

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп'ютерних систем

«На правах рукопису»

УДК 519.718

До захисту допущено:

Завідувач кафедри

_____ Віталій РОМАНКЕВИЧ

«__» _____ 2023 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою

«Системне програмування і спеціалізовані комп'ютерні системи»

зі спеціальності 123 «Комп'ютерна інженерія»

**на тему: «Нероздільні блочні завадостійкі коди для виправлення двох помилок в
комп'ютерних системах»**

Виконав:

студент II курсу, групи КВ-11мн

Федоров Костянтин Сергійович _____

Науковий керівник:

Доцент кафедри СПіСКС, канд. техн. наук,

Тесленко Олександр Кирилович _____

Засвідчую, що у цій магістерській дисертації
немає запозичень з праць інших авторів без
відповідних посилань.

Студент _____

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – другий (магістерський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-наукова програма «Системне програмування і спеціалізовані комп'ютерні системи»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Віталій РОМАНКЕВИЧ

«___» _____ 2023 р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Федорову Костянтину Сергійовичу

1. Те
ма дисертації «Нероздільні блочні завадостійкі коди для виправлення двох помилок в комп'ютерних системах», науковий керівник дисертації доцент кафедри СПСКС, к.т.н., Тесленко Олександр Кирилович, затверджені наказом по університету МД № 1359-С від 30.03.2023 р.

2. Те
рмін подання студентом дисертації 18.05.2023.

3. Об
'єкт дослідження: нероздільні блочні завадостійкі коди для виправлення помилок і способи їх пошуку.

4. Пр
едмет дослідження: програмне забезпечення для визначення всіх 8-розрядних кодів а також визначені в результаті його використання коди, які дозволяють виправляти до двох помилок.

5. Пе
релік завдань, які потрібно розробити:

- аналіз існуючих теоретичних матеріалів і дослідів нероздільних кодів,
 - розробка програмного забезпечення для визначення всіх блочних завадостійких кодів із визначеними параметрами, оптимізація алгоритму Брона-Кербоша під конкретне завдання,
 - аналіз отриманих результатів і пошук можливого їх застосування.
6. Орґанізаційний перелік графічного (ілюстративного) матеріалу - презентація.
7. Орґанізаційний перелік публікацій - 2 тез.
8. Да
- та видачі завдання 22.02.2023.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Затвердження теми	06.02.2023	
2	Збір та дослідження літератури	22.02.2023	
3	Аналіз існуючих рішень	10.03.2023	
4	Зміст та вступ	16.03.2023	
5	Розробка програмної моделі рішення	20.03.2023	
6	Реферат	22.03.2023	
7	Перший розділ	06.04.2023	
8	Другий розділ	16.04.2023	
9	Третій розділ	25.04.2023	
10	Попередній захист магістерської дисертації	04.05.2023	

Студент

Костянтин ФЕДОРОВ

Науковий керівник

Олександр ТЕСЛЕНКО

РЕФЕРАТ

Актуальність теми. Завадостійких кодів для виправлення однієї, двох та більше помилок існує багато, але вони всі роздільні – до інформаційних символів додаються додаткові, які створюються і використовуються з метою ідентифікації та виправлення помилок. В нероздільних кодах такого розподілу до інформаційні і додаткові не існує. Нероздільні блочні завадостійкі коди - це спосіб кодування, який не потребує використання додаткових символів для виявлення та виправлення помилок. Ці коди працюють з усіма символами в блоку даних і застосовують складні математичні алгоритми для забезпечення високої надійності передачі.

Нероздільні блочні завадостійкі коди можуть виправляти одну, дві або більше помилок, залежно від алгоритму кодування. Однією з основних переваг таких кодів є ефективне використання ресурсів, наприклад, пропускної здатності каналу передачі даних, порівняно з роздільними кодами.

Однак, розробка нероздільних блочних завадостійких кодів може бути складнішою, оскільки вона вимагає більш високого рівня математичної та алгоритмічної експертизи.

Об'єктом дослідження є аналіз і порівняння блочних роздільних та нероздільних 8-розрядних кодів для виправлення 2-х помилок.

Предметом дослідження є програмне забезпечення для визначення всіх 8-розрядних кодів для виправлення двох помилок

Мета роботи: визначення та пошук варіантів застосування 8-розрядних нероздільних кодів, де відстань Хемінга між будь якими кодословами не менша за 5.

Наукова новизна полягає в визначенні всіх нероздільних блочних кодів із заданими параметрами, пошуку варіантів застосування, в також оптимізації програми пошуку кодів.

Практична цінність отриманих в роботі результатів полягає в

забезпечені більшої швидкості передачі в порівнянні з роздільними кодами та оптимізована програма реалізації алгоритму їх будовання.

Апробація роботи. Основні положення і результати роботи були представлені та обговорювались на міжнародній науково-практичній конференції “Наука, освіта, технології і суспільство в XXI столітті: наукові ідеї та механізми реалізації” (Житомир, квітень 2023) та на міжнародній науково-практичній конференції “Теоретико-практичні аспекти розвитку науки, освіти та суспільства”, (Рівне, квітень 2023).

Структура та обсяг роботи. Магістерська дисертація складається з вступу, трьох розділів та висновків.

У вступі подано загальну характеристику роботи, зроблено оцінку сучасного стану проблеми, обґрунтовано актуальність наряду досліджень, сформульовано мету і задачі досліджень, показано наукову новизну отриманих результатів і практичну цінність роботи, наведено відомості про апробацію результатів і їхнє впровадження.

У першому розділі розглянуті існуючі способи отримання роздільних та нероздільних кодів, зроблена порівняльна характеристика.

У другому розділі описано принципи роботи нероздільних блочних кодів, розглянуті основні алгоритми їх отримання.

У третьому розділі наведено структуру та опис роботи програмного забезпечення, а також проведено тестування та проаналізовано результати виконаного дослідження.

У висновках представлені результати проведеної роботи.

Робота представлена на 75 аркушах, містить посилання на список використаних літературних джерел.

Ключові слова: нероздільні коди, блочні, булева функція, кодер, декодер завадостійкість, виправлення, алгоритм.

ABSTRACT

Actuality of theme. There are many tamper-proof codes for correcting one, two or more errors, but they are all separate - additional information symbols are added, which are created and used for the purpose of identifying and correcting errors. In non-separable codes of such distribution, there are no informational and additional yeses. Non-separable block-proof codes are a method of coding that does not require the use of additional characters to detect and correct errors. These codes work with all symbols in the data block and apply complex mathematical algorithms to ensure high transmission reliability.

Non-separable block-proof codes can correct one, two, or more errors, depending on the coding algorithm. One of the main advantages of such codes is the efficient use of resources, for example, the bandwidth of the data transmission channel, compared to discrete codes.

However, the development of non-separable block-proof codes can be more difficult because it requires a higher level of mathematical and algorithmic expertise.

The object of research is the analysis and comparison of block-separated and 8-bit codes for correcting 2 errors.

The subject of the study is software for determining all 8-bit codes.

The goal of the work: to determine and search for options for the use of 8-bit codes, where the Hamming distance between any code words is not less than 5.

The scientific novelty consists in the determination of all non-separable block codes with given parameters, the search for application options, as well as the optimization of the code search program.

Practical value of the results obtained in the work is provided with a higher transmission speed compared to separate codes and an optimized program for implementing the algorithm for their construction.

Approbation of work. The main provisions and results of the work were presented and discussed at the international scientific-practical conference "Science, education, technology and society in the XXI century: scientific ideas and implementation mechanisms" (Zhytomyr, April 2023) and at the international scientific-practical conference "Theoretical-Practical aspects of the development of science, education and society", (Rivne, April 2023).

Structure and scope of work. The master's thesis consists of an introduction, three chapters and conclusions.

The introduction provides a general description of the work, assesses the current state of the problem, substantiates the relevance of the research direction, formulates the goal and objectives of the research, shows the scientific novelty of the obtained results and the practical value of the work, provides information on the approbation of the results and their implementation.

In the first section, the existing methods of obtaining separable and non-separable codes are considered, a comparative characteristic is made.

In the second chapter, the principles of non-separable block codes are described, and the main algorithms for obtaining them are considered.

The third chapter provides the structure and description of the software, as well as testing and analyzing the results of the research.

The results of the work are presented in the conclusions.

The work is presented on 75 sheets, containing links to the list of used literary sources.

Keywords: non-separable codes, block, Boolean function, encoder, decoder, immunity, correction, algorithm.

ЗМІСТ

ЗМІСТ	1
ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ	3
ВСТУП.....	4
1. СФЕРИ ЗАСТОСУВАННЯ ТА ЗАГАЛЬНИЙ ОГЛЯД ЗАВАДОСТІЙКОГО КОДУВАННЯ	7
1.1. Завадостійке кодування інформації	7
1.2. Виявлення і виправлення помилок в завадостійких кодах	9
1.3. Класифікація і основні характеристики завадостійких кодів	11
ВИСНОВКИ ДО РОЗДІЛУ 1	18
2. БЛОКОВІ ЗАВАДОСТІЙКІ КОРИГУЮЧІ КОДИ.....	20
2.1. Загальна теорія виправлення помилок.....	20
2.2. Відстань Хеммінга	22
2.3. Блочні завадостійкі коди	24
2.4. Межа Хемінга	25
2.5. Огляд конкретних нероздільних кодів.....	28
2.6. Реалізація кодерів і декодерів.....	33
ВИСНОВКИ ДО РОЗДІЛУ 2	38
3. ДОСЛІДЖЕННЯ НЕРОЗДІЛЬНИХ КОДІВ ДЛЯ ВИПРАВЛЕННЯ ДВОХ ПОМИЛОК	40
3.1. Загальний алгоритм визначення кодів.....	40
3.2. Алгоритм Брона-Кербоша.....	41
3.3. Створення кодеру та декодеру для нероздільних кодів.....	50
ВИСНОВКИ ДО РОЗДІЛУ 3	64
ВИСНОВКИ.....	66

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	69
--------------------------------------	----

ДОДАТКИ

Додаток А. Лістинг програми

Додаток Б. Презентація магістерської дисертації

Додаток В. Наукові публікації

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

ПЛІС - це електронний пристрій, призначений для програмованої реалізації цифрових логічних функцій. Це програмована інтегральна схема (ПІС), яка містить логічні блоки, вхідні і вихідні засоби та роутери для з'єднання між ними.

LUT - це елемент в програмованій логічній матриці (PLD), який використовується для зберігання та перетворення бітових значень відповідно до заданої логіки. Кожен LUT містить масив бітів, де кожен може бути налаштований відповідно до бажаної функції.

Код - деяка сукупність двійкових чисел, для позначення будь яких об'єктів чи суб'єктів одного класу.

Кодослово - це послідовність кодованих символів, яка передається через канал передачі даних.

Кодер - це пристрій або програмний засіб, який здійснює перетворення вхідних даних з одного представлення в інше. Зазвичай, кодери використовуються для перетворення аналогових сигналів в цифровий формат або для зменшення розміру даних, що передаються по мережі.

Декодер - це пристрій або програмний засіб, який виконує зворотне перетворення даних, яке раніше було здійснене кодером. Зазвичай, декодери використовуються для відтворення аналогового сигналу з цифрових даних або для розшифрування даних, що були зменшені за допомогою кодера перед передачею.

FPGA - це інтегральна схема спеціального призначення, яка складається з мережі логічних елементів (гейтів), які можуть бути програмовані для виконання різноманітних функцій

Мікросхема - електронний пристрій, що містить інтегровані електронні компоненти, такі як транзистори, резистори, конденсатори та діоди,

ВСТУП

В сучасному світі, де цифрові технології є необхідною складовою нашого життя, швидкість та надійність передачі даних має велике значення. Щоб забезпечити ефективну передачу даних, використовуються різні методи, одним з яких є кодування даних. Кодування даних полягає в додаванні додаткової інформації до переданих даних для забезпечення більш надійної передачі та можливості виявлення та виправлення помилок.

Один з таких методів - це кодування з використанням надмірності. Цей метод базується на введенні додаткової інформації, яка дозволяє не тільки виявляти спотворення переданих даних, але й відновлювати правильні значення. Це робиться за допомогою кодів, які містять додаткові біти, що генеруються на основі інформаційних бітів.

Кодування з використанням надмірності є ефективним методом забезпечення надійної передачі даних, оскільки дозволяє виявляти та виправляти помилки, які можуть виникнути під час передачі даних. Таким чином, кодування з використанням надмірності є важливим інструментом в сучасних цифрових телекомунікаційних технологіях.

Блочні завадостійкі коди є ефективним рішенням для забезпечення надійності передачі інформації при наявності помилок в каналі передачі. Існує багато різних роздільних кодів, які додають додаткові символи для виправлення помилок. Проте, в нероздільних кодах такі символи відсутні, що забезпечує швидку передачу інформації, але ускладнює процес їх побудови. Проте, з'являється проблема вибору кодослів для забезпечення мінімальної відстані між ними, що забезпечує надійність передачі інформації.

Однією з найбільш розвинених галузей теорії та практики кодування є побудова блочних роздільних кодів. В таких кодах існує чіткий розподіл між інформаційними та додатковими бітами, при цьому додаткові біти

створюються на основі інформаційних. Наприклад, коди Хеммінга, Боуза-Чоудхурі-Хоквінгема є прикладами таких кодів. Однією з основних переваг роздільних кодів є можливість аналітичного визначення дозволених комбінацій та порівняно нескладна реалізація кодерів та декодерів. Однак, швидкість таких кодів може бути максимально можливою не на всіх значеннях n , тому важливо проводити дослідження та вибирати оптимальні коди для конкретних застосувань.

Блочні завадостійкі нероздільні коди не мають розподілу розрядів коду на інформативні та додаткові, на відміну від роздільних кодів. Теоретично, нероздільні коди можуть досягти кращих показників швидкості передачі, ніж роздільні коди, що стимулює проведення досліджень у наступних напрямках: визначення складу кодослів у нероздільних кодах, розроблення кодерів та декодерів для нероздільних кодів. Проте на даний момент відсутня загальна теорія побудови завадостійких нероздільних кодів, яка зберігала б переваги по швидкості передачі, як для самого коду, так і для його кодерів та декодерів. Існують окремі класи нероздільних кодів, такі як коди Адамара, проте вони не забезпечують переваг перед роздільними кодами у відношенні швидкості передачі.

Нероздільні коди мають недолік, пов'язаний зі складністю їх створення. Для визначення цих кодів потрібно знайти максимальну кліку в графі, що у загальному випадку є NP-повною задачею.

Запропоновано створити всі можливі 8-розрядні коди для виправлення двох помилок та розробити для них кодери та декодери. На даний момент немає загальної теорії побудови таких кодів, тому необхідно провести додаткові дослідження. Для створення універсальних кодерів та декодерів для всіх можливих кодів пропонується використовувати програмовані логічні інтегральні схеми (ПЛІС) або готові мікросхеми, які можуть бути налаштовані на будь-який з цих кодів. Це дозволить забезпечити високу швидкість та надійність передачі даних.

1. СФЕРИ ЗАСТОСУВАННЯ ТА ЗАГАЛЬНИЙ ОГЛЯД ЗАВАДОСТІЙКОГО КОДУВАННЯ

1.1. Завадостійке кодування інформації

Кодування - це процес заміни послідовності символів інформаційного джерела на послідовність символів коду, що формується кодером. Виділяють два типи кодування: блокове та безперервне.

При передачі інформації по каналах зв'язку можуть виникати помилки через завади та спотворення сигналів. Для виявлення та виправлення таких помилок використовуються завадостійкі (коректуючі) коди. Ці коди дозволяють виявляти та/або виправляти помилки в прийнятому повідомленні, завдяки введенню надмірності в кодоване повідомлення за певними правилами. Надмірні символи, які називаються перевірочними або контрольними, підвищують здатність коду до виявлення та виправлення помилок, але знижують швидкість передачі інформації. При передачі інформації по каналах зв'язку можуть виникати помилки через завади та спотворення сигналів. Для виявлення та виправлення таких помилок використовуються завадостійкі (коректуючі) коди.

Ці коди дозволяють виявляти та/або виправляти помилки в прийнятому повідомленні, завдяки введенню надмірності в кодоване повідомлення за певними правилами (рис 1.1). Надмірні символи, які називаються перевірочними або контрольними, підвищують здатність коду до виявлення та виправлення помилок, але знижують швидкість передачі інформації.

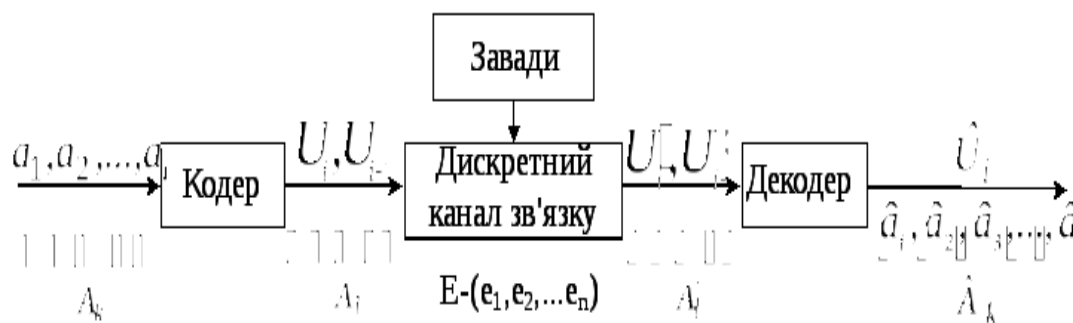


Рисунок 1.1 - Схема кодування інформації

Завадостійкі коди є дуже ефективним засобом забезпечення точності при зберіганні та передачі дискретної інформації. Теорія завадостійкого кодування, яка нині стрімко розвивається, була створена для забезпечення цієї завадостійкості. К. Шеннон сформулював теорему для випадку передачі дискретної інформації з каналу з перешкодами, яка показує, що вибравши відповідний метод кодування сигналів, ймовірність помилкового декодування може бути забезпечена скільки завгодно малою. Завадостійкі коди дозволяють виявляти та/або виправляти помилки, що виникають в результаті перешкод. Це досягається за рахунок введення надмірності в кодові комбінації, тобто тим, що не всі символи в кодових комбінаціях використовуються для передачі інформації.

При блоковому кодуванні, послідовність символів повідомлення розбивається на блоки з k символів, які перетворюються на блоки з n символів (де $n > k$). Блоки з символами повідомлення називаються інформаційними, а послідовність з n символів на виході кодера називається кодовою комбінацією або кодовим словом. Набір кодових слів утворює (n, k) код. Якщо код містить однакову кількість символів у комбінаціях, то він називається рівномірним. Рівномірні коди дозволяють спростити схеми кодерів і декодерів порівняно з нерівномірними.

Якщо інформаційна послідовність не розділяється на блоки, то використовують безперервне кодування, яке також може бути назване

дерезовидним або ланцюговим кодуванням. Блочне кодування може розглядатися як окремий випадок безперервного.

Кодування може бути систематичним, коли відоме розміщення інформаційних і перевірочних символів в кодовій комбінації. Зазвичай перші k символів кодової комбінації вважаються інформаційними символами.

Алфавіт коду складається з безлічі символів, з яких утворена кодова комбінація. Найпоширенішими є двійкові коди з підставою коду, рівною двом.

Завадостійкі коди поділяються на лінійні і нелінійні. Лінійні, рівномірні, роздільні двійкові коди є найпоширенішими. Кодові комбінації цих кодів можуть бути представлені в вигляді векторів в n -мірному лінійному просторі або вигляду багаточленів (поліномів) від формальної змінної x ступеня $n-1$. Операція складання по модулю 2 ($\text{mod } 2$) є основною операцією в таких кодах, де складання і віднімання по $\text{mod } 2$ збігаються.

Завадостійкі коди будуються з використанням алгебраїчних структур, таких як групи, кільця і поля, що визначають правила формування кодових комбінацій та дозволяють виявляти та виправляти помилки в них. Для передачі кодових комбінацій між кодером і декодером використовується дискретний канал зв'язку, який може бути описаний як технічний засіб, який має певну кількість можливих значень на вході і виході. Двійковий канал зв'язку з завадою є найпростішою моделлю дискретного каналу, де завада є адитивним збуренням, яке взаємодіє зі сигналом.

1.2. Виявлення і виправлення помилок в завадостійких кодах

Для виявлення та виправлення помилок в завадостійких кодах, дозволені та заборонені комбінації коду розділяються на дві непересічні множини. Кодер використовує тільки дозволені комбінації для передачі, а

заборонені можуть утворюватися тільки через вплив завади. Декодер визначає приналежність прийнятої комбінації до дозволеного коду. Якщо комбінація належить коду, то вважається, що помилок немає.

Якщо комбінація є забороненою, то вважається, що відбулася помилка. Імовірність виявлення помилок залежить від вибору дозволених кодових комбінацій, а не від властивостей каналу зв'язку. Однак, може виникнути ситуація, коли одна кодова комбінація перейшла в іншу через вплив завади, і в цьому випадку помилки не вдається виявити.

Принцип виправлення помилок використовує розділення всіх можливих комбінацій коду на непересічні підмножини, які називаються областями рішень. Кожна область включає одну з дозволених кодових комбінацій, і якщо прийнята комбінація належить до однієї з областей, то вона приймається як кодова комбінація, що відповідає цій області. Виправлення помилок відбувається в тих випадках, коли кодове слово та прийнята комбінація належать до однієї області рішень.

Кожна область містить такі заборонені комбінації, при прийомі яких найбільш імовірною є одна з дозволених кодових комбінацій, що відповідає принципу максимальної правдоподібності. При передачі рівноімовірних кодових комбінацій, розподіл комбінацій по областях рішень залежить від характеру помилок в каналі зв'язку.

Кодові комбінації вибираються таким чином, щоб виправляти помилки малої кратності. Для однозначного декодування кожна заборонена комбінація повинна входити тільки в одну область рішень (рис. 1.2).

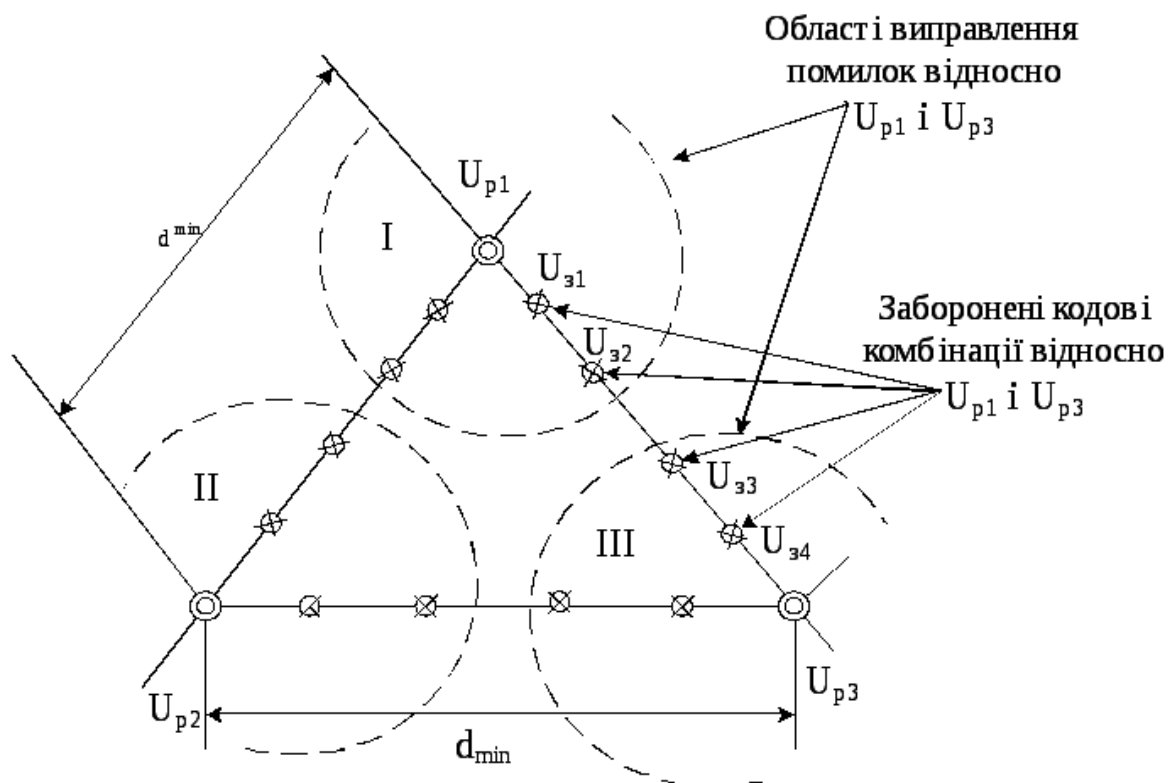


Рисунок 1.2 - Области виправлення помилок

Комбінації коду визначаються відстанню Хеммінга, яка змінюється від 1 до n . Деякі комбінації можуть використовуватися при кодуванні, тоді як інші - заборонені. Мінімальна відстань між кодовими комбінаціями називається мінімальною кодовою відстанню і використовується для оцінки виявляючої і виправляючої здатності кодів [1].

Для виявлення помилок в прийнятій комбінації необхідно, щоб її кратність була менше мінімальної кодової відстані. Прийнята комбінація вважається забороненою, якщо її кратність перевищує мінімальну кодову відстань.

1.3. Класифікація і основні характеристики завадостійких кодів

Існує велика кількість різних кодів, і їх класифікація може бути складною через їх численні ознаки. При цьому, структурні характеристики кодів допомагають в розгляді класифікації (рис. 1.3).



Рисунок 1.3 - Класифікація кодів

Перешкодостійкі коди поділяються на два типи: рівномірні та нерівномірні. Рівномірні коди містять кодові комбінації, в яких кількість розрядів є постійною. Нерівномірні коди містять кодові комбінації з різною кількістю розрядів, але їх складність реалізації призводить до того, що вони не використовуються широко.

Надмірні коди поділяються на два типи: безперервні та блокові. У безперервних кодах процес кодування та декодування відбувається безперервно (рис. 1.4), але вони не мають широкого застосування.

У блокових кодах (рис. 1.5) кожне повідомлення кодується окремо своєю власною кодовою комбінацією (блоком) з n символів. Кожен блок кодується окремо, незалежно від інших блоків.

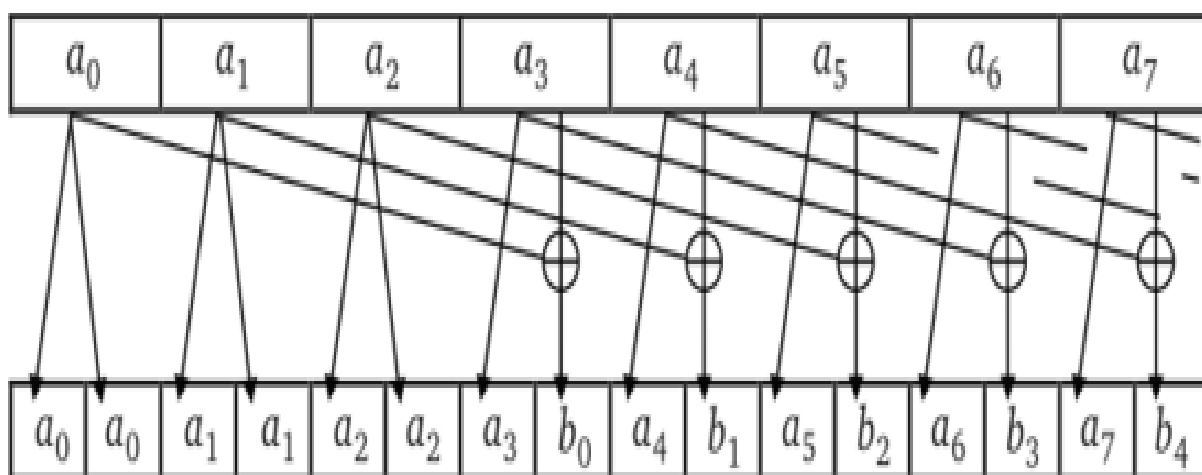


Рисунок 1.4 -Ланцюговий безперервний код

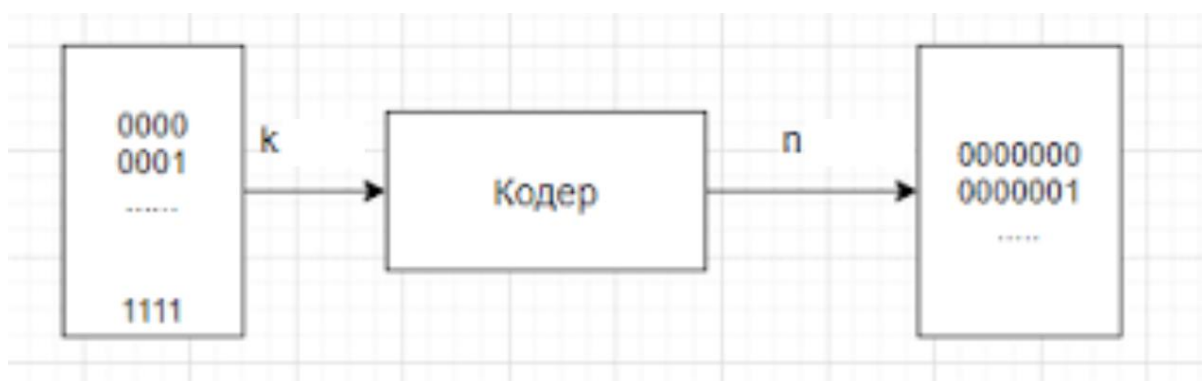


Рисунок 1.5 - Блоковий код

Роздільні блокові коди є надмірними кодами, в яких деякі розряди відводяться для інформаційних та перевірочних символів. Вони можуть бути несистематичними і систематичними. У несистематичних роздільних блокових кодах, перевірочні символи обчислюються як сума підблоків

інформаційних символів довжиною 1. Ці коди можуть виявляти пачки помилок довжиною 1. Прикладом таких кодів є коди Бергера.

Систематичні коди є найбільш поширеними серед роздільних кодів і використовуються в сучасній теорії кодування. Їх основа - лінійна алгебра, векторний аналіз, матричне числення та інші математичні поняття. Ці коди мають перевірочні символи, які можуть бути обчислені шляхом лінійних операцій над інформаційними символами.

Для двійкових кодів, ці операції полягають у виборі кожного перевірконого символу так, щоб його сума з певними інформаційними символами за модулем два була рівна нулю.

Систематичні коди включають в себе багато видів кодів, таких як коди з перевіркою на парність (непарність), коди з повторенням, інверсний (код Бауера), кореляційний, коди Хеммінга, Голя, Ріда-Мюллера, Макдональда, Варшамова [2], коди з малою щільністю перевірок на парність, ітеративний код.

Циклічні коди є одним з видів систематичних кодів, які мають додаткову властивість. Якщо дозволена кодова комбінація належить циклічному коду, то будь-яка інша кодова комбінація, отримана з неї шляхом циклічної перестановки символів, також буде належати цьому коду. Це забезпечує ефективну обробку даних з точки зору операцій з циклічним зсувом і дозволяє ефективно реалізовувати апаратну реалізацію цих кодів.

Циклічні коди є різновидом систематичних кодів, і вони мають особливість - якщо деяка кодова комбінація є допустимою для циклічного коду, то будь-яка інша кодова комбінація, що отримується циклічним зсувом, також буде допустимою. Серед найбільш відомих циклічних кодів можна виділити коди Хеммінга (рис. 1.6), Ріда-Соломона та інші [3].

Однак, в останні часи велика увага приділяється згортаним кодам (рис. 1.7), які належать до класу безперервних кодів. Такі коди забезпечують ефективне виявлення і виправлення помилок в каналах зі завмиранням

сигналу, що є актуальною проблемою для сучасних телекомунікаційних систем.

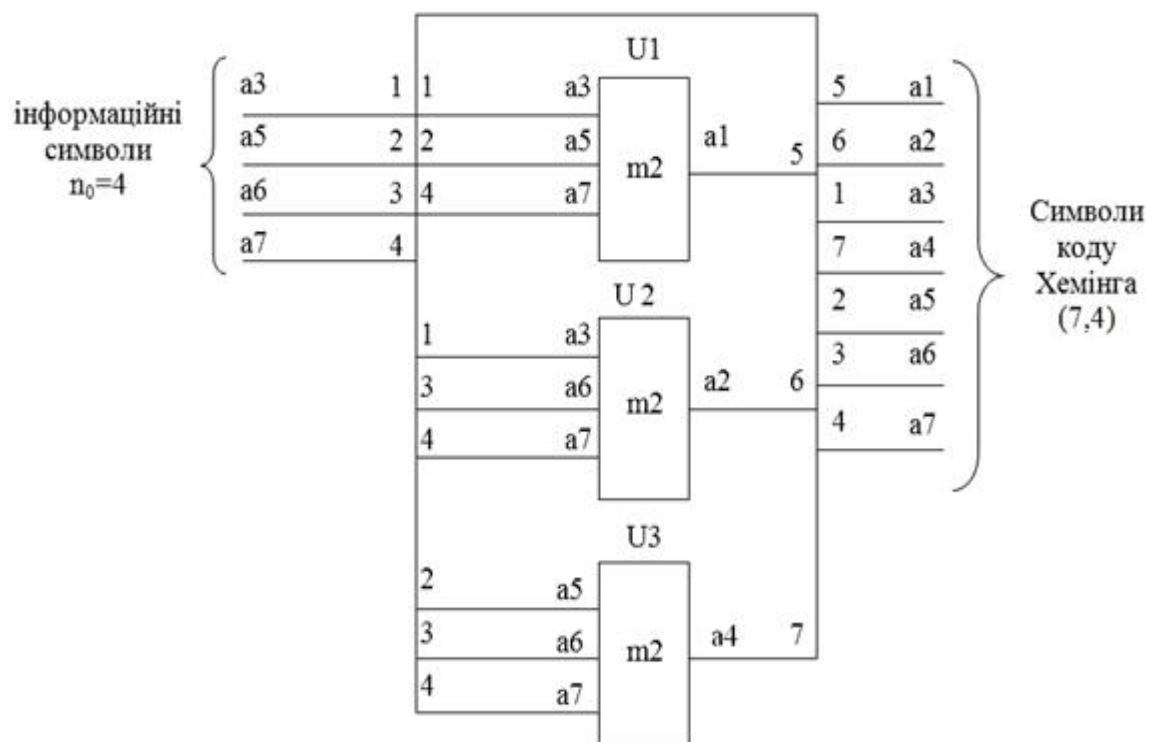


Рисунок 1.7 - Кодер коду Хемінга

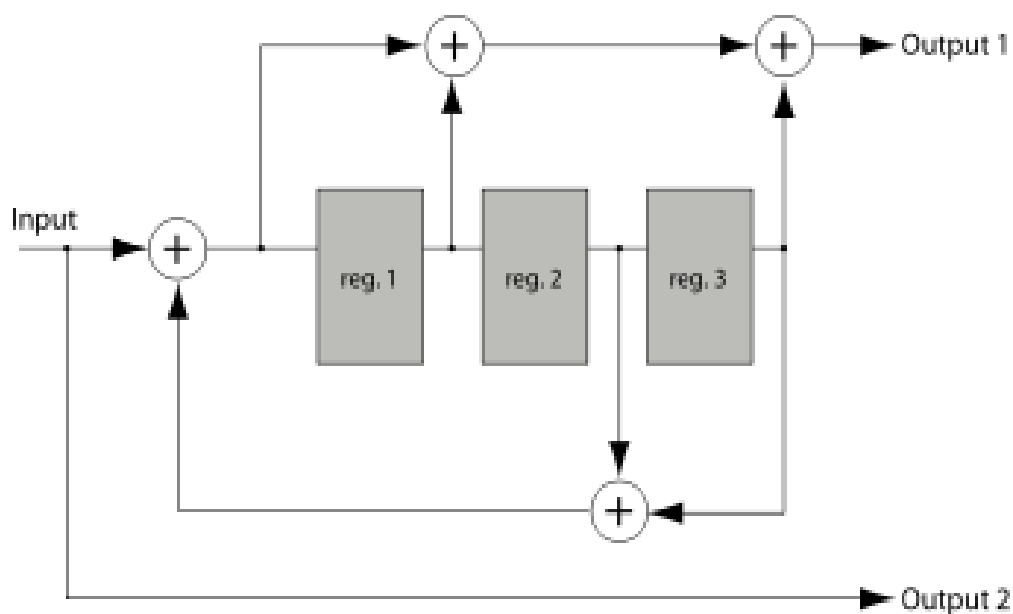


Рисунок 1.8 - Згортний код

Характеристики кодів використовуються для оцінки їх ефективності при передачі, зберіганні і обробці інформації. Ці характеристики включають такі показники, як довжина коду, загальне число можливих кодових комбінацій, число контрольних символів, мінімальна кодова відстань, швидкість передачі коду, вірогідність помилкового декодування і вірогідність не виявленої помилки. Ці показники дозволяють обґрунтувати вибір коду для певної задачі, зокрема враховуючи його потужність, вагу, надмірність і оптимальність.

Довжина коду n визначає кількість символів, що входять до кодової комбінації. Підстава коду m вказує на кількість різних значень, які можуть бути закодовані в цій комбінації. Потужність коду відображає кількість дозволених кодових комбінацій, які можуть бути використані для передачі повідомлень. Загальне число кодових комбінацій N показує кількість всіх можливих комбінацій для даної довжини і підстави коду.

У теорії кодування, надмірність коду R визначається як відношення числа перевірочних символів до загальної довжини кодової комбінації. Це відношення вказує на те, яка частка загальної довжини коду використовується для перевірки на наявність помилок, тобто характеризує відносну надмірність коду. Кількість інформаційних символів k та перевірочних символів r використовуються для обчислення надмірності коду.

Вектор помилок "е" використовується для ідентифікації місць у кодовій комбінації, де відбулися спотворення символів. Цей вектор є n -бітовою комбінацією, де кожна одиниця вказує на положення спотворених символів у кодовій комбінації.

Вага вектора помилки визначає кількість спотворень, тобто кількість символів, які були неправильно передані або спотворені. Вага вектора помилки може бути обчислена як кількість одиниць у векторі помилки.

Щоб відновити неспотворену комбінацію, застосовується операція суми по модулю 2 між спотвореною кодовою комбінацією та вектором помилок. Це означає, що для кожного біта в спотвореній комбінації та векторі помилок виконується операція XOR (виключне "або"). Результат цієї операції буде вихідною неспотвореною комбінацією.

Таким чином, використовуючи вектор помилок та операцію суми по модулю 2, можна відновити початкову неспотворену комбінацію зі спотвореної кодової комбінації.

ВИСНОВКИ ДО РОЗДІЛУ 1

У розділі було розглянуто різні підходи до захисту цифрових пристроїв від завад та помилок.

Залежно від характеру завад можуть використовуватися різні види кодів, зокрема, блочні, циклічні, конволютивні та нероздільні.

Блочні коди складаються з блоків фіксованої довжини, кожен з яких кодує окремі символи. Циклічні коди забезпечують стійкість до помилок шляхом додавання до повідомлення контрольної суми, яка може бути використана для виявлення та виправлення помилок. Конволютивні коди використовують властивість зсуву, щоб кодувати послідовність символів, та забезпечують стійкість до завад шляхом додавання додаткової інформації до повідомлення.

Крім того, було розглянуто такі види завадостійких кодів, як коди Хеммінга та Боуза-Чоудхурі-Хоквінгема та ін., а також було описано їхні особливості та переваги.

Також були розглянуті основні характеристики кодів. Довжина коду визначає кількість символів в кодовій комбінації, підстава коду вказує на кількість різних значень, які можуть бути закодовані, а потужність коду відображає кількість дозволених кодових комбінацій. Надмірність коду вказує на те, яка частка загальної довжини коду використовується для перевірки наявності помилок, тобто характеризує відносну надмірність коду. Кількість інформаційних та перевірочних символів використовуються для обчислення надмірності коду. Враховуючи ці характеристики, можна обрати оптимальний код для конкретного випадку.

У цілому, було показано, що застосування завадостійких кодів є важливим елементом захисту цифрових пристроїв від помилок та завад, що можуть виникати в процесі їх роботи. Основні переваги застосування завадостійких кодів полягають в тому, що вони дозволяють збільшити

завадостійкість пристроїв без необхідності в резервуванні та дублюванні апаратури, що робить їх більш ефективними та зручними в експлуатації.

2. БЛОКОВІ ЗАВАДОСТІЙКІ КОРИГУЮЧІ КОДИ

2.1. Загальна теорія виправлення помилок

Коректуючі коди, також відомі як завадостійкі коди, є комплексним класом кодів, які використовуються для виявлення та виправлення помилок в передачі даних. Вони є важливим і широко застосовуваним інструментом в багатьох сферах, включаючи телекомунікації, комп'ютерні мережі, зберігання даних на носіях, таких як CD, DVD, флеш-пам'ять, а також в багатьох інших додатках.

Одним з ключових завдань коректуючих кодів є виявлення та виправлення помилок, які можуть виникнути під час передачі даних по каналам зі зниженою якістю, таких як шумні радіоканали, провідні лінії зі спотворенням сигналу або неправильне зчитування даних з пам'яті. Ці помилки можуть спричинити втрату або спотворення даних, що може призвести до неправильної роботи системи або недостовірної інформації.

Для вирішення цих проблем використовуються різноманітні коректуючі коди, такі як Ріда-Соломона, Хеммінга, Гадамарда, Експандерні, Голая, Рід-Мюллера та інші. Ці коди можуть бути лінійними або нелінійними, а також можуть мати різні структури, такі як блочні, циклічні, некеровані (унітарні) та інші.

Перш за все, варто зрозуміти, звідки узагалі беруться помилки, які ми збираємося виправляти. Завдання полягає у передачі кількох блоків даних, кожен з яких кодується послідовністю двійкових цифр. Отримана послідовність нулів та одиниць передається через канал зв'язку. Однак, реальні канали зв'язку часто піддаються помилкам. Загалом помилки можуть мати різні види — може з'явитися зайвий символ або ж деяка втрата. Проте розглядатимемо лише ті ситуації, коли у каналі можуть відбуватися

заміни нуля на одиницю та навпаки. Щоб спростити ситуацію, вважаємо, що такі заміни мають однакову ймовірність. Помилка - це малоїмовірна подія, а отже, ймовірність двох помилок менше, а трьох взагалі мало. Можемо вибрати допустиме значення ймовірності, відзначивши межу "це вже точно неможливо". Це дозволить заявити, що у каналі може бути не більше, ніж k помилок. Це стане характеристикою каналу зв'язку.

Для того щоб код міг виправляти більше помилок, необхідно, щоб відстані були якомога ширші. З іншого боку, вони не повинні перетинатися, оскільки якщо точка потрапить в область перетину, незрозуміло, до якої околиці її віднести. В коді з подвоєнням між кодовими словами 00 і 11 відстань дорівнює 2 (обидва розряди відрізняються)(рис 2.1). Це означає, що якщо ми побудуємо навколо них кулі радіусом 1, вони будуть торкатися. Це означає, що точка торкання буде належати обом кулям і незрозуміло буде, до якої з них її віднести.

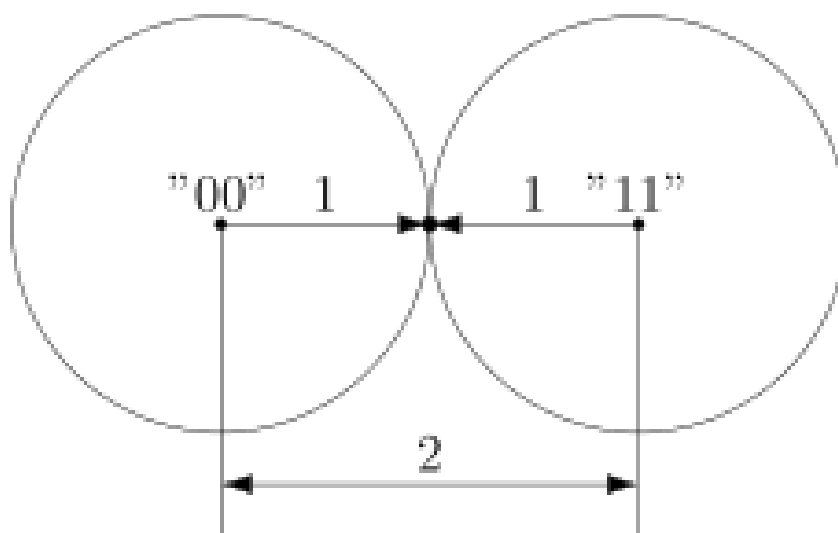


Рисунок 2.1 - Мінімальна кодова відстань

Цей результат має велике значення. Він означає, що код з мінімальною відстанню між кодовими словами d_{min} буде успішно працювати в каналі з k помилками, якщо виконується співвідношення (1)

$$d_{min} \geq 2k + 1 \quad (1)$$

Отримане рівняння дозволяє легко визначити, скільки помилок може виправити певний код. А скільки помилок може виявити код? Розмірковування такі ж. Код виявляє k помилок, якщо в результаті не отримується інше кодове слово (рис 2.2). Іншими словами, кодові слова не повинні знаходитися в околицях радіусом k інших кодових слів. Математично це записується так: $d_{min} \geq k + 1$

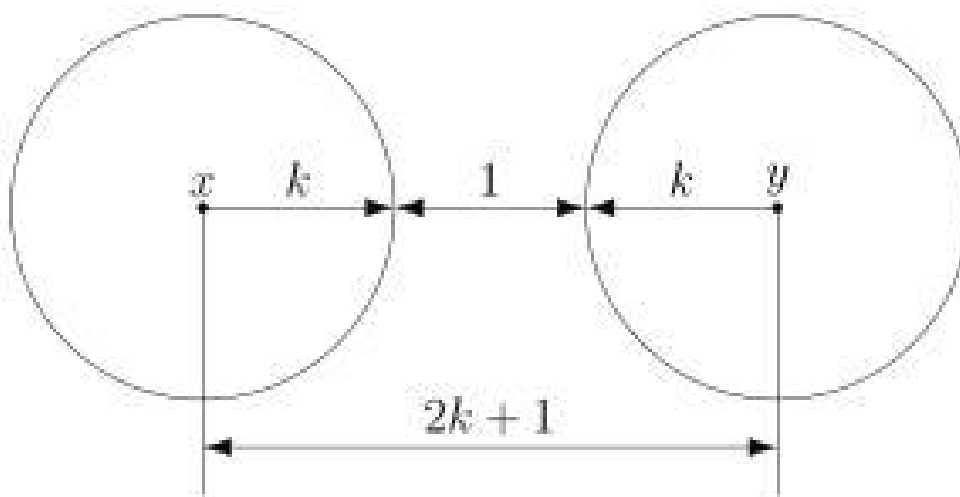


Рисунок 2.2 - Ілюстрація виправлення k помилок

2.2. Відстань Хеммінга

Відстань Хеммінга - це метрика відмінності між двома об'єктами однакової вимірності. У загальному випадку, вона застосовується для рядків будь-якої абетки з q символів однакової довжини. Відстань Хеммінга обчислюється як кількість позицій, у яких відповідні символи двох рядків різні. У контексті двійкових послідовностей, відстань Хеммінга є числом позицій, у яких відповідні біти двох послідовностей різні [4].

Ця метрика широко використовується в теорії кодування та інформації для визначення ефективності та якості кодів. Відстань Хеммінга може бути використана для виявлення та виправлення помилок в кодових послідовностях, таких як бітові послідовності. Вона також може використовуватися для порівняння двох послідовностей та для знаходження найближчого сусіда в певному множині об'єктів.

Загальною властивістю відстані Хеммінга є те, що вона задовольняє всім аксіомам метрики, тобто вона завжди буде невід'ємною, симетричною та виконувати нерівність трикутника. Відстань Хеммінга допомагає вирішувати широкий спектр завдань, пов'язаних зі знаходженням подібних об'єктів, виявленням та виправленням помилок, зіставленням та з ігноруванням різниць між об'єктами.

У більш загальному контексті, відстань Хеммінга є однією з метрик рядків, що використовується для вимірювання відстані редагування між двома послідовностями, незалежно від їхньої природи. Вона може бути застосована до рядків будь-якої абетки, складеної з q символів, тому що вона вимірює різницю між символами у двох послідовностях без урахування їхнього значення. У практичних застосуваннях, відстань Хеммінга використовується для визначення якості кодування, перевірки правильності передачі даних, та в інших сферах, де необхідно виміряти відмінності між двома послідовностями.

Відстань Хеммінга володіє властивостями метрики, що означає, що вона задовольняє наступним умовам: відстань між двома об'єктами завжди додатня або нульова, вона симетрична (тобто відстань між об'єктом A і об'єктом B така ж, як і відстань між об'єктом B і об'єктом A), і вона задовольняє нерівності трикутника (тобто відстань між об'єктом A і об'єктом C не більша, ніж відстань між об'єктом A і об'єктом B плюс відстань між об'єктом B і об'єктом C).

Hamming distance = 3 —

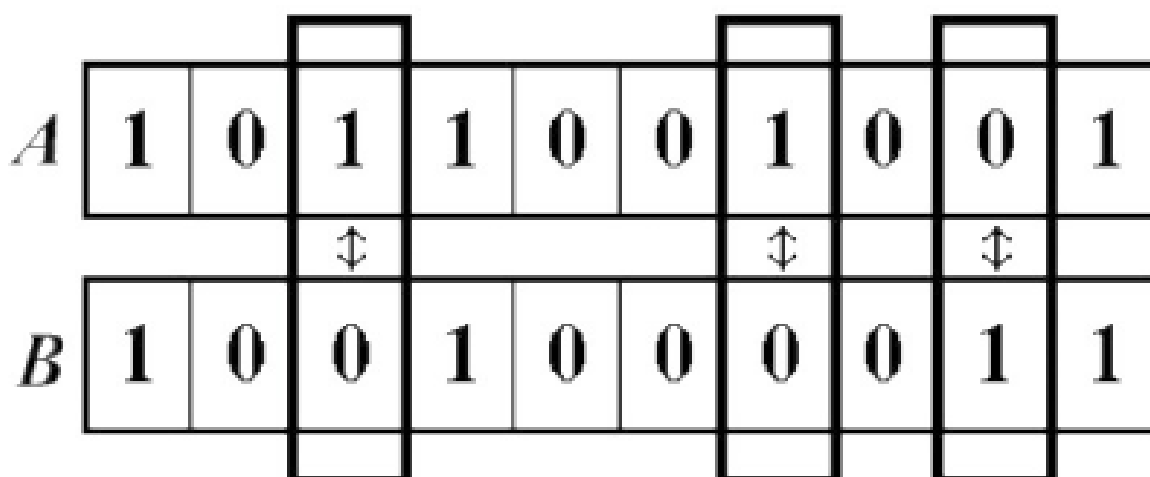


Рисунок 2.3 - Відстань Хемінга

2.3. Блочні завадостійкі коди

У теорії кодування блочні коди є важливою та великою родиною кодів, які використовуються для корекції помилок при передачі даних. Існує велика кількість прикладів блочних кодів, багато з яких мають практичне застосування. Абстрактне визначення блочних кодів є концептуально корисним, оскільки дозволяє досліджувати обмеження всіх блочних кодів у єдиної формі. Такі обмеження часто виражаються у вигляді меж, які пов'язують різні параметри блочного коду, такі як його швидкість та здатність виявляти та коригувати помилки. Приклади блочних кодів включають коди Ріда-Соломона, коди Хемінга, коди Адамара, коди Експандера, коди Голя та коди Ріда-Мюллера [5]. Ці приклади також належать до класу лінійних кодів, і тому їх називають лінійними блочними кодами. Більш конкретно, ці коди відомі як алгебраїчні блочні коди, або циклічні блочні коди, оскільки їх можна створити за допомогою булевих

поліномів. Алгебраїчні блочні коди зазвичай декодуються з використанням алгебраїчних декодерів. Проте, термін "блочний код" може також відноситися до будь-якого кодування з корекцією помилок, яке працює з блоками k біт вхідних даних, щоб згенерувати блоки n біт вихідних даних. Такі кодери є безпам'ятними пристроями. За цим визначенням коди, такі як турбокоди, закінчені конволютивні коди та інші ітеративно декодовані коди (подібні до турбо-кодів), також вважаються блочними кодами. Нетермінований конволютивний код буде прикладом неблочного (безрамкового) коду, який має пам'ять і класифікується як деревовидний код.

Прості коди з перевіркою на парність (непарність) є швидкими кодами з низькими корегуючими характеристиками, що використовуються для виявлення помилок при передачі даних. Для створення такого коду, до кожних k інформаційних біт додається ще один біт, таким чином, щоб кількість одиниць в кодовому слові була парною (parity). Цей біт називається бітом перевірки на парність. Такий код представляє собою $(k+1, k)$ -код або $(n, n-1)$ -код і має кодову відстань рівну двом, що означає, що цей код не може виправити помилки. Однак він може виявляти одну помилку в кодовій комбінації.

2.4. Межа Хемінга

Між параметрами блокового коду, а саме кількістю елементів в кодовій комбінації n , кількістю інформаційних елементів k , мінімальною кодовою відстанню Хеммінга d_{min} і кратністю помилок $t = (d_{min} - 1) / 2$ існує важливе співвідношення, яке відоме як межа Хеммінга (1). Це співвідношення визначає, наскільки ефективно код може виявляти та виправляти помилки передачі інформації в каналі зв'язку.

$$2^{n-k} \geq \sum_{i=0}^t \frac{n!}{i!(n-i)!} \quad (1)$$

Зокрема, відповідно до межі Хеммінга (рис 2.5), код з кількістю інформаційних елементів k , довжиною кодової комбінації n і мінімальною кодовою відстанню Хеммінга d_{min} може виправити не більше ніж $(d_{min} - 1) / 2$ помилок в каналі зв'язку. Звісно, що цей результат може бути досягнутий тільки тоді, коли код буде сконструйований належним чином і відповідатиме вимогам мінімальної кодової відстані.

Коди, що задовольняють цю умову, називаються досконалими. Серед прикладів досконалих кодів можна згадати код Хеммінга, який дозволяє виправляти поодинокі помилки, а також код Голя (рис. 2.4). Інші коди можуть виправляти лише деякі конфігурації помилок, кратність яких перевищує t . Для таких кодів можна встановити верхню межу на кількість помилок, яку вони можуть виправити. Така межа може бути корисною при визначенні потенційної ефективності коду в різних умовах, а також при порівнянні різних кодів.

$$\begin{bmatrix} 1 & 1 & 1 & 2 & 2 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 2 & 1 & 0 & 2 & 0 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 1 & 2 & 0 & 0 & 1 & 0 & 0 \\ 1 & 2 & 0 & 1 & 2 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 2 & 2 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Рисунок 2.4 - Код Голя

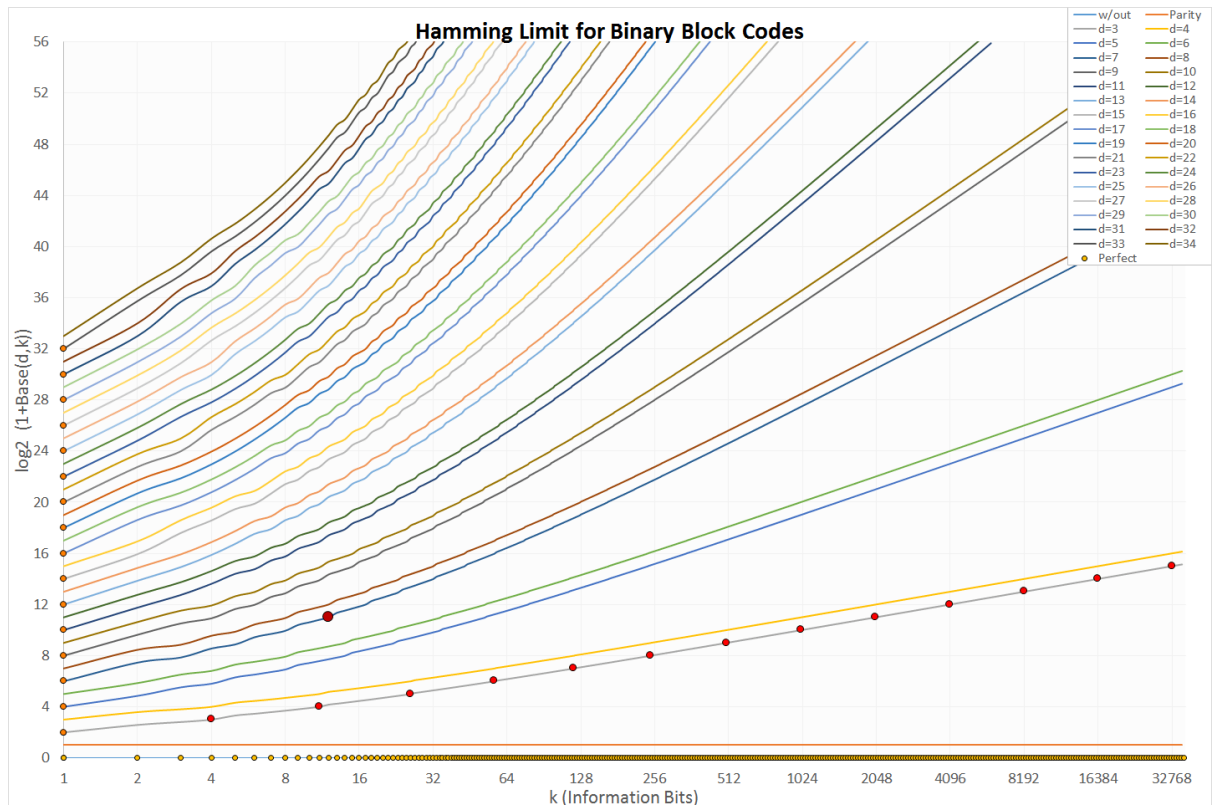


Рисунок 2.5 - Межа Хемінга

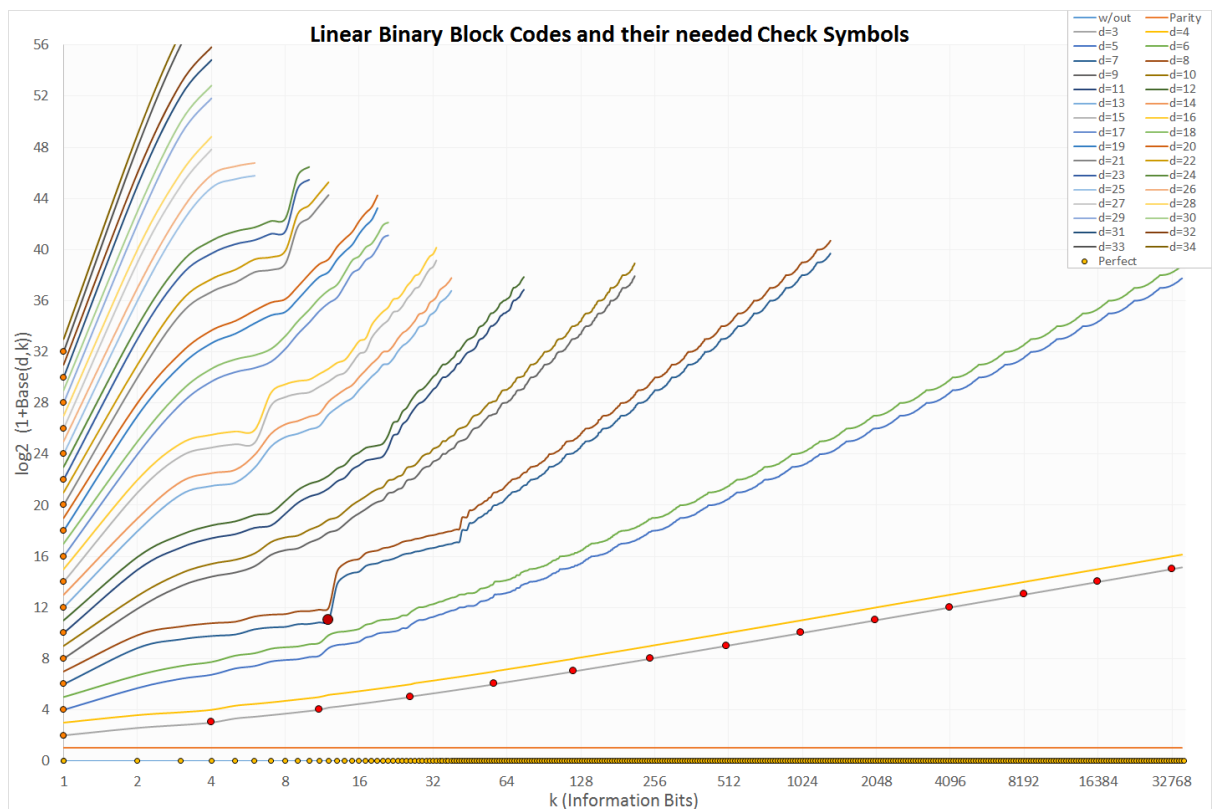


Рисунок 2.6 - Кількість перевірочних символів в блокових кодах

Блокові коди можна використовувати для виявлення помилок кратності u . У цьому випадку мінімальна відстань Хеммінга між словами коду має бути не меншою за $d_{min} = u + 1$. Якщо ж блоковий код призначений як для виправлення, так і для виявлення помилок кратності t та u відповідно, то мінімальна кодова відстань повинна задовольняти умові $d_{min} \geq t + u + 1$. Це означає, що найменша відстань між будь-якими двома кодовими словами повинна бути достатньою для виявлення та виправлення не тільки одиночних помилок, але й конфігурацій помилок кратності t та u .

2.5. Огляд конкретних нероздільних кодів

Однією з найбільш розвинутих областей теорії і практики кодування є побудова блочних роздільних кодів, які характеризуються чітким розподілом між інформаційними та додатковими бітами. Додаткові біти створюються на основі інформаційних бітів і відповідають за забезпечення надійності передачі даних.

Один з прикладів таких кодів - це коди Хеммінга, Боуза-Чоудхурі-Хоквінгема, які мають важливе значення в різних галузях інформаційної технології. Особливістю цих кодів є аналітичне визначення дозволених комбінацій інформаційних та додаткових бітів, що забезпечує зниження ймовірності помилок при передачі даних.

Для побудови кодерів і декодерів таких кодів використовуються різноманітні алгоритми, які забезпечують ефективність і надійність передачі інформації. Однією з переваг таких кодів є те, що їх реалізація не вимагає складних обчислень та використання великої кількості ресурсів.

Теорія та практика побудови блочних завадостійких кодів є актуальною в сучасній телекомунікаційній технології. Ці коди забезпечують надійний та ефективний передачу інформації з помилками в каналах зв'язку. У таких кодах використовується розподіл розрядів коду на

інформативні та додаткові, що сприяє ефективній передачі даних. Однак, в деяких випадках швидкість роздільних кодів може бути максимальною далеко не на всіх значеннях n . Наприклад, для коду Хемінга з дистанцією 3 максимально можлива швидкість досягається для значень n із ряду 7, 15, 31 і т.д.

Таким чином, для інших значень n можлива більш ефективна передача даних за допомогою нероздільних блочних завадостійких кодів, де в кодословах коду відсутній розподіл розрядів на інформативні та додаткові. Використання таких кодів може забезпечити збільшення швидкості передачі даних в певних випадках, що є важливим аспектом в сучасних телекомунікаційних системах.

Нероздільні коди демонструють кращі показники теоретичної швидкості, що стимулює проведення досліджень в наступних напрямках: розробка методів визначення складу кодослів у нероздільних кодах, створення ефективних кодерів та декодерів для нероздільних кодів. Варто зауважити, що на даний момент не існує загальної теорії побудови нероздільних кодів, яка б зберігала переваги по швидкості, як для самого коду, так і для його кодерів та декодерів.

У телекомунікаційних системах, нероздільні коди використовуються менш часто, ніж роздільні коди, оскільки вони зазвичай є менш потужними. Крім того, з'являється потреба у перетворенні вхідних комбінацій одного коду на вихідні комбінації іншого коду. При цьому, як і в роздільних кодах, виникає інформаційна надмірність, але вона залишається прихованою. Отже, використання нероздільних кодів для виявлення та виправлення помилок в комбінаціях після кодування в повідомленнях може бути складнішим, ніж у роздільних кодах.

Нероздільні коди мають перевагу в телекомунікаційних системах, оскільки дозволяють захистити передану інформацію від помилок та несанкціонованого доступу. Однак, їхній рівень захисту вважається

недостатнім для реальних задач і вимагає покращення. Наразі, досліджуються перспективні розробки, які поєднують захист від помилок та несанкціонованого доступу, що робить нероздільні коди більш привабливими для зберігання та передачі інформації в майбутньому. Залишається відкритим питання, які саме коди будуть найбільш перспективними в цьому напрямку.

Роздільні коди, що застосовуються в телекомунікаційних системах, не дозволяють ефективно організувати стійку обробку інформації цифровими автоматами через їхню природу. Однак, нероздільні коди дозволяють будувати завадостійкі цифрові автомати завдяки використанню лише дозволених комбінацій при синтезі коду. Це забезпечує помилковий перехід автомата в стан, який не є дозволеною комбінацією.

В порівнянні з нероздільними кодами, використання роздільних кодів у цифрових автоматах не є ефективним для забезпечення завадостійкості. Це пов'язано з тим, що автомати обробляють інформацію поетапно, відповідно до вхідних сигналів на кожному такті. Виявлення помилок в таких автоматах за допомогою роздільного кодування є складним, оскільки потрібно передбачити, яка буде комбінація на виході автомата на наступному такті та порівняти її з фактично з'явленою комбінацією. Тому, зазвичай, для забезпечення надійності роботи автоматів з роздільними кодами необхідно дублювати апаратуру або потроїти її, що значно збільшує вартість та складність цифрових пристроїв та систем.

Наприклад, рівноважний код, який є нероздільним, для виявлення помилки вимагає підрахунок кількості одиниць у кодовій комбінації та порівняння результату з контрольним числом. Це зробить процес виявлення помилок складнішим у порівнянні з кодом з перевіркою на парність або на непарність, де відбуваються тільки операції додавання одиниць за модулем 2. Крім того, ще потрібно перетворювати вхідні комбінації в комбінації

рівноважного коду, що може зробити процес кодування та декодування ще більш складним.

Один з найбільш відомих нероздільних є код з постійною вагою, в якому дозволені тільки комбінації, що містять визначену кількість одиниць, однакову у всіх дозволених комбінаціях. Найбільш поширений на практиці семирозрядний код з постійною вагою 3, який використовується при передаванні телеграфних повідомлень по короткохвильових радіоканалах і відомий як міжнародний телеграфний код № 3. Цей код виявляє всі одиничні помилки, а також усі помилки непарної кратності, але не виявляє помилок парної кратності, таких як зсуви (транспозиції). У цілком асиметричних каналах транспозиції не можуть мати місця, тому коди з постійною вагою виявляють усі помилки. Однак, недоліком таких кодів є те, що вони належать до числа нероздільних кодів, у комбінаціях яких неможливо виділити інформаційні та перевірні розряди, що ускладнює кодування і декодування та збільшує вартість апаратури.

Кодування нероздільних кодів полягає у включенні інформації в комбінації бітів, яка не може бути виділена окремо як інформаційний або перевірний розряд. Це означає, що при кодуванні інформації за допомогою нероздільних кодів, всі розряди мають однакову важливість і не можуть бути перевірені окремо.

У сфері блокового нероздільного кодування досягнення не є такими значними, порівняно з блоковими кодами, які передбачають роздільне кодування кожного символу інформації. При нероздільному кодуванні група інформаційних символів перетворюється на нову групу кодуючих символів, які будуть передаватися по каналу і не міститимуть у собі інформаційні символи у явному вигляді. Один з прикладів таких кодів - це коди Адамара, які були створені на основі матриці Адамара (рис. 2.7). Хоча всі коди Адамара базуються на цій матриці, побудова може дещо відрізнятися в залежності від призначення, наукового напрямку та автора.

Інженери, які використовують коди Адамара для передавання даних, зазвичай ставлять за мету досягнення максимальної швидкості кодування, навіть якщо це приводить до менш елегантної математичної конструкції. З іншого боку, для багатьох застосувань кодів Адамара в галузі теоретичної інформатики, досягнення оптимальної швидкості не є таким важливим, тому вони віддають перевагу простішим методам побудови, що дозволяє їх більш ефективно проаналізувати та зрозуміти їх властивості.

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix} \cdot$$

$n = 8$

Рисунок 2.7 - Матриця Адамара

Для будь-якої матриці Адамара можна побудувати код, який має властивість відстані Хемінга між кодovими словами рівною $n/2$. Для цього необхідно замінити всі значення $+1$ в матриці на 0 , а -1 - на 1 . Відповідна матриця отриманого коду буде мати розмірність n на $(n-1)$, причому перший стовпчик, що містить лише нулі, можна відкинути без зміни відстані Хемінга між кодovими словами. Таким чином, отримується код $C(n, n-1, n/2)$, де n - довжина кодovих слів, $n-1$ - кількість символів у кожному кодovому слові, а $n/2$ - відстань Хемінга між будь-якими двома кодovими словами. Це дозволяє побудувати широкий спектр кодів з різними

параметрами відповідно до потреб різних застосувань. Наприклад, інженери, які використовують ці коди для передавання даних, можуть зосередитися на максимальній швидкості кодування, в той час як теоретики кодування можуть бути зацікавлені в аналізі екстремальних властивостей цих кодів, що не обов'язково пов'язані з максимальною швидкістю.

2.6. Реалізація кодерів і декодерів

Останнім часом, у зв'язку з тенденцією до створення досить універсальних пристроїв захисту від помилок, нероздільні коди поступово витісняються систематичними і, зокрема, циклічними кодами.

Для ефективного використання кодів з корекцією та виявленням помилок необхідні спеціальні пристрої - кодери та декодери, які забезпечують відповідну обробку інформації. Реалізація цих пристроїв може мати програмний або апаратний характер, а зокрема, можуть бути реалізовані на програмованих логічних інтегральних схемах (ПЛІС).

Сучасні ПЛІС, зокрема FPGA (рис. 2.8) (поле-програмована матриця логічних вентилів), забезпечують високу функціональність та продуктивність. Такі ПЛІС можуть містити до мільйона логічних вентилів, кожен з яких може бути програмований для виконання будь-якої булевої функції від шести змінних. Це дозволяє реалізувати складні алгоритми обробки даних, в тому числі кодери та декодери, з високою швидкістю та точністю.

Використання FPGA для реалізації кодерів та декодерів важливо у масштабованих системах збереження даних, де вимоги до швидкості, надійності та ефективності є критичними. FPGA забезпечують високу швидкість обробки даних та можливість працювати з великим обсягом інформації. Це робить їх ідеальними кандидатами для реалізації кодеків у таких системах.

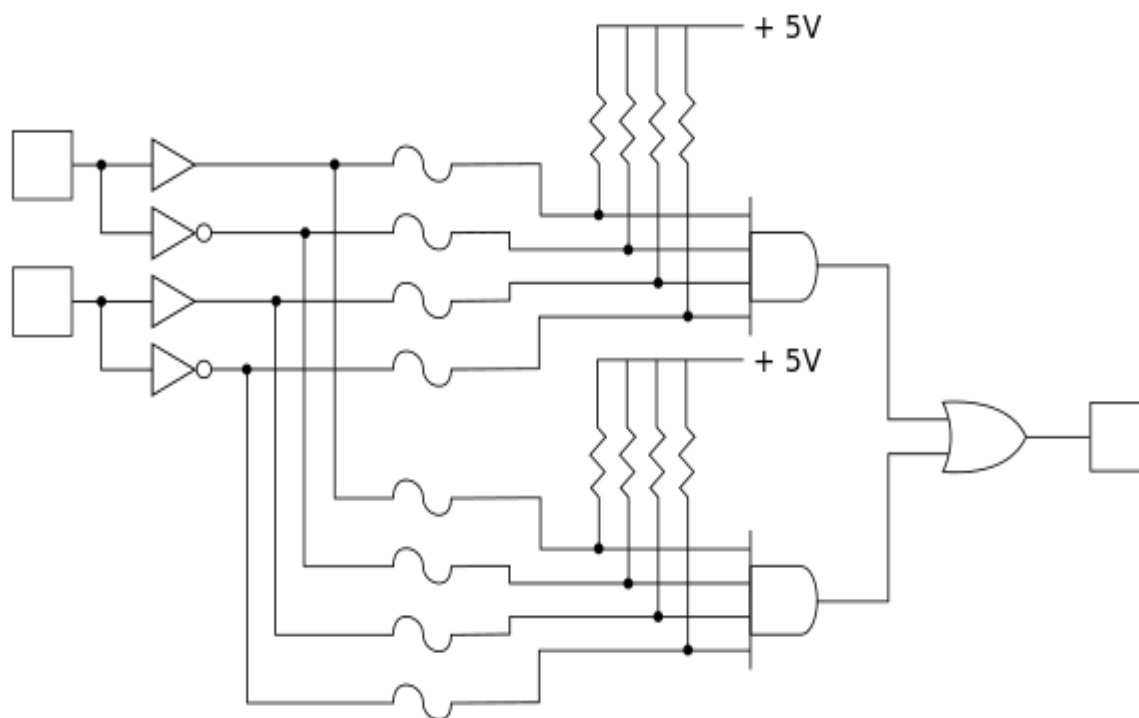


Рисунок 2.8 - ПЛІС

Оскільки завадостійкі коди використовуються в різних сферах, таких як телекомунікації, зберігання даних, космічна техніка тощо, ефективна реалізація кодерів та декодерів на ПЛІС має велике значення для забезпечення стабільної та надійної роботи систем, що використовують ці коди.

Для реалізації кодера необхідно реалізувати восьми булевих функцій від п'яти змінних. Сучасні ПЛІС FPGA блоки LUT можуть бути налаштовані на реалізацію будь-якої булевої функції від шести змінних або будь-яких двох булевих функцій від п'яти змінних. Для реалізації кодера достатньо 4 LUT для будь-якого 8-розрядного нероздільного коду при будь-якому назначенні кодословам пар кортежу. Однак, булеві функції кодера не визначені на спотворених входних даних, тобто таких, які не є відображенням десяткової цифри.

Для контролю можливих спотворень вхідних даних кодера можна реалізувати ще дві додаткові булеві функції, які б сигналізували наявність спотворень, а при довізначенні булевих функцій варто враховувати найбільш ймовірні спотворення. З точки зору апаратних витрат (по кількості LUT) довізначення булевих функцій може бути довільним.

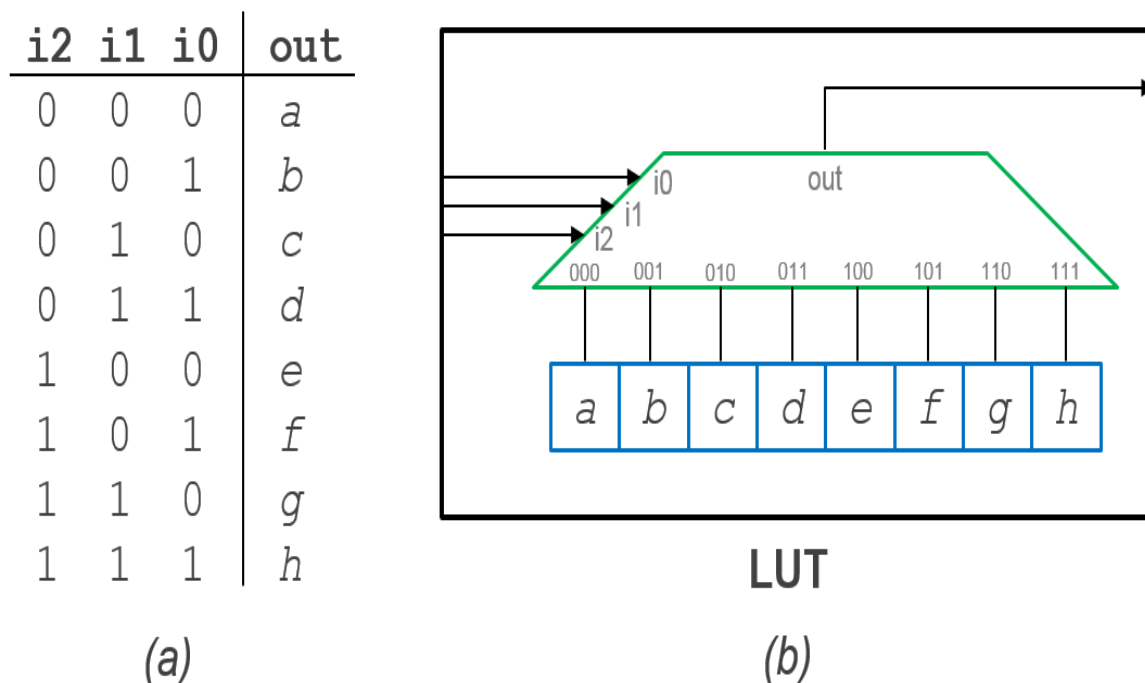


Рисунок 2.9 - LUT

Прототип пристрою для декодування корегуючих нероздільних 8-розрядних кодів складається з чотирьох вузлів. Перший вузол відповідає за визначення наявності помилки в сигналі. Другий вузол визначає наявність помилки в першому та п'ятому виходах пристрою. Третій вузол визначає наявність помилки в другому та третьому виходах, а четвертий - наявність помилки в четвертому виході. Проте булева функція для визначення подвійної помилки не мінімізується, а реалізована за допомогою розкладу Шеннона.

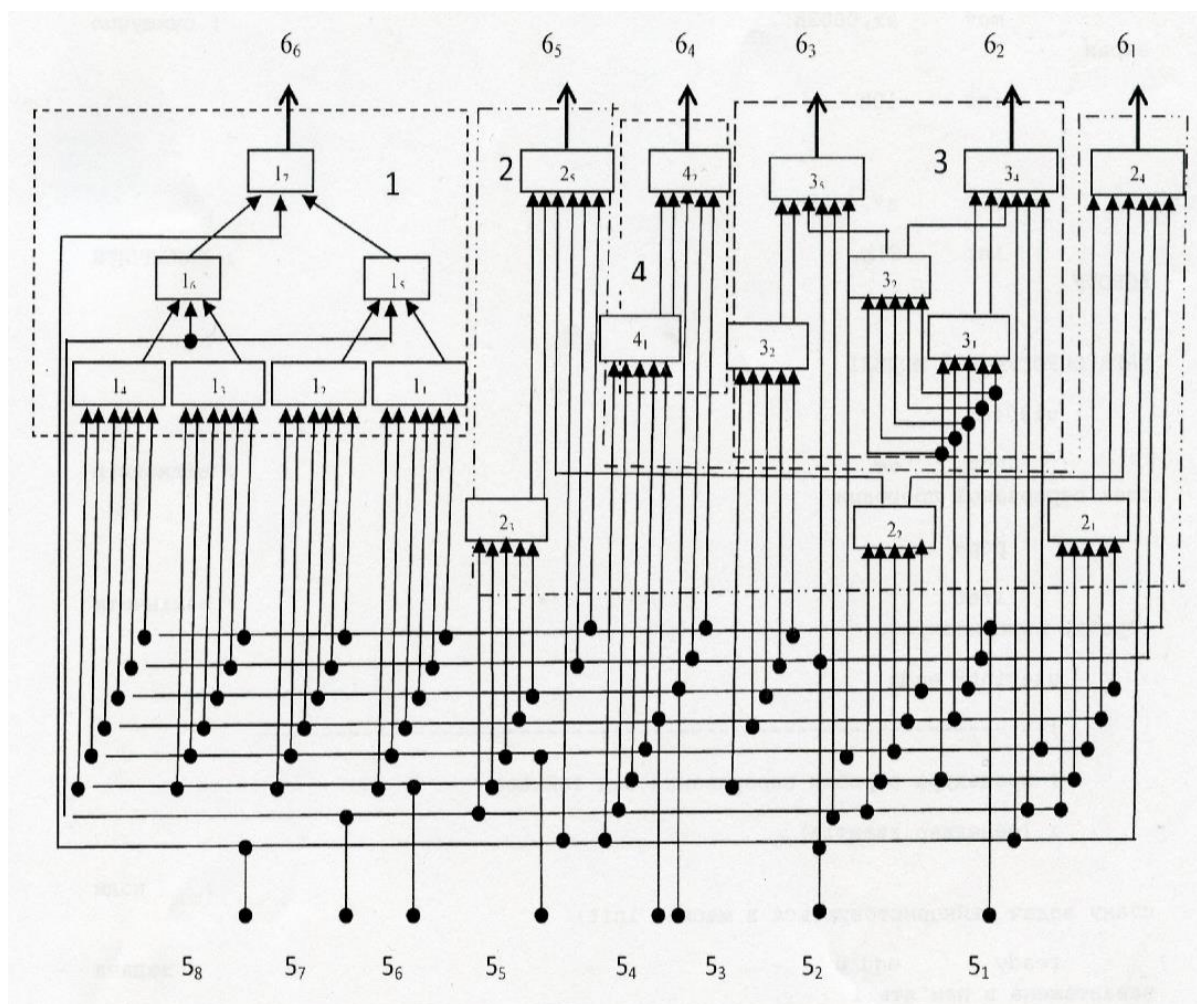


Рисунок 2.10 - Можливий декодер 8-розрядного коду

Розклад Шеннона - це спосіб подання булевої функції за допомогою комбінації логічних операцій "І" та "АБО". Цей метод полягає в тому, що функцію можна розкласти на декілька підфункцій, які залежать від підмножини вхідних змінних. Кожна підфункція може бути подана у вигляді логічного виразу, де використовуються логічні операції "І" та "АБО".

Наприклад, якщо функція залежить від трьох вхідних змінних, то її можна розкласти на три підфункції, кожна з яких залежить від двох змінних. Кожна з цих підфункцій може бути подана у вигляді логічного виразу, який містить логічні операції "І" та "АБО".

Розклад Шеннона дозволяє спростити булеві функції та зменшити кількість логічних елементів, які необхідно використовувати для їх реалізації. Таким чином, розклад Шеннона є важливим інструментом у дизайні логічних схем та цифрових пристроїв.

$$\bar{f}(x_1, \dots, x_n) = \bigvee_{\delta_1, \dots, \delta_n} x_1^{\delta_1} \dots x_n^{\delta_n} \bar{f}(\delta_1, \dots, \delta_n)$$

Рисунок 2.11 - Розклад Шеннона

ВИСНОВКИ ДО РОЗДІЛУ 2

У даному розділі було розглянуто блокові завадостійкі коди та їх використання для виявлення та виправлення помилок в передачі даних. Було описано, що блоковий код передає дані у вигляді блоків, кожен з яких містить інформацію та додаткову контрольну суму.

Для визначення якості блокового коду використовується відстань Хеммінга, яка вимірює мінімальну кількість бітів, які потрібно змінити, щоб перетворити одне кодове слово в інше. Межа Хеммінга визначає, як багато помилок можна виявити та виправити за допомогою даного коду.

Були описані різні способи виправлення помилок в залежності від типу блокового коду. Для простих кодів можна використовувати повторення бітів, а для більш складних - циклічні коди та коди Боуза-Чоудхурі-Хоквінгема.

Для використання блокових завадостійких кодів у реальних системах необхідно враховувати додаткові вимоги до швидкодії та ефективності коду. Також потрібно визначити, який тип коду найбільш підходить для конкретної задачі з врахуванням вимог до надійності та швидкодії передачі даних.

У цілому, блокові завадостійкі коди є ефективним інструментом для виявлення та виправлення помилок у передачі даних. Використовуючи ці коди, можна забезпечити достатню надійність передачі даних навіть в умовах наявності помилок у каналі зв'язку.

Розглядалися також можливі реалізації кодерів та декодерів для блокових завадостійких кодів. Було пояснено, що кодери та декодери можуть бути реалізовані як апаратні засоби, програмні засоби, або комбіновані рішення. Крім того, було розглянуто різні способи передачі даних за допомогою блокових завадостійких кодів, такі як простий режим передачі даних, каскадний режим передачі даних та інші.

В розділі також розглядалися нероздільні коди, які використовуються для корекції та виявлення помилок в передачі даних. Нероздільні коди є більш ефективними за роздільні в тому випадку, коли потрібно передавати багато даних, так як вони дозволяють зменшити кількість передаваних бітів. Крім того, у нероздільних кодах можуть бути використані прості арифметичні операції для виявлення та корекції помилок.

3. ДОСЛІДЖЕННЯ НЕРОЗДІЛЬНИХ КОДІВ ДЛЯ ВИПРАВЛЕННЯ ДВОХ ПОМИЛОК

3.1. Загальний алгоритм визначення кодів

Конкретний алгоритм використання нероздільних блочних 8-розрядних кодів для виправлення двох помилок може залежати від конкретної реалізації коду. Але загальний алгоритм може виглядати наступним чином:

1. Розбиваємо повідомлення, яке треба передати, на блоки довжиною в 2 біти.
2. Кожен блок кодується за допомогою нероздільного блочного 8-розрядного коду. Це дає нам кортеж кодів, кожен з яких представляє 2-розрядний блок в закодованому вигляді
3. При отриманні кожного блоку, здійснюється перевірка на наявність помилок за допомогою алгоритму декодування, що базується на відстані Хемінга.
4. Якщо виявляється, що в блоку є дві помилки, то застосовується алгоритм виправлення помилок на основі кодового слова, який дозволяє знайти та виправити дві помилки в одному блоку.

При використанні нероздільних блочних 8-розрядних кодів для виправлення двох помилок, оптимальною є мінімальна відстань 5 між будь-якими двома кодовими словами. Це означає, що відстань між будь-якими двома різними кодовими словами становить принаймні п'ять бітів. Така велика відстань між кодовими словами дозволяє виявляти та виправляти будь-які дві помилки, які відбулися в цьому кодовому слові. Наприклад, якщо виникають дві помилки, то існує тільки одне інше кодове слово,

відстань до якого становить не менше 5. Тому можна виявити, яке кодове слово було передано, порівнявши його з іншими кодовими словами, і виправити дві помилки в ньому. Використання мінімальної відстані 5 забезпечує ефективну корекцію помилок при використанні нероздільних блочних 8-розрядних кодів.

3.2. Алгоритм Брона-Кербоша

На сьогодні одним з ефективних методів пошуку множини максимальних клік на графі є алгоритм Брона-Кербоша[6; 7]. Однак, цей алгоритм не може бути розпаралелений, що є його недоліком з урахуванням сучасної тенденції збільшення швидкодії обчислювальної техніки за рахунок багатоядерності. Розроблений голландськими математиками Броном та Кербошем у 1973 році, алгоритм Брона-Кербоша використовує факт, що будь-яка кліка є максимальним повним підграфом. Цей метод полягає в послідовному переборі всіх можливих клік графу з підрахунком розміру кожної з них та збереженням максимальної знайденої кліки. Оскільки перебір здійснюється без перевірки на коректність побудови кліки, а також без урахування її структури, алгоритм може бути дуже ресурсоємним та має складність порядку $O(3^{n/3})$.

Для використання на практиці алгоритм Брона-Кербоша може бути оптимізований шляхом використання побітових операцій та зберіганням інформації про вже перевірені вершини.

Алгоритм Брона-Кербоша є рекурсивною процедурою, яка починається з найпершої вершини, що являє собою повний підграф. На кожному кроці алгоритм намагається розширити вже побудований підграф, додаючи в нього нові вершини. Процедура оперує трьома множинами вершин:

1. **compsub** - Множина вершин, що вже включені до поточної кліки. Ця множина починається з однієї вершини, яка є початковою вершиною повного підграфу.

2. **candidates** - Множина вершин, які можуть бути додані до поточної кліки. Ця множина складається з сусідніх вершин до вершин, що вже включені до поточної кліки, але ще не включені до неї. Ці вершини ще не використовувалися в поточному шляху рекурсії.

3. **not** - Множина вершин, які були виключені з поточної кліки. Ця множина включає вершини, які не можуть бути додані до поточної кліки, оскільки вони не мають відповідних зв'язків з вершинами, що вже включені до кліки. Вони виключаються з розгляду на даному кроці рекурсії, але можуть бути включені на наступних кроках, якщо відповідні умови будуть виконані.

Під час роботи алгоритму Брона-Кербоша рекурсивно перебираються всі можливі комбінації вершин з множини 2, додаються до множини 1, і рекурсивно викликається наступний крок. Коли більше вершин не може бути додано до поточної кліки, алгоритм завершується і повертає знайдену максимальну кліку.

Алгоритм Брона-Кербоша, хоча ефективний для пошуку всіх клік у графі, також має кілька проблем:

1. Експоненційний час роботи: У загальному випадку, алгоритм Брона-Кербоша має експоненційну складність часу в залежності від кількості вершин у графі. Це робить його неефективним для великих графів, де кількість вершин велика.
2. Повторні обробки: Алгоритм може повторно обробляти підграфи, що призводить до зайвої обчислювальної витрати. Це стає особливо помітним у випадку графів з багатою кількістю клік або клік великого розміру.

3. Велика кількість знайдених клік: У деяких випадках, особливо в графах з великою кількістю вершин та ребер, алгоритм може знайти велику кількість клік. Це може призвести до значного збільшення обсягу даних та витрат на зберігання.
4. Невикористання властивостей графа: Алгоритм Брона-Кербоша не використовує специфічні властивості графа, такі як симетрія, структура або інші особливості. Це може призвести до нездатності використовувати деякі оптимізації або евристики, які можуть покращити продуктивність алгоритму.
5. Обмеження на розмір клік: Алгоритм Брона-Кербоша не завжди здатний знаходити найбільші кліки у графі. У деяких випадках, він може зупинятися на меншому розмірі клік, що може бути недостатнім для певних задач.

ПРОЦЕДУРА *extend(candidates, not):*

ПОКИ *candidates* НЕ порожньо І *not* НЕ містить вершини, з'єднаної з усіма вершинами з *candidates*,
ВИКОНУВАТИ:

- 1 Вибираємо вершину *v* з *candidates* і додаємо її в *compsub*
- 2 Формуємо *new_candidates* і *new_not*, видаляючи з *candidates* і *not* вершини, не з'єднані з *v*
- 3 **ЯКЩО** *new_candidates* і *new_not* порожні
- 4 **ТО** *compsub* - кліка
- 5 **ІНАКШЕ** рекурсивно викликаємо *extend(new_candidates, new_not)*
- 6 Видаляємо *v* з *compsub* і *candidates* і поміщаємо в *not*

Рисунок 3.1 - Алгоритм Брона-Кербоша

Для початку, алгоритм Брона-Кербоша може бути реалізований без застосування рекурсії, що дозволяє

підвищити швидкість його виконання. Для цього можна використовувати множину `comprsub` як стек та зберігати множини `candidates` і `not` у глобальному одновимірному масиві, використовуючи два індекси `ne` і `se`. При такому підході, перші `ne` елементів цього масиву відповідають елементам множини `not`, а наступні `se-ne` елементів відповідають елементам множини `candidates`.

Для перевірки умов у рядках 1 та 4 можна використовувати наступні твердження:

- `ne = se`: множина `candidates` порожня
- `ne = 0`: множина `not` порожня
- `se = 0`: обидві множини `candidates` і `not` порожні
- Для переміщення елементу з `candidates` в `not` потрібно збільшити індекс `ne` (за умови, що як вершина-кандидат обиралася перша вершина з `candidates`, або вибрана вершина була обмінювана з першою)

Для покращення продуктивності алгоритму можна використовувати різні евристичні стратегії при виборі вершин-кандидатів на першому кроці. Наприклад, однією з таких стратегій є вибір вершин, які мають більше сусідів у поточній максимальній кліці. Це обумовлено тим, що такі вершини можуть бути більш ймовірними кандидатами на включення в нову максимальну кліку.

Крім того, можна використовувати підхід з виключенням вершин, які гарантовано не можуть бути частиною максимальної кліки. Цей підхід базується на використанні інформації про попередні максимальні кліки. За допомогою цієї інформації можна виключити з множини кандидатів ті вершини, які вже були перевірені і не входили до максимальних клік у попередніх ітераціях. Це дозволяє зменшити кількість вершин, які потрібно перевірити, та прискорити виконання алгоритму.

Також можуть використовуватись інші евристичні методи вибору кандидатів, які базуються на специфічних властивостях графа або задачі, що вирішується. Наприклад, можна використовувати метрики центральності графа для вибору вершин, які мають більший вплив на структуру графа і, ймовірно, більше сприяють утворенню максимальної кліки.

Всі ці евристичні підходи можуть бути комбіновані і адаптовані залежно від конкретної задачі та властивостей графа для досягнення кращої продуктивності алгоритму пошуку максимальних клік.

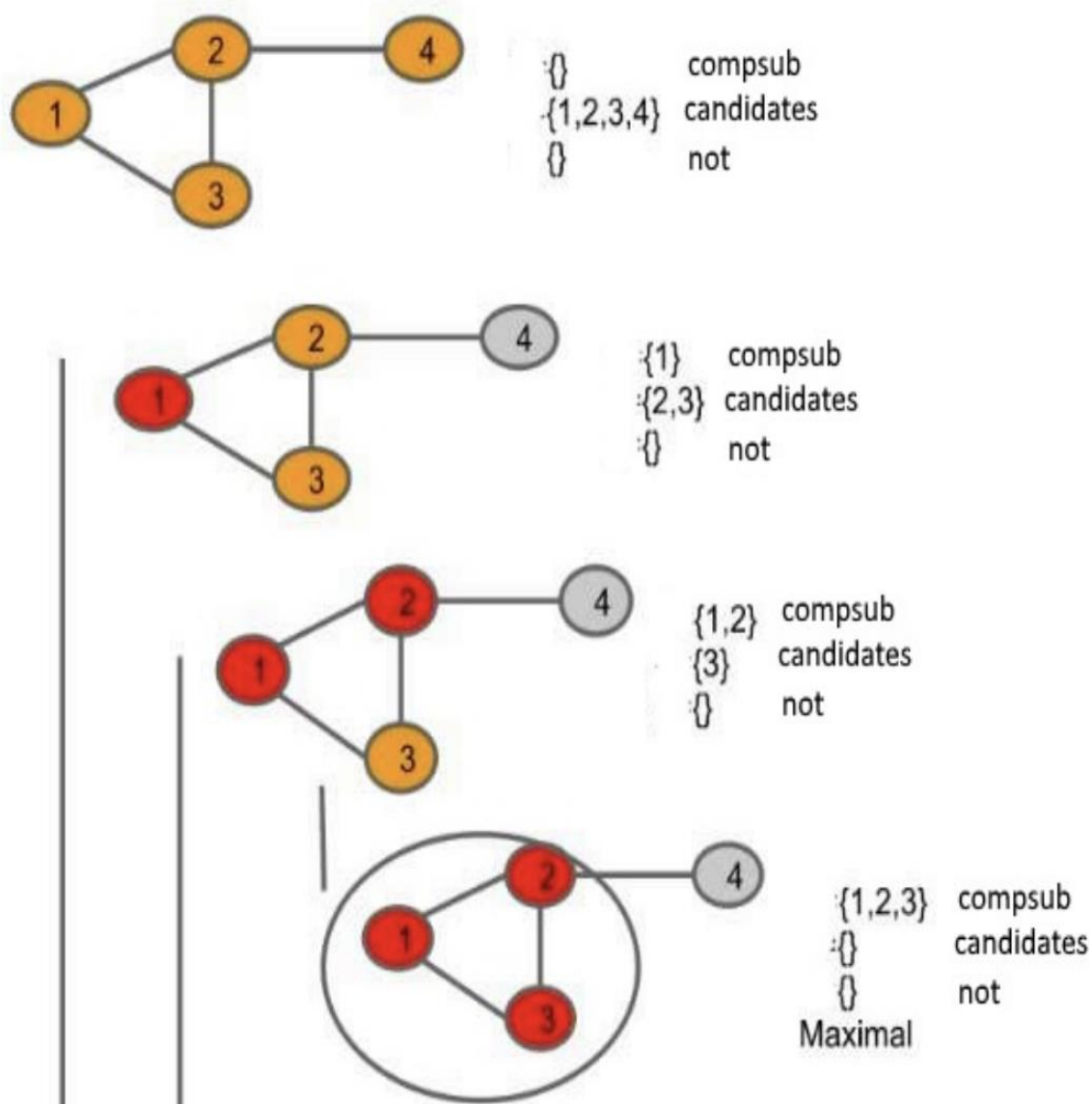


Рисунок 3.2 - Візуалізація алгоритма Брона-Кербоша

```

list<set<int> >kerbosh(int **a,int SIZE)
{
    set <int> M,G,K,P;
    list<set<int> > RESULT;
    for (int i=0; i<SIZE;i++)
    {
        K.insert(i);
    }
    int v,Count=0,cnt=0;
    int Stack1[100];
    std::set<int> Stack2[100];
    std::set<int>::iterator theIterator;
    theIterator=K.begin();
    while ((K.size()!=0) || (M.size()!=0))
    {
        if (K.size()!=0)
        {
            theIterator=K.begin();
            v=*theIterator;
            Stack2[++Count]=M;
            Stack2[++Count]=K;
            Stack2[++Count]=P;
            Stack1[++cnt]=v;
            M.insert(v);
            for (int i=0;i<SIZE;i++)
            {
                if (a[v][i])
                {
                    theIterator=K.find(i);
                    if (theIterator!=K.end())
                    {
                        K.erase(theIterator);
                    }
                    theIterator=P.find(i);
                    if (theIterator!=P.end())
                    {
                        P.erase(theIterator);
                    }
                }
            }
            theIterator=K.find(v);
            if (theIterator!=K.end())
            {
                K.erase(theIterator);
            }
        }
        else
        {
            if (P.size()==0)
            {
                RESULT.push_back(M);
            }
            v=Stack1[cnt--];
            P=Stack2[Count--];
            K=Stack2[Count--];
            M=Stack2[Count--];
            theIterator=K.find(v);
            if (theIterator!=K.end())
            {
                K.erase(theIterator);
            }
            P.insert(v);
        }
    }
    return RESULT;
}

```

Рисунок 3.3 - Реалізація алгоритма мовою C++

Основний алгоритм Брона-Кербоша [8] можна модифікувати для пошуку всіх 8-розрядних кодів, де відстань Хемінга між будь-якими двома кодовими словами не менше 5.

Основна ідея полягає в тому, що спочатку генерується всі можливі 8-розрядні послідовності. Для кожної пари кодових слів обчислюється відстань Хемінга за допомогою згенерованої таблиці відстаней. Якщо знайдена відстань менша за мінімальну відстань, то оновлюється її значення.

Якщо мінімальна відстань більша за або дорівнює 5, то поточна послідовність додається до списку правильних кодових слів. Для ефективного виконання цього алгоритму використовується змінна "мінімальна відстань" та таблиця відстаней, яку згенеровано за допомогою модифікованого алгоритму Брона-Кербоша.

```
def bron_kerbosch_modified(possible_clique,
    remaining_vertices,
    excluded_vertices,
    minimum_distance,
    distances,
    correct_sequences):
    """Модифікований алгоритм Брона-Кербоша."""
    if not remaining_vertices and not excluded_vertices:
        if minimum_distance >= 5:
            correct_sequences.append(possible_clique)
        return

    for vertex in remaining_vertices.copy():
        new_possible_clique = possible_clique + [vertex]
        new_remaining_vertices = [v for v in remaining_vertices if v in distances[vertex]]
        new_excluded_vertices = [v for v in excluded_vertices if v in distances[vertex]]
        bron_kerbosch_modified(new_possible_clique, new_remaining_vertices, new_excluded_vertices,
            min(minimum_distance, calculate_hamming_distance(vertex,
possible_clique[-1],
distances))),
            distances, correct_sequences)
        remaining_vertices.remove(vertex)
        excluded_vertices.append(vertex)
```

Рисунок 3.4 - Запропонована реалізація оптимізованого алгоритму

Для порівняння швидкості модифікованого алгоритму та оригінального алгоритму Брона-Кербоша, потрібно визначити час виконання кожного з них на вхідних даних однакового розміру та складності.

Оригінальний алгоритм Брона-Кербоша має складність $O(3^{(n/3)})$, де n - кількість вершин у графі. Для знаходження всіх 8-розрядних кодів, складність буде дуже великою, оскільки кількість 8-розрядних послідовностей становить $2^8 = 256$.

Модифікований алгоритм також має складність $O(3^{(n/3)})$, оскільки він виконує перебір всіх 8-розрядних послідовностей. Однак, у нашому випадку, $n = 8$, тому час виконання буде значно меншим, порівняно з оригінальним алгоритмом.

Залежно від реалізації та характеристик обчислювальної системи, можлива різна швидкість виконання для кожного з алгоритмів. Однак, в загальному, можна очікувати, що модифікований алгоритм буде працювати значно швидше, ніж оригінальний алгоритм, на вхідних даних фіксованого розміру та складності.

У цьому дослідженні було встановлено максимальну кількість кодослів у 8-розрядному коді, де відстань Хемінга між будь-якими двома кодословами становить не менше 5. Для цього використовувався алгоритм Брона-Кербоша, який дозволяє знайти всі максимальні кліки в графі.

У контексті пошуку 8-розрядних кодів з мінімальною відстанню Хемінга не менше 5, кожен кодослов розглядається як вершина графу. Дві вершини з'єднані ребром, якщо відстань Хемінга між ними менше 5. Таким чином, граф складається з усіх можливих кодослів, і ми шукаємо всі максимальні кліки в цьому графі.

Було встановлено, що максимальна кількість кодослів у 8-розрядному коді з відстанню Хемінга не менше 5 становить 4. Це означає, що в рамках

будь-якого 8-розрядного коду можна виділити набір з 4 кодословів, які мають відстань Хемінга не менше 5 між собою.

Дослідження також виявило, що загалом існує 17920 підмножин, що задовольняють умову мінімальної відстані Хемінга не менше 5. Це дає нам широкий простір для використання цих кодів у практичних застосуваннях, забезпечуючи надійну передачу даних з високою помилковою стійкістю.

Важливою є велика кількість знайдених блочних нероздільних кодів з відстанню Хеммінга не менше за 5 з кількох причин.

1. Велика кількість доступних кодів дозволяє забезпечити гнучкість і варіативність в виборі коду для конкретного застосування. Різні коди можуть мати різні характеристики та властивості, такі як надійність, ефективність, спотворення та витрати ресурсів. Мати велику кількість кодів дозволяє знайти оптимальний код для конкретних вимог та обмежень.
2. Велика кількість кодів сприяє досягненню більш високої ступені забезпечення завадостійкості. Знаходження кодів з відстанню Хеммінга не менше за 5 гарантує, що ці коди можуть виявляти та виправляти до двох помилок у переданому повідомленні. Чим більше доступних кодів, тим більша ймовірність знаходження оптимального коду, який забезпечує найкращу надійність і корекцію помилок для конкретної ситуації.
3. Велика кількість кодів дозволяє підібрати код з найкращою ефективністю ресурсів. Різні коди можуть вимагати різних кількостей пам'яті та обчислювальних ресурсів для кодування та декодування. Знаходження великої кількості кодів дозволяє вибрати той, який найкраще відповідає обмеженням системи, забезпечуючи ефективне використання ресурсів та оптимальну продуктивність.

```

# Знаходимо всі незалежні множини та їх розміри
cliques = []
bron_kerbosch(set(), set(graph.keys()), set(), graph, cliques)
clique_sizes = [len(clique) for clique in cliques]

# Знаходимо найбільшу незалежну множину
max_clique = max(cliques, key=len)

maximal_cliques = find_maximal_cliques(graph)
# Знаходимо найбільші незалежні множини
max_cliques = [clique for clique in cliques if len(clique) == len(max_cliques, key=len)]

# Знаходимо кількість клік довжиною 4
cliques_4 = [clique for clique in max_cliques if len(clique) == 4]
num_cliques_4 = len(cliques_4)

# Виводимо результати
print("Total number of 8-bit codes: ", len(codes))
print("Total number of edges in the graph: ", sum(len(neighbors) for neighbors in graph.values()) // 2)
print("Number of independent sets found: ", len(cliques))
print("Maximum independent set size: ", len(max_clique))
print("Maximum independent set: ", max_clique)
print("Number of cliques of size 4: ", num_cliques_4)

```

Рисунок 3.5 - Лістинг із виводом результатів

```

Total number of 8-bit codes: 256
Total number of edges in the graph: 11904
Number of independent sets found: 19072
Maximum independent set size: 4
Maximum independent set: {'10101000', '00011111', '11110110', '01000001'}
Number of cliques of size 4: 17920

```

Рисунок 3.6 - Результати

3.3. Створення кодеру та декодеру для нероздільних кодів

Нехай необхідно передавати по каналу довільний файл. Оскільки максимальний розмір отриманої кліки дорівнює 4, то розбиваємо його на

частинки довжиною по 2 розряди (2 біти). Кожна з цих частинок може мати лише 4 значення.

Обираємо будь який із одержаних 8-розрядних кодів на 4 кодослова.

Задача полягає у перекодуванні двобітових значень в 8-розрядний код, де кожне двобітове значення замінюється на кодове слово, що вибирається з попередньо визначеного коду.

Для кожного можливого значення двох бітів буде встановлене відповідне кодове слово, і кожне двобітове значення буде перекодовано в 8-розрядний код, використовуючи визначений код.

Після передачі даних за допомогою 8-розрядного коду на приймальній стороні проводиться процес декодування, за допомогою якого кодові слова перетворюються на двобітові значення, і відбувається виправлення будь-яких спотворень, що можуть виникнути під час передачі.

В рамках цього процесу будь-яке спотворення на кодослово, які не перевищують двох, буде виправлено. Коли всі спотворення будуть виправлені, отримані двобітові значення можуть бути використані для подальшої обробки і аналізу даних.

Кодер реалізується комбінаційною на два входи і 8 виходів, то він приймає два входи з можливими значеннями 0 або 1 і перетворює їх у восьмибітове двійкове число, яке відповідає одному з восьми можливих варіантів вхідних значень.

Кожен вихід кодера пов'язаний з однією з можливих комбінацій вхідних значень, тому на кожен вихід кодера можна підключити окрему булеву функцію від двох змінних.

Визначимо булеві функції кодера і декодера для довільного коду на 4 кодослова із тих, що задовольняють умові. Нехай обраний код буде включати в себе наступні кодослова 00000111, 11100100, 00011000, 11111011.

Кодер реалізується комбінаційною на два входи і 8 виходів, то він приймає два входи з можливими значеннями 0 або 1 і перетворює їх у восьмибітове двійкове число, яке відповідає одному з восьми можливих варіантів вхідних значень.

Кожен вихід кодера пов'язаний з однією з можливих комбінацій вхідних значень, тому на кожен вихід кодера можна підключити окрему булеву функцію від двох змінних.

Отже, кодер можна реалізувати за допомогою комбінації логічних елементів, таких як І, АБО, НЕ і т.д.

Кількість логічних елементів, необхідних для реалізації кодера, залежить від використаних булевих функцій. Наприклад, для реалізації кодера за допомогою функцій "І" і "НЕ" потрібно використовувати 6 логічних елементів.

Булеві функції кодеру:

- Функція 1: $F1(A, B) = A$
- Функція 2: $F2(A, B) = B$
- Функція 3: $F3(A, B) = A \text{ XOR } B$
- Функція 4: $F4(A, B) = A \text{ AND } B$
- Функція 5: $F5(A, B) = \text{NOT } A$
- Функція 6: $F6(A, B) = \text{NOT } B$
- Функція 7: $F7(A, B) = \text{NOR } (A, B)$
- Функція 8: $F8(A, B) = \text{NAND } (A, B)$

A	B	F1	F2	F3	F4	F5	F6	F7	F8
0	0	0	0	0	0	1	1	1	1
0	1	0	1	1	0	1	0	0	1
1	0	1	0	1	0	0	1	0	1
1	1	1	1	0	1	0	0	0	0

Рисунок 3.7 - Таблица істинності кодеру

X0	X1	X2	X3	X4	X5	X6	X7	Вихід1	Вихід2
0	0	0	0	0	1	1	1	0	0
1	1	1	0	0	1	0	0	0	1
0	0	0	1	1	0	0	0	1	0
1	1	1	1	1	1	0	1	1	1

Рисунок 3.8 - Таблица істинності декодера

Однак крім обраного кодослова на декодер під дією завад можуть поступити коди із однією або двома помилками. Всього кодослів з однією помилкою може бути 8, кодослів із двома помилками - $C_2^8 = 28$ двійкових кодів.

На всіх двійкових кодах, одержаних із каналу зв'язку в результаті однієї або двох помилок в кодослові декодер їх виправляє і встановлює значення відповідні значення для Вихід1, Вихід2.

Задане слово: 00000111

Слова з 1 та 2 помилками (відстань Хеммінга 1 або 2):

00000001

00000010

00000011

00000100

00000101

00000110

00001011

00001101

00001110

00001111

00010011

00010101

00010110

00010111

00011111

00100011

00100101

00100110

00100111

00101111

00110111

01000011

01000101

01000110

01000111

01001111

01010111

01100111

10000011

10000101

10000110

10000111

10001111

10010111

10100111

11000111

Задане слово: 11100100

Слова з 1 та 2 помилками (відстань Хеммінга 1 або 2):

00100100

01000100

01100000

01100100

01100101

01100110

01101100

01110100

10000100

10100000

10100100

10100101

10100110

10101100

10110100

11000000

11000100

11000101

11000110

11001100

11010100

11100000

11100001

11100010

11100101

11100110

11100111

11101000

11101100

11101101

11101110

11110000

11110100

11110101

11110110

11111100

Задане слово: 00011000

Слова з 1 та 2 помилками (відстань Хеммінга 1 або 2):

00000000

00001000

00001001

00001010

00001100

00010000

00010001

00010010

00010100

00011001

00011010

00011011

00011100

00011101

00011110

00101000

00110000

00111000

00111001

00111010

00111100

01001000

01010000

01011000

01011001

01011010

01011100

01111000

10001000

10010000

10011000

10011001

10011010

10011100

10111000

11011000

```
Задане слово: 11111011
Слова з 1 та 2 помилками (відстань Хеммінга 1 або 2):
00111011
01011011
01101011
01110011
01111001
01111010
01111011
01111111
10011011
10101011
10110011
10111001
10111010
10111011
10111111
11001011
11010011
11011001
11011010
11011011
11011111
11100011
11101001
11101010
11101011
11101111
11110001
11110010
11110011
11110111
11111000
11111001
11111010
11111101
11111110
11111111
```

Рисунок 3.9 - Можливі входи декодеру.

```

def hamming_distance(word1, word2):
    """
    Функція для обчислення відстані Хеммінга між двома 8-розрядними кодословами.
    """
    distance = 0
    for bit1, bit2 in zip(word1, word2):
        if bit1 != bit2:
            distance += 1
    return distance

def find_similar_words(input_word, max_distance):
    """
    Функція для пошуку та виведення всіх 8-розрядних кодослів, відстань Хеммінга яких до заданого слова становить
    1 або 2.
    """
    similar_words = []
    for i in range(256): # 256 можливих 8-розрядних кодослів
        word = format(i, '08b') # Перетворення числа у 8-розрядний бінарний рядок
        distance = hamming_distance(input_word, word)
        if distance == 1 or distance == 2:
            similar_words.append(word)
    return similar_words

# Приклад використання:
input_word = '11111011'
similar_words = find_similar_words(input_word, max_distance=2)

print(f"Задане слово: {input_word}")
print("Слова з 1 та 2 помилками (відстань Хеммінга 1 або 2):")
for word in similar_words:
    print(word)
print(len(similar_words))

```

Рисунок 3.10 - Лістинг визначення входів декодера.

```

{'10011000', '00000110', '11110011', '01101101'}
{'10011000', '01100001', '00000110', '11111111'}
{'10011000', '00000110', '11100101', '01111011'}
{'00000110', '11001001', '00111011', '11110100'}
{'00000110', '00011001', '11101011', '11110100'}
{'11001011', '00000110', '00111001', '11110100'}
{'00000110', '10001001', '01111011', '11110100'}
{'01011001', '00000110', '10101011', '11110100'}
{'10111111', '01101000', '00000110', '11010001'}
{'01101000', '00000110', '11010011', '10111101'}
{'01101000', '00000110', '10010001', '11111111'}
{'01101000', '00000110', '11110111', '10011001'}
{'01010001', '00000110', '11111100', '10101011'}
{'01100001', '00000110', '11111100', '10011011'}
{'11000001', '00000110', '11111100', '00111011'}
{'00000110', '11111100', '11010011', '00101001'}
{'11001011', '00000110', '11111100', '00110001'}
{'11111010', '00000110', '11001101', '00110001'}
{'11111110', '00000110', '11001001', '00110001'}
{'00000110', '00110001', '11111111', '11001000'}
{'11101000', '00000110', '00110001', '11011111'}
{'00000110', '00110001', '11011100', '11101011'}
{'00000110', '11001101', '01111011', '10110000'}
{'10100001', '00000110', '01111011', '11011100'}
{'11101100', '00000110', '10010001', '01111011'}
{'00000110', '10110101', '01111011', '11001000'}
{'11000001', '00000110', '01111011', '10111100'}
{'00000110', '10101011', '01110001', '11011100'}
{'00000110', '00101001', '11110011', '11011100'}
{'11101100', '00000110', '10011011', '01110001'}
{'11010001', '11101100', '00000110', '00111011'}
{'11101100', '00000110', '11110011', '00011001'}
{'11101100', '00000110', '11010011', '00111001'}
{'00111011', '00000110', '11011000', '11100101'}
{'11110111', '00000110', '11011000', '00101001'}
{'10111111', '01100001', '00000110', '11011000'}
{'10111111', '01010001', '00000110', '11101000'}
{'10111111', '00000110', '11001001', '01110000'}
{'10111111', '00000110', '01110001', '11001000'}

```

```

{'10011000', '11000011', '01111110', '00100101'}
{'10011000', '10000111', '01111110', '01100001'}
{'10011000', '10100101', '01111110', '01000011'}
{'10011000', '11110011', '00000101', '01101110'}
{'10011000', '00000101', '11100110', '01111011'}
{'10011000', '00000101', '01110110', '11101011'}
{'10011000', '01101110', '11000011', '00110101'}
{'10011000', '01101100', '00110111', '11000011'}
{'10011000', '00110110', '11000011', '01101101'}
{'10011000', '00101111', '11000011', '01110100'}
{'10011000', '01100100', '11000011', '00111111'}
{'10011000', '00100101', '01010110', '11101011'}
{'10011000', '00000111', '11100100', '01111011'}
{'10011000', '00110111', '01001011', '11100100'}
{'10011000', '01010011', '00101111', '11100100'}
{'10011000', '00111111', '11100100', '01000011'}
{'10011000', '10000111', '01101101', '01110010'}
{'10011000', '11110111', '01001011', '00100100'}
{'10011000', '10100101', '01001111', '01110010'}
{'10011000', '11110011', '01001111', '00100100'}
{'10011000', '01100001', '11000110', '00111111'}
{'10011000', '01000010', '11100101', '00111111'}
{'10011000', '11000110', '00100101', '01111011'}
{'10011000', '10000111', '01100100', '01111011'}
{'10011000', '01001011', '11100110', '00110101'}
{'10011000', '00101111', '11000101', '01110010'}
{'10011000', '11100110', '01110001', '00001111'}
{'10011000', '00110110', '01001011', '11100101'}
{'10011000', '00101110', '01010011', '11100101'}
{'10011000', '00001111', '11100101', '01110010'}
{'10011000', '11110011', '01100100', '00001111'}
{'10011000', '10100101', '01001011', '01110110'}
{'10011000', '01000010', '00100101', '11111111'}
{'10011000', '10100101', '01010011', '01101110'}
{'10011000', '00000111', '11101011', '01110100'}
{'10011000', '01100100', '00010111', '11101011'}
{'10011000', '01000100', '00110111', '11101011'}
{'10011000', '01010111', '00100100', '11101011'}

```

```

{'00001010', '01010111', '10111001', '11100100'}
{'10111010', '01100001', '00011101', '11000110'}
{'11000110', '00011101', '00100000', '11111011'}
{'01100001', '11001010', '00011101', '10110110'}
{'11010011', '00011101', '00101010', '11100100'}
{'11000110', '11101001', '00011101', '00110010'}
{'00110010', '11101100', '00011101', '11000011'}
{'11000001', '01001110', '00111001', '10110110'}
{'11010011', '10100100', '01001110', '00111001'}
{'10001001', '01001110', '01110001', '10110110'}
{'10010000', '01100001', '10111111', '01001110'}
{'10010011', '01100001', '01001110', '10111100'}
{'00111001', '10110110', '11000011', '01001100'}
{'01110111', '10101010', '10010001', '01001100'}
{'01100001', '11011101', '10110110', '00001010'}
{'01100001', '11000110', '00011011', '10111100'}
{'00001010', '11010011', '01100101', '10111100'}
{'11000010', '01100101', '00011011', '10111100'}
{'10010011', '01001010', '01100101', '10111100'}
{'01110111', '11000010', '00001001', '10111100'}
{'11000010', '01110001', '00001111', '10111100'}
{'01110111', '00001011', '11000000', '10111100'}
{'01110111', '00001010', '11000001', '10111100'}
{'00001011', '11000110', '01110001', '10111100'}
{'10010011', '11001101', '01111110', '00100000'}
{'10100100', '11010011', '01111110', '00001001'}
{'11000010', '10111001', '01111110', '00000101'}
{'11000010', '01111110', '10110101', '00001001'}
{'00001011', '11000000', '01111110', '10110101'}
{'00010000', '00001111', '11100100', '11111011'}
{'10111111', '01101100', '00010000', '11000011'}
{'10111010', '01010001', '00001111', '11100100'}
{'10111111', '01010001', '11100100', '00001010'}
{'10001001', '01011111', '00110010', '11100100'}
{'00111110', '11010011', '11100100', '00001001'}
{'00101111', '01011000', '11100100', '10010011'}
{'10010001', '01011111', '00101010', '11100100'}
{'01110111', '00010000', '10101010', '11001101'}
{'11000001', '01101100', '10110110', '00011011'}

```

```

{'10111111', '00001010', '11000001', '01110100'}
{'00101111', '01010001', '10011010', '11100100'}
{'01110111', '10011010', '00100000', '11001101'}
{'01010001', '10001011', '01101100', '10110110'}
{'00111110', '01010001', '10001011', '11100100'}
{'10100100', '01010001', '10001011', '01111110'}
{'10001011', '10110110', '01100101', '01011000'}
{'01110111', '10001011', '10100100', '01011000'}
{'11100000', '00111100', '10001011', '01010111'}
{'01110111', '11101100', '00010000', '10001011'}
{'01100001', '10001011', '10110110', '01011100'}
{'10001011', '10110110', '01110001', '01001100'}
{'11010111', '01101100', '10001001', '00110010'}
{'11010111', '10100100', '01001010', '00111001'}
{'11010111', '10100010', '01001100', '00111001'}
{'11010111', '10100100', '01001000', '00111011'}
{'11010111', '00010000', '10101010', '01101101'}
{'11010111', '11101100', '00110010', '00001001'}
{'11010111', '00001011', '00111100', '11100000'}
{'11010111', '00111110', '11100000', '00001001'}
{'11010111', '01100001', '10111100', '00001010'}
{'10111010', '10100101', '01010001', '01001110'}
{'10111010', '10100101', '01010111', '01001000'}
{'00111110', '10100101', '01010001', '11001010'}
{'10100101', '00101010', '11010011', '01011100'}
{'00111110', '10100101', '11010011', '01001000'}
{'10100101', '00010000', '01001110', '11111011'}
{'10100101', '11000010', '00111011', '01011100'}
{'00111110', '10100101', '11000010', '01011001'}
{'10100101', '00110010', '01001000', '11011111'}
{'10111111', '01011001', '11000110', '00100000'}
{'11000110', '01011001', '10110101', '00101010'}
{'10101111', '00100000', '11010011', '01011100'}
{'11000001', '10101111', '00110010', '01011100'}
{'10111111', '00100000', '11000011', '01011100'}
{'00101111', '11100000', '01011100', '10010011'}
{'00111011', '11000011', '10100100', '01011100'}
{'00101010', '11000011', '10110101', '01011100'}

```

```

{'10100001', '00111110', '01011001', '11000110'}
{'10100001', '01010111', '10111110', '01001000'}
{'11101110', '11010011', '00011101', '00100000'}
{'11101110', '11000001', '00110010', '00011101'}
{'11101110', '00111011', '00010100', '11000001'}
{'10111010', '01101111', '11000001', '00010100'}
{'01010001', '01101111', '10011010', '10100100'}
{'10100010', '01010001', '01101111', '10011100'}
{'01101111', '11010011', '00100000', '10011100'}
{'01101111', '10111001', '00010100', '11000010'}
{'10010011', '01101111', '10100100', '01011000'}
{'11000001', '01101111', '00110010', '10011100'}
{'10100100', '11001001', '00111011', '01010110'}
{'10111111', '11001001', '00100000', '01010110'}
{'11001001', '10110101', '00101010', '01010110'}
{'10100010', '00111001', '01010110', '11001101'}
{'11001011', '00111001', '10100100', '01010110'}
{'10100100', '00001001', '01010110', '11111011'}
{'10010001', '10101010', '01101101', '01010110'}
{'10111111', '11100000', '01010110', '00001001'}
{'00101111', '11000001', '10111000', '01010110'}
{'01100001', '10001011', '01010110', '10111100'}
{'01010001', '10100100', '00111111', '11001010'}
{'10100010', '11001001', '01010100', '00111111'}
{'11011010', '11000101', '00100000', '00111111'}
{'11101001', '11000110', '00010000', '00111111'}
{'10100100', '11000011', '00111111', '01011000'}
{'11101100', '00010000', '11000011', '00111111'}
{'01001010', '10010001', '00111111', '11100100'}
{'11001011', '00010000', '00111111', '11100100'}
{'11011010', '00000001', '00111111', '11100100'}
{'11010011', '01001000', '10100100', '00111111'}
{'10100010', '11000101', '00111111', '01011000'}
{'00110101', '10111010', '01001110', '11000001'}
{'00110101', '10111010', '11000011', '01001100'}
{'00110101', '01111010', '10001001', '11000110'}
{'00110101', '00001010', '11010011', '11101100'}
{'00110101', '11010011', '01001100', '10101010'}

```

Рисунок 3.11 - Деякі варіанти знайдених клік

ВИСНОВКИ ДО РОЗДІЛУ 3

В цьому розділі було розглянуто нероздільні блочні коди з відстанню Хеммінга не менше за 5. Було виявлено, що такі коди можуть бути ефективними при виправленні двох або більше помилок, оскільки вони забезпечують кращу кореляцію між даними бітами. Однак, їх побудова може бути складною, а досягнення потрібної завадостійкості може вимагати більшої вибірки кодів.

Пропонується використовувати знайдені 8-розрядні нероздільні коди з відстанню Хеммінга не менше за 5 у практичних застосуваннях. Зокрема, ці коди можуть бути використані при створенні кодувальних схем для масштабованих систем збереження даних, де надійність та ефективність збереження є ключовими факторами. Використання цих кодів допоможе забезпечити надійне збереження даних та забезпечити виправлення помилок при передачі.

Для практичної реалізації універсальних кодеків/декодерів 8-розрядних нероздільних кодів пропонується використовувати програмовані логічні пристрої (ПЛІС) або готові мікросхеми, які можуть бути налаштовані на будь-який зі знайдених кодів. Це дозволить забезпечити гнучкість та ефективність в реалізації кодувальних та декодувальних схем.

Ці нероздільні блочні коди можуть бути особливо ефективними у ситуаціях, коли вхідне повідомлення розбивається на частинки довжиною по 2 розряди (2 біти) через максимальний розмір кліки 4. Завдяки їх властивості виправлення двох або більше помилок, ці коди можуть забезпечити надійну передачу та виявлення спотворень у таких невеликих блоках даних.

Розбиття повідомлення на частинки довжиною по 2 розряди дозволяє ефективно використовувати ці коди для декодування та відновлення вихідної інформації. При виявленні помилок у таких блоках даних,

нероздільні блочні коди можуть не тільки ідентифікувати пошкоджені біти, але й виправляти їх, використовуючи кореляцію між бітами у кодослові.

Узагальнюючи, знайдені нероздільні блочні коди з відстанню Хеммінга не менше за 5 представляють інтерес для різноманітних застосувань, де надійність та ефективність передачі та збереження даних є важливими факторами.

Запропоновані варіанти використання нероздільних блочних кодів з відстанню Хеммінга не менше за 5 відкривають нові можливості у різних областях, де надійність, ефективність та аналіз даних є ключовими аспектами. Подальші дослідження та практична реалізація цих кодів можуть привести до розвитку нових методів кодування, передачі та збереження даних, що сприятиме покращенню різних технологій та систем.

ВИСНОВКИ

Метою магістерської роботи було визначення та дослідження блокових нероздільних завадостійких кодів для виправлення двох помилок та пошук можливих варіантів їх використання. В роботі було виділено 3 розділи

У першому розділі розглядалися різні підходи до захисту цифрових пристроїв від помилок та завад. Для забезпечення стійкості пристроїв до завад та помилок можуть використовуватися різні типи кодів, зокрема блочні, циклічні, конволютивні та нероздільні коди.

Блочні коди складаються з блоків фіксованої довжини, які кодують окремі символи, тоді як циклічні коди використовують контрольну суму для виявлення та виправлення помилок. Конволютивні коди використовують властивість зсуву для кодування послідовності символів та забезпечують стійкість до завад завдяки додатковій інформації. Застосування завадостійких кодів дозволяє збільшити завадостійкість пристроїв без необхідності в резервуванні апаратури, що робить їх більш ефективними та зручними в експлуатації. Отже, використання завадостійких кодів є важливим елементом захисту цифрових пристроїв від помилок та завад, які можуть виникати під час їх роботи.

У другому розділі було розглянуто різні види завадостійких кодів та їх використання для виявлення та виправлення помилок в передачі даних. Блокові завадостійкі коди використовуються для передачі даних у вигляді блоків, кожен з яких має додаткову контрольну суму. Важливою характеристикою блокових кодів є відстань Хеммінга, яка визначає межу кількості помилок, які можуть бути виявлені та виправлені кодом. Блокові завадостійкі коди є ефективним засобом для забезпечення надійної передачі даних, навіть у присутності помилок у каналі зв'язку.

В розділі також було розглянуто різні реалізації кодерів та декодерів для блокових завадостійких кодів, включаючи апаратні та програмні засоби. Було

описано різні способи передачі даних за допомогою цих кодів, такі як простий режим та каскадний режим передачі даних. Це дозволяє вибирати найбільш підходящий метод передачі даних залежно від конкретних потреб та обмежень системи.

Крім блокових кодів, було розглянуто й нероздільні коди, які є альтернативою роздільним кодам. Нероздільні коди є більш ефективними, оскільки дозволяють зменшити кількість передаваних бітів при передачі великого обсягу даних. Вони також можуть використовувати прості арифметичні операції для виявлення та виправлення помилок. Застосування нероздільних кодів може бути корисним у ситуаціях, де потрібно передавати значний обсяг інформації з мінімальною кількістю передаваних бітів.

У третьому розділі детально розглядалися нероздільні блочні коди з відстанню Хеммінга не менше за 5. Виявлено, що такі коди мають високу ефективність при виправленні двох або більше помилок, оскільки вони демонструють кращу кореляцію між бітами даних. Однак, побудова цих кодів може бути складною, а досягнення необхідної завадостійкості може вимагати великої вибірки кодів.

У практичних застосуваннях пропонується використовувати знайдені 8-розрядні нероздільні коди з відстанню Хеммінга не менше за 5. Ці коди можуть бути використані при розробці кодувальних схем для масштабованих систем збереження даних, де надійність та ефективність збереження є критичними. Використання цих кодів допоможе забезпечити надійне збереження даних та виправлення помилок при передачі.

Для практичної реалізації універсальних кодеків/декодерів 8-розрядних нероздільних кодів рекомендується використовувати програмовані логічні пристрої (ПЛІС) або готові мікросхеми, які можуть бути налаштовані на будь-який зі знайдених кодів. Це забезпечить гнучкість та ефективність в реалізації кодувальних та декодувальних схем.

Важливо зазначити, що нероздільні блочні коди можуть бути особливо ефективними у випадках, коли вхідне повідомлення розбивається на частинки довжиною 2 розряди (2 біти) через обмеження на розмір блоку (кліки) в розмірі 4.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Richard E. Blahut// Theory and practice of error control codes, 1984
2. Elwyn R. Berlekamp (2014), Algebraic Coding Theory, World Scientific Publishing (revised edition).
3. Error-Correction Coding and Decoding/F.M. Tomlinson, C.J. Tjhai, M.A. Ambroze, M. Ahmed, M. Jibril. –Cham, Switzerland: SpringerOpen, 2017. – 520 p.
4. The art of error correcting coding /R.H. Morelos-Zaragoza. –John Wiley, 2016. –263 p.3.The theory of Error-Correcting Codes /F. MacWilliams, N. Sloane. – North Holland, 1977. –762p.
5. Vera Pless (1982), Introduction to the Theory of Error-Correcting Codes, John Wiley & Sons, Inc.
6. Тесленко О.К., Шаповал І.І. Нероздільні блочні коригуючі коди. // Системний аналіз та інформаційні технології: матеріали 18-ї Міжнародної науково-технічної конференції SAIT 2016, Київ, 30 травня – 2 червня 2016 р. / ННК “ІПСА” НТУУ “КПІ”. – К.: ННК “ІПСА” НТУУ “КПІ”, 2016. С. 292-293.
7. Bron, Coen; Kerbosch, Joep (1973), “Algorithm 457: finding all cliques of an undirected graph”, Commun. ACM. С. 575–577.
8. Johnston, H. C. (1976), “Cliques of a graph—variations on the Bron–Kerbosch algorithm”, International Journal of Parallel Programming: С. 209–238.
9. Майданюк В. П. Кодування та захист інформації. Навчальний посібник. - Вінниця: ВНТУ, 2009. - 164 с.
10. Randy Yates (19–Aug–2009), A Coding Theory Tutorial.
11. Електронні системи: навчальний посібник / Й. Й. Білинський, К. В. Огороднік, М. Й. Юкиш. — Вінниця : ВНТУ, 2011. — 208 с.
12. MacKay, David J. C.. Information Theory, Inference, and Learning Algorithms Cambridge: Cambridge University Press, 2003.

13. Summary of Virtex-6 FPGA Features. Virtex-6 Family Overview. XILINX DS150 (v2.5) August 20, 2015.