeros(hc, vn);
qout=zeros(hc, vn);
 for ii=1:hc
              iout(ii,:)= rot90(flipud(rot90(reshape( idata...
    (ii,:),vc,vn))) *rot90(code1(ii,:),3));
              qout(ii,:)= rot90(flipud(rot90(reshape( qdata...
    (ii,:),vc,vn))) *rot90(code1(ii,:),3));
iout(ii,:)=iout(ii,:)./7;
qout(ii,:)=qout(ii,:)./7;
end
%*********************end of file*****************************


%ber_cdma.m
% ********************Preparation part******************
sr=256000.0;        % Symbol rate
ml=2;           % ml : Number of modulation levels
br=sr.*ml;      % bit rate
nd=1000;        % Number of symbols
ebn0=10;        % Eb/N0
IPOINT=8;       % Number of oversamples
 %********************Filter initialization************************
irfn=21;    % Number of taps
alfs=0.5;   % Rolloff factor
[xh]=hrollfcoef(irfn,IPOINT,sr,alfs,1); Tranamitter filter coefficients
[xh2]=hrollfcoef(irfn,IPOINT,sr,alfs,0);  % Receiver filter coefficients
 %****************Spreading code  initialization*******************
```

```
user = 1;            % number of users
seq =1;              % 1: M-sequence    2: Gold   3:Orthigonal Gold
stage=3;            % number of stages
ptap1 =[1,3];       %position of taps for 1st
ptap2 =[2,3];       %position of taps for 2nd
regi1 =[1,1,1];    % initial value of register for 1st
regi2=[1,1,1];
 %**************Generation of the spreading code*****************
switch seq
case 1
        code = msec(stage, ptap1, regi1,user);
case 2
        m1 = msec(stage, ptap1, regi1);
          m2 = msec(stage, ptap2, regi2);
          code = [goldseq(m1, m2, user)];
case 3
         m1 = msec(stage, ptap1, regi1);
        m2 = msec(stage, ptap2, regi2);
        code = [goldseq(m1, m2, user),zeros(user, 1)];
end
code=code*2-1;
clen=length(code);
fprintf('%d',code)
   %******************Fading initialization**********************
 rfade=0;     % Rayleigh fading    0: nothing   1:  consider
itau=[0, 8];
dlvll=[0.0,40.0];    % attenuation level
n0=[6,7]; % number of waves togenerate fading
th1=[0.0,0.0]; % initial phase of delayed waves
itnd1=[3001, 4004];   %set fading counter
nowl=2;    % number of direct waves+ number of delayed waves
 tstp=1/sr/IPOINT/clen;   % time resolution
 fd=160; % maximum Doppler frequency
 flat=1; % flat fading
    itnde1=nd*IPOINT*clen*30;  % number of fading counter to skip
  %******************Start calculations**********************
 nloop=200;  % Number of simulation loops
```

```
noe=0;      % Number of  error data
nod=0;      %Number of transmitted data
  ebn0_array=0:1:20;
ber_array=zeros(length(ebn0_array),1);
ber_array_theory=zeros(length(ebn0_array),1);
ber_array_theory1=zeros(length(ebn0_array),1);
    ebn0=ebn0_array(ebn0_loop);
            for  iii=1:nloop
%******************Data generation***********************
 data1=rand(user,nd*ml)>0.5;    % rand: built in function
  %******************QPSK Modulation**********************
 ich,qch]=qpskmod(data1,user,nd,ml);
%******************Spreading**********************
 [ich1,qch1]=spread(ich,qch,code);   % spreading
 %******************Oversampling**********************
 [ich2,qch2]=compoversamp2(ich1,qch1, IPOINT);
 %******************Convolution**********************
 [ich3,qch3]=compconv2(ich2,qch2,xh); % filter
                    if user==1  %transmission
                        ich4=ich3;
                        qch4=qch3;
                    else
                        ich4=sum(ich3);
                        qch4=sum(qch3);
                    end
%******************Fading channel**********************
                    if rfade==0  %transmission
                        ich5=ich4;
                        qch5=qch4;
                    else
[ich5,qch5]=...
sefade (ich4,qch4, itau, dlvll,th1,n0,itnd1,...
nowl, length(ich4), tstp, fd, flat);
                        itnd1=itnd1+itnde1;
                    end
  %******************Receiver**********************
spow=sum(rot90(ich3.^2+qch3.^2))/(nd);    % sum: built in function
```

```
attn=0.5*spow*sr/br*10.^(-ebn0/10);
attn=sqrt(attn);    % sqrt: built in function
 %***************Add White GaussianNoise (AWGN)*************
 [ich6,qch6]=comb2(ich5,qch5,attn);     % add white Gaussian noise
[ich7,qch7]=compconv2(ich6,qch6,xh2);
 sampl=irfn*IPOINT+1;
ich8=ich7(:, sampl:IPOINT:IPOINT*nd*clen+sampl-1);
qch8=qch7(:, sampl:IPOINT:IPOINT*nd*clen+sampl-1);
 ***************Despreading**********************************
 [ich9,qch9]=despread(ich8, qch8,code);
  %******************QPSK Demodulation************************
 [demodata]=qpskdemod(ich9,qch9,user,nd,ml);
 %******************Bit Error Rate (BER)************************
 noe2=sum(sum(abs(data1-demodata)));    % sum: built in function
nod2=user*nd*ml; %
noe=noe+noe2;
nod=nod+nod2;
fprintf('%d\t%e\n',iii,noe2/nod2);    % fprint: built in function
 end     % for iii=1:nloop
 %******************Output  result ************************
 ber=noe/nod;
noe=0;
   nod=0;
   ber_array(ebn0_loop)=ber;
ber_array_theory(ebn0_loop)=0.5*erfc(sqrt(10^(ebn0_array(ebn0_loop)/10)));
ber_array_theory1(ebn0_loop)=…
0.5*(1-1./(sqrt(1+1/(10^(ebn0_array(ebn0_loop)/10)))));
     end
fprintf('%d\t%d\t%d\t%e\n',ebn0,noe,nod,noe/nod);
fid=fopen('BERqpsk,dat','a');
fprintf(fid,'%d\t%e\t%f\t%f\t\n',ebn0,noe/nod,noe,nod);
fclose(fid);
    figure;
h=semilogy(ebn0_array,ber_array,'o',ebn0_array,ber_array_theory,'*',ebn0
_array,ber_array_theory1,'-');
h=semilogy(ebn0_array,ber_array,'o',ebn0_array,ber_array_theory,'*',ebn0
_array,ber_array_theory1,'-');
```

```
set(h,{'DisplayName', 'Display name','Display
name'},{'Experiment','Theory','Theory1'})
legend show

%******************End of file ***********************
```

ДОДАТОК В

```
% qpskmod.m
   % Function to perform QPSK modulation
function [iout,qout]=qpskmod(paradata, para, nd, ml)
%******************Variables ************************
   % paradata : input data (para by nd matrix)
   % iout : output Ich data
   % qout : output Qch data
   % para : Number of parallel channels
   % nd : Number of data
   % ml : number of modulation levels
   % ( QPSK -2, 16QAM -4)
   %****************** ************************
m2=ml./2;
paradata2=paradata.*2-1;
count2=0;

for jj=1:nd
    isi=zeros(para,1);
   isq=zeros(para,1);
 for ii=1:m2
       isi=isi + 2.^(m2-ii).*paradata2((1:para), ii+count2);
       isq=isq + 2.^(m2-ii).*paradata2((1:para), m2+ii+count2);
 end
 iout((1:para),jj)=isi;
qout((1:para),jj)=isq;
 count2=count2+ml;
end
 %******************End of file ************************


   % qpskdemod.m
   % Function to perform QPSK demodulation
function [demodata]=qpskdemod(idata, qdata, para, nd, ml)
%******************Variables ************************
```

```
   % idata : input Ich data
   % qdata : input Qch data
   % demodata : demodulated data (para by nd matrix)
   % para : Number of parallel channels
   % nd : Number of data
   % ml : number of modulation levels
   % ( QPSK -2, 16QAM -4)
    demodata=zeros(para,ml*nd);
   demodata((1:para),(1:ml:ml*nd-1))=idata((1:para),(1:nd))>=0;
   demodata((1:para),(2:ml:ml*nd))=qdata((1:para),(1:nd))>=0;
 %******************End of file **********************


% compconv2.m
% function to perform convolution between signal and filter
function [iout, qout]=compconv2(idata, qdata, filter)
%*********************************************************
% idata       : ich data sequence
% qdata       : qch data sequence
% iout       : ich output data sequence
% qout       : qch output data sequence
%*********************************************************
iout=conv2(idata,filter);
qout=conv2(qdata,filter);
%********************end of file*****************************


% compoversamp2.m
% function to sample “sample” time
function [iout, qout]=compoversamp2(iin, qin, sample)
%*********************************************************
% iin           : input ich sequence
% qdata       : input qch sequence
% iout          : ich output data sequence
% qout         : qch output data sequence
%sample      : number of oversamples
%*********************************************************
```

```
[h,v]=size(iin);
iout=zeros(h,v*sample);
     qout=zeros(h,v*sample);
 iout(:, 1:sample:1+sample*(v-1)) =iin;
 qout(:, 1:sample:1+sample*(v-1)) =qin;
%**********************end of file*****************************

% comb2.m
% function to add white Gaussian noise
function [iout, qout]=comb2(idata, qdata, attn)
%*********************************************************
% idata       : input ich data
% qdata       : input qch data
% iout         : output ich data
% qout         : output qch data
% attn         : attenuation level
%*********************************************************
v=length(idata);
h=length(attn);
iout=zeros(h,v);
qout=zeros(h,v);
for ii=1:h
            iout(ii,:)=idata+randn(1,v)*attn(ii);
            qout(ii,:)=qdata+randn(1,v)*attn(ii);
end


 % hrollfcoef.m
 % Generate coefficience of Nyquist filter
   function[xh]= hrollfcoef(irfn,ipoint,sr,alfs,ncc)
   % alfs: rolloff coefficients
   %*******************
xi=zeros(1,irfn*ipoint+1);
xq=zeros(1,irfn*ipoint+1);
point=ipoint;
tr=sr;
tstp=1.0./tr./ipoint;
```

```
n=ipoint.*irfn;
mid=(n./2)+1;
sub1=4.0 .*alfs .*tr;   % 4*alfs*R s
     for i=1:n
 icon=i-mid;
ym=icon;

       if icon==0.0
   xt=(1.0-alfs+4.0.*alfs/pi).*tr;     % h(0)
       else
   sub2=16.0 .*alfs.*alfs.*ym.*ym./ipoint./ipoint;
           if sub2~=1.0
      x1=sin(pi*(1.0-alfs)/ipoint*ym)./pi./(1.0-sub2)./ym./tstp;
      x2=cos(pi*(1.0+alfs)/ipoint*ym)./pi.*sub1./(1.0-sub2);
      xt=x1+x2;       % h(t)
         else
   xt=alfs.*tr.*((1.0-2.0/pi).*cos...
(pi/4.0/alfs)+ (1.0+2.0./pi).*sin...
(pi/4.0/alfs))./sqrt(2.0);
    end % if sub2 -=1.0
   end     % if icon==0
 if ncc==0       % in case of receiver
   xh(i)=xt./ipoint./tr;   % normalization
else if ncc==1  % in the case of transmitter
   xh(i)=xt./tr;   % normalization
else
   error ('ncc error');
end     % if ncc==0
end % for i=1:n
end
 %******************End of file **********************



% rollfcoef
function [xh]=hrollfcoef(irfn,ipoint,sr,alfs,ncc)
```

```
%irfn:  Number of symbols to use filtering
%ipoint:Number of samples in one symbol
% sr : symbol rate
% alfs: rolloff coefficients
% 1 - transmission filter 0 - receiving filter
 xi=zeros(1,irfn*ipoint+1);
xq=zeros(1,irfn*ipoint+1);
 point=ipoint;
tr=sr;
tstp=1.0./tr./ipoint;
n=ipoint .*irfn;
mid=(n./2)+1;
sub1=4.0*alfs.*tr; % 4* alpha*R_s
 for i=1:n
    icon=i-mid;
    ym =icon;
    if icon==0.0
       xt=(1.0-alfs+4.0.*alfs/pi).*tr;  %  h(0)
    else
       sub2=16.0.*alfs.*alfs.*ym.*ym./ipoint./ipoint;
       if sub2~=1.0
          x1=sin(pi*(1.0-alfs)/ipoint*ym)./pi./(1.0-sub2)./ym./tstp;
          x2=cos(pi*(1.0+alfs)/ipoint*ym)./pi.*sub1./(1.0-sub2);
          xt=x1+x2;
       else
          xt=alfs.*tr.*((1.0-2.0/pi).*cos...
             (pi/4.0/alfs)+ (1.0+2.0./pi).*sin...
             (pi/4.0/alfs))./sqrt(2.0);
       end % if sub2~=1.0
    end     % if icon==0
       if ncc==0         % in case of receiver
       xh(i)=xt ./ipoint ./tr;    % normalization
    else if ncc==1    % in the case of transmitter
            xh(i)=xt ./tr;         % normalization
       else
          error ('ncc error');
       end     % if ncc==0
```

```
    end % for i=1:n
end
    % *****************End of file*****************************
```