Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

БАКАЛАВРСЬКА ДИПЛОМНА РОБОТА

на тему:

Програмно апаратні засоби модифікації циклічних кодів

ПОЯСНЮВАЛЬНА ЗАПИСКА

Виконав: студент 4 курсу, групи 2КІ-18Б
Спеціальності 123 — комп'ютерна інженерія
Мартинов П.Г.
Керівник к.т.н., проф. каф. ОТ:
Семеренко В.П.
Рецензент к.т.н., доц. каф. МБІС:
Карпінець В.В.

Допущено до захисту
д.т.н., проф. Азаров О.Д.
" 21 " червня 2022 р.

Вінниця ВНТУ — 2022 року

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

Освітньо-кваліфікаційний рівень бакалавр

Спеціальність 123 — «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

Завідувач кафедри

обчислювальної техніки

проф. Азарову О.Д.

«07» 02 2022 р.

З А В Д А Н Н Я

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Мартинову Павлу Геннадійовичу

1 Тема проекту «Програмно-апаратні засоби модифікації циклічних кодів», керівник роботи к.т.н., доц. каф. ОТ Семеренко В.П затверджена наказом вищого навчального закладу від «24» березня 2022 року №66

2 Строк подання студентом проекту 21.06.2022 р.

3 Вихідні дані до проекту: тип завадостійких кодів - циклічні коди; операція модифікації: перфорація,вкорочення;мова програмування - java.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): вступ, аналіз предметної області, операції кодування та декодування.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): технічне завдання.

6 Консультанти розділів роботи приведений в таблиці 1

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Спеціальна частина	Семеренко В.П доцент кафедри ОТ		

7 Дата видачі завдання «24» березня 2022 р.

8 Календарний план виконання БДР приведений в таблиці 2.

Таблиця 2 — Календарний план

№ з/п	Назва етапів виконання комплексної бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі роботи	08.03.22	виконано
2	Характеристика предметної області	17.03-01.04.22	виконано
3	Аналіз методів модифікації циклічних кодів	03.04-09.04.22	виконано
4	Модифікація циклічних кодів на основі лінійних автоматів	12.04-29.04.22	виконано
5	Програмно-апаратна реалізація методів модифікації кодів	01.05-18.05.22	виконано
8	Аналіз виконання роботи, висновки, додатки	21.05-25.05.22	виконано
9	Перевірка якості виконання бакалаврського проекту та усунення недоліків	28.05-09.06.22	виконано

Студент Мартинов П.Г.

Керівник роботи Семеренко В.П.

АНОТАЦІЯ

Мартинов П.Г. Програмно-апаратні засоби модифікації циклічних кодів. Бакалаврська кваліфікаційна робота зі спеціальності 123 — Комп'ютерна Інженерія, Вінниця: ВНТУ, 2022. Пояснювальна записка містить 67 сторінки, 25 рисунків та 15 посилань.

Метою цієї дипломної роботи є перфорація цикличного коду, тобто в канал повинні передаватись всі інформаційні символи і частина перевірочних символів. В результаті по каналу швидше передаються кодові дані, але на боці приймача ускладнюється задача перевірки правильності передачі даних. Далі розглянуті можливості застосування виколювання біт у циклічних кодах.

Ключові слова: завадостійке кодування, циклічні коди, модифікація параметрів коду, перфорація циклічних кодів.

ABSTRACT

Martynov P.G. Hardware and program modification of cyclic codes. Bachelor's degree in specialty 123 — Computer Engineering, Vinnytsia: VNTU, 2022. The explanatory note contains 67 pages, 16 figures and 15 references.

The purpose of this thesis is the perforation of the cyclic code, all information symbols and part of the test symbols must be transmitted to the channel. As a result, the code data is transmitted faster on the channel, but on the receiver's side the task of checking the correctness of data transmission is complicated. The possibilities of using bit puncturing in cyclic codes are given below.

Keywords: noise-tolerant coding, cyclic codes, modified code, perforation of cyclic codes.

ЗМІСТ

ВСТУП

Ця робота є актуальною в сучасних телекомуційних системах, де інформація як правило, передається у цифровому вигляді і обробляється цифровими методами. Дуже важливу роль в цьому грають саме методи кодування інформації.

Ефективність електронного зв'язку залежить не тільки від швидкості передачі, а і від достовірності переданої інформації, яка при передачі по каналам цифрового зв'язку оцінюється здатністю сигналу протистояти дії завад в каналі, тобто завадостійкістю.

При передачі інформації по каналу, який знаходиться під впливом завад, початкове повідомлення *a*, яке представляє собою послідовність нулів та одиниць, зазнає змін і на виході вже представляє зовсім іншу послідовність b. Тоді постає питання, як же за допомогою отриманого повідомлення b можна отримати початкове повідомлення *a*. Для вирішення даної проблеми було запропоновано кодувати повідомлення a більшим довгим кодовим словом x и передавати його через канал зв'язку замість *a*. Тоді вже на виході каналу отримуємо двійкове повідомлення y, за допомогою якого отримуємо кодове слово x, а потім вже і початкове повідомлення *a*. Це вдається зробити за допомогою введення надлишкових символів при кодуванні(при переході від *a* до *x*). Кодування, яке дозволяє виправляти помилки в каналі і називають завадостійким[4].

Під час вибору завадостійкого коду необхідно визначити його параметри в залежності від елементів ланцюга передачі даних. Найбільш динамічно змініючимись параметрами в подібних системах є параметри радіоканалу. Їх значення або не відомі, або часто міняються на достатньо коротких інтервалах часу.

В подібних ситуаціях при наявності достатньо досконалих процесорів передавач і отримувача вибір параметрів коду на основі деякої множини середніх показників каналу зв'язку часто виявляється не продуктивним. Одним із шляхів вирішення даної проблеми є використання систем адаптивного

кодування – автоматична і ціле направлена корекція параметрів кодів в залежності від зміни якості каналу зв’язку.

Об’єктом досліджень є операція модифікації циклічних кодів на основі лінійних автоматів.

Предметом досліджень є завадостійке кодування на основі циклічних кодів.

Метою роботи є збільшення швидкості передачі даних завдяки модифікації параметрів завадостійкого коду.

Для досягнення мети необхідно виконати наступні задачі:

— провести аналіз існуючих завадостійких кодів;

— провести аналіз існуючих методів модифікації параметрів циклічних кодів;

— розробити метод перфорації циклічних кодів на основі ЛПС;

— розробити метод розв’язання систем лінійних рівнянь в полях Галуа.

Завдяки зменшення часу передачі даних дана робота може знайти практичне застосування у радіо , мобільному та супутниковому зв’язку, цифровому телебаченні.

Методи дослідження, що були використані для вирішення поставленої задачі: теорія завадостійкого кодування, теорія інформації, теорія автоматів, теорія алгоритмів, цифрова схемотехніка.

1 АНАЛІЗ МЕТОДІВ МОДИФІКАЦІЇ ЦИКЛІЧНИХ КОДІВ

1.1 Спрощена схема передачі інформації

Вихідний сигнал блока джерела інформації поступає на вхід блока кодування джерела, задача якого полягає в перетворенні вхідного сигналу у двійкову послідовність k (інформаційне слово). Блок завадостійкого кодування перетворює інформаційне слово в кодову послідовність(кодове слово) n. В більшості випадків кодове слово n є також двійковим і містить надлишковість, необхідну для виявлення та виправлення помилок. Мірою надлишковості(завадостійкості коду) є швидкість коду R, яка обчислюється як відношення інформаційного слова до кодового слова $(R = k/n)$[1].

Вихідний сигнал блоку завадостійкого кодування, як правило, непридатний до прямої передачі інформації по фізичному каналу. Для перетворення кодової послідовності в аналоговий сигнал застосовують модулятор.

Модулятор породжує множину неперервних сигналів безкінечної тривалості та реалізує перетворення вихідної цифрової форми в множину сигналів[2]. Демодулятор на основі спостереження прийнятого сигналу оцінює який із ймовірних символів було отримано. Вірогідність правильної оцінки отриманого символу залежить від відношення потужності сигналу до потужності завад в полосі частот, від спотворення сигналу і від використовуємої схеми демодулятора

Сигнал з виходу модулятора поступає в реальний канал передачі даних.

Фізичний канал — це вся апаратура та вся фізична середа, через яку проходить сигнал на шляху від модулятора до демодулятора. Фізичний канал не обов’язково представляє з себе систему зв’язку в реальному часі, це може бути система зберігання або запису даних. Спотворення сигналу може відбуватись через сильну фільтрацію або ж наявності кількох шляхів розповсюдження сигналу. Завади можуть викликати завмирання сигналу і призводити до змінення амплітуди на виході, що еквівалентно зміні в часі параметрів самого каналу.

Далі прийнятий сигнал поступає на блок демодулятора, який перетворює сигнал в цифрову форму, придатну для декодування. Частину шляху передачі від входу модулятора до виходу демодулятора часто називають дискретним каналом, оскільки входом і виходом каналу є символи кінцевого алфавіту.

Декодер каналу спочатку оцінює прийняте кодове слово. Якщо в ньому відсутні спотворення, тоді в ньому відразу виділяється кодове слово джерела. В протилежному випадку декодер каналу, використовуючи надлишковість кодового слова каналу, виправляє наявні помилки і видає виправлене кодове слово джерела. Декодер джерела виконує операцію, яка є оберненою операції кодера джерела, результат якої вже поступає до користувача. Схему передачі даних наведено на рис.1 [14].

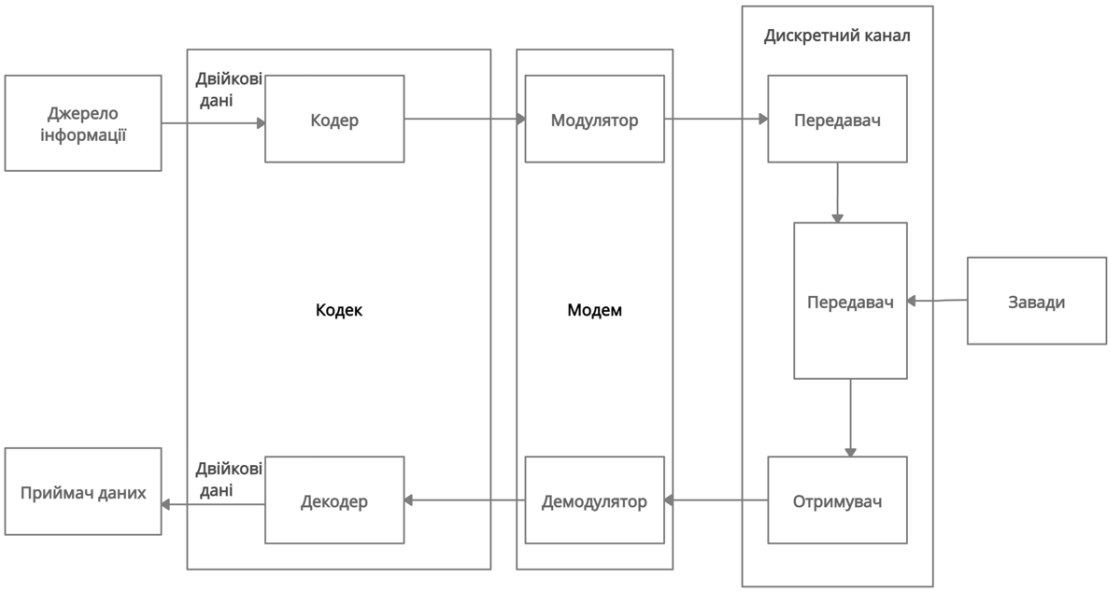


Рисунок 1 — Схема передачі даних

1.2 Класифікація завадостійких кодів

В теорії і техніці завадостійкого кодування наразі відомо багато завадостійких кодів.

За способом формування коди поділяються на блокові та неперервні[5]. Формування блокових кодів передбачає розбиття передаваємої цифрової послідовності на окремі блоки, які подаються на вхід кодера. Кожному такому

блоку відповідає блок кодових слів. Формування неперервних кодів відбувається неперервно в часі, без розбиття на блоки. Додаткові елементи розміщуються в певному порядку з інформаційними.

До неперервних належать: рекурентні, ланцюгові, згорткові коди. Особливо виділяють згорткові коди, які за характеристиками перевершують блокові коди.

Блокові коди діляться на подільні та неподільні. В подільних кодах елементи інформаційного слова і кодового слова завжди стоять на перших місцях. В неподільних кодах чіткий розподіл розташування перевірочного та інформаційного слова відсутній.

Подільні блокові коди поділяють на лінійні коди та нелінійні. В систематичному блоковому коді окремо присутні інформаційні символи, окремо кодові слова. В несистематичних-символи повідомлення в явному виді не присутні.

До лінійних кодів відносять: циклічні, турбо-коди.

До циклічних відносять: Хеммінга, Голея, Ріда-Соломона, БЧХ коди.

Для опису процедур кодування та декодування як блокових, так і згорткових кодів використовують апарат лінійної алгебри. Формування нелінійних кодів відбувається за допомогою нелінійних процедур. Схематичне представлення завадостійких кодів наведено на рис.2 [8]

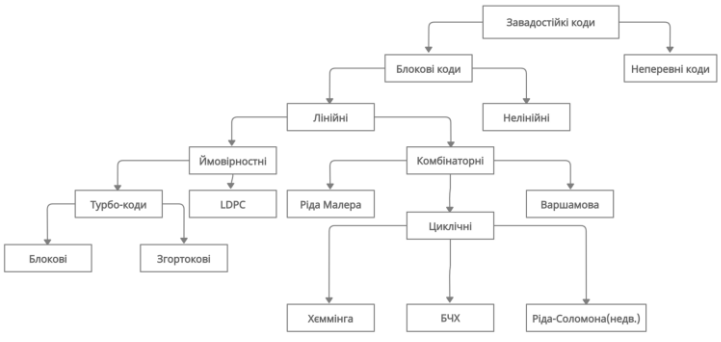


Рисунок 2 — Завадостійкі коди

1.3 Характеристика циклічних кодів

1.3.1 Загальні положення.

Циклічні коди відносяться до систематичних кодів і відповідно мають всі їх властивості і ряд додаткових.

До основних властивостей циклічних кодів відносять[2]:

— вага $W_{\text{к.п}}$ дозволеної кодової послідовності $\geq d_0$;

— вага перевірочної частини $W_{\text{п.ч}}$ кодової послідовності $\geq d_0$-1;

— зсув кодових символів дозволеної кодової послідовності вліво або вправо на один, два,… ,(k-1) символ в результаті дає дозволену кодову послідовність;

— дозволена кодова послідовність без помилок $F_p`(x)$ при діленні на поліном *P(x)* дає нульовий залишок, тобто $F_p`\frac{x}{P(x)} = R(x) = 0$ і $R(x)$ не дорівнює 0 -при присутності помилок;

— сума по модулю два символів двох, трьох,…,*(k-1)* дозволених кодових послідовностей знову дає дозволену кодову послідовність;

— двочлен $x^n + 1$ повинен ділитись на породжувальний поліном P(x) без залишку;

— якщо всі операції над поліномами (кодовими послідовностями) відбуваються в кодовому полі Галуа*(GF(2))*, тобто дії на коефіцієнтами поліномів виконується за модулем два, а множення поліномів виконується за модулем породжувального полінома P(x), то використання вказаних операцій не призводить до кодових послідовностей, довжина яких більше довжини заданого коду (n);

— результат ділення двочлена $x^n + 1$ на породжувальний поліном *P(x)* в результаті дає перевірочний поліном і позначається як $h(x) = (x^n + 1)/P(x)$. Добуток $h(x)P(x) = x^n + 1 = 1$, тому поліноми h(x) і P(x) оцінюються як ортогональні і операції ділення $(x^n + 1)/P(x)$ використовуються в основі побудови алгоритмів декодування;

— двочлен циклічного коду $(x^n + 1)$ можно розкласти на множники $x^n + 1 = (x - 1)(x^{n-1} + x^{n-2} + \cdots + 1)$, які можна використовувати в ролі породжувальних поліномів циклічного коду з *n=const*, k=var і $d_0 = var$.

Циклічні коди були створені для спрощення схем кодування і декодування. Їх ефективність в знаходженні та виправленні помилок забезпечила їм широке застосування.

Циклічні коди прийнято розглядати, представляючи комбінацію двійкового коду у вигляді полінома деякої степені[2]:

$$F(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 + a_0 \quad (1.0)$$

де x — основа системи числення;

a_i — відповідні числа системи числення (в двійковій системі 0 та 1).

Наприклад, двійкова послідовність 01001 може записана у вигляді полінома від змінної x:

$$F(x) = 0x^4 + 1x^3 + 0x^2 + 0x^1 + 10x^0 = x^3 + 1 \quad (1.1)$$

Представлення кодових комбінацій в такій форму дозволяє звести операції над комбінаціями двійкових чисел до дій над поліномами. При цьому операція додавання двійкових чисел зводиться до додавання за модулем два. Множення відбувається за звичайним правилом множення функцій, однак отримані коефіцієнти при одній степені сумуються за модулем два. Ділення відбувається за звичайним правилом ділення степеневих функцій, при цьому операція віднімання заміняється на додавання за модулем два.

Основна властивість циклічних кодів заключається в наступному. Якщо комбінація $a_0, a_1, a_2, \dots, a_{n-1}$ є дозволеною,то комбінація, отримана з неї через перестановку розрядів, тобто комбінація $a_{n-1}, a_0, a_1, \dots, a_{n-2}$ також належить цьому коду[6].

Представляти код в поліноміальній формі зручно ще й тому, що циклічна перестановка є результатом простого множення полінома на x. Якщо одна з комбінацій представлена поліномом $V(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}$, то нова комбінація за рахунок циклічного зсуву буде мати вигляд $xV(x) = a_0x + a_1x^2 + a_2x^3 + \cdots + a_{n-1}x^n$. В останньому члені необхідно замінити x^n на 1 (якщо цього не зробити, то довжина кодової комбінації буде більше n). Відповідно, отримана комбінація $V'^{(x)} = xV(x) = a_{n-1} + a_0x + a_1x^2 + a_2x^3 + \cdots + a_{n-2}x^{n-1}$ є циклічним зсувом комбінації $V(x)$.

Побудова циклічних кодів базується на використанні незвідних поліномів.[3] Незвідним називається такий поліном, який не може бути представлений у вигляді добутку поліномів менших степеней, тобто такий поліном ділиться тільки на себе або одиницю. На такий поліном ділиться без частки двочлен $x^n + 1$.

Незвідні поліноми в теорії циклічних кодів грають роль породжувальних поліномів. Для розуміння побудови циклічного коду, множимо комбінації. Простого k-значного коду $Q(x)$ на одночлен x^r, після чого ділимо на породжувальним поліном $P(x)$, степінь якого дорівнює r. В результаті множення $Q(x)$ на x^r степінь кожного одночлена в $Q(x)$ збільшується на r. При ділені добутку $x^rQ(x)$ на породжувальний поліном отримуємо частку $C(x)$ такої ж степені, як і $Q(x)$. Результат множення і ділення можна представити, як:

$$\frac{Q(x)x^r}{P(x)} = C(x) + \frac{R(x)}{P(x)} \tag{1.2}$$

де $R(x)$ – залишок від ділення $Q(x)x^r$ на $P(x)$

Частка $C(x)$ має таку ж саму степінь,як і кодова комбінація $Q(x)$ простого коду, тому $C(x)$ є кодовою комбінацією цього простого k-значного коду. Степінь остачі не може дорівнювати $r - 1$. Відповідно, найбільше число розрядів остачі $R(x)$ не перевищує r.

Перемноживши обидві частини рівняння і виконавши перестановки отримуємо:

$$F(x) = C(x)P(x) = Q(x)x^r + R(x). \quad (1.3)$$

Знак мінус було замінено на плюс, так як віднімання виконується по модулю два.

Таким чином бачимо, що кодова комбінація циклічного n-значного коду може бути отримана двома способами[4]:

— множення кодової комбінації $Q(x)$ простого кода на одночлен x^r з додаванням залишку $R(x)$, отриманого в результаті ділення добутку $Q(x)x^r$ на породжувальний поліном $P(x)$;

— множення кодової комбінації $C(x)$ простого k-значного коду на породжувальний поліном $P(x)$;

При побудові циклічних кодів першим способом положення інформаційних символів у всіх комбінаціях строго визначено — вони займають k старших розрядів кодової комбінації, а інші $(n\text{-}k)$ розрядів призначені для перевірочних(кодових).

При використанні другого способу інформаційні та контрольні символи в комбінаціях циклічного коду не розділені один від одного, що робить процес декодування складнішим.

1.4 Основні параметри кодів

Нижче наведені основі характеристики кодів [9]:

Основа коду (модуль) — кількість різних одиничних символів, використовуємих при створенні коду та кодових послідовностей.

Основа коду частіше всього позначається через q:

— якщо $q=2$, тобто використовуються символи 1 та 0,то такі коді називаються двійковими і кодування відбувається у двійковому полі Галуа, тобто $GF(q) = GF(2)$;

— якщо ж $q>2$, то такі коди називають недвійковими і кодування відбувається в недвійковому полі Галуа $GF(q^m) = GF(2^m)$.

При використанні в процесі кодування електричних імпульсів значення й відображає число різних градацій їх амплітуди, фази, частоти або інших ознак.

В системах зв'язку двійкова система числення часто використовується через відносно просту апаратну реалізацію логічних операцій та арифметичних дій. Перетворення повідомлення в сигнал при двопозиційних кодах відбувається за допомогою простих електронних схем-тригерів, які можуть перебувати тільки в двох станах: 0 та 1.

Довжина кодової послідовності (n) — називається розрядністю коду і дорівнює кількості двійкових символів (біт) в кодовій послідовності. Довжину кодової послідовності прийнято позначати через n: мінімальна довжина кодової послідовності $n = 2$(нижня межа); верхня межа може досягати тисяч та десятків тисяч двійкових символів. Код називається рівномірним, якщо всі кодові комбінації однакові за довжиною$(n = const)$, і неравномірним, якщо величина n в коді непостійна $(n = var)$.

Кількість інформаційних символів k — кількість символів в інформаційному слові.

Швидкість передачі коду R – кількість надлишкових символів на один інформаційний символ. Чим більше R, тим ефективніший завадостійкий код. R завжди менше 0 і обраховується за формулою $R = n/k$

Число кодових комбінацій (N) – совокупність кодових комбінацій називається кодовим словником. Значення N для коду з основою m та числом елементів n задається наступним виразом:

$$N = m^n \tag{1.4}$$

Вага кодової структури (w) – кількість ненульових символів в кодовій послідовності. Якщо задані такі параметри коді, як $n\ k$,то можна визначити:

— $К_{заг} = 2^n$— загальну кількість кодових послідовностей;

— $К_{доз} = 2^k$ — кількість дозволених кодових послідовностей;

— $К_{заб} = 2^l$ — кількість заборонених кодових послідовностей.

Надлишковість коду поділяється на:

— абсолютна надлишковість двійкових символів $l = (n - k)$;

— відносну надлишковість $r = \frac{(n-k)}{n} = \left(\frac{l}{n}\right) * 100\%$.

Коди, які мають більшу надлишковість, відповідно, мають і більшу завадостійкість. Збільшення надлишковості призводить до зменшення пропускної можливості каналу, через збільшену кількість передаваємих елементів.

Кодова відстань (d_{min}) — дорівнює кількості позицій, якими відрізняються дві кодові послідовності. Для двійкового коду d_{min} визначається кількістю ненульових розрядів при додавання по модулю 2 двох кодових комбінацій. Мінімальна кодова відстань — це найменше число не співпадаючих розрядів.

Кратність помилок g – число позицій кодової комбінації, на яких під впливом завад одні символи були замінені на інші.

Умова знаходження всіх помилок кратністю g :

$$g_{он} \leq d_{min}) \text{ -1} \qquad (1.5)$$

Для того, щоб мати можливість виправити всі помилки кратності п необхідно мати d_{min}, яке задовольняє умову:

$$d_{min} \geq 2g_{випр} + 1 \qquad (1.6)$$

1.5 Необхідність модифікації завадостійких кодів

Модифікація коду – це ціле направлена зміна параметрів коду. До параметрів коду, які можуть бути використані в адаптації відносять кількість інформаційних та надлишкових розрядів в кодовій комбінації. Адаптивне кодування збільшує завадостійкість або швидкість передачі даних за рахунок розподілення надлишковості коду між каналами зв'язку.

Розрізняють шість основних модифікацій коду[7].

Розширення коду – це збільшення кодової послідовності за рахунок збільшення кількості перевірочних символів, без зміни довжини інформаційних символів, що призводить до збільшення рангу породжувальної матриці $G(x)$. Ранг матриці $G(x)$ збільшується за рахунок збільшення кількості стовбців $G\ k,n(x)\ =>\ G\ k,n+1(x)$.

Додаткові перевірочні символи вибирають так, щоб вони покращували вагу кодової структури. Зазвичай вводиться загальна перевірка на парність. Значення відповідного перевірочного символу дорівнює залишку від ділення першого кодового слова на поліном $P(x) = (x-1)$. При додаванні загальної перевірки на парність вага W кожної кодової послідовності становиться парною і d_0(кодова відстань) збільшується на одиницю.

Перфорація – зменшення довжини кодовою послідовності за рахунок зменшення кількості перевірочних символів. В результаті це дає лінійний блоковий код з параметрами $(n-f,k,d`)$, у якого $d` < d$. Швидкість коду зростає, оскільки надлишковість(кількість перевірок) зменшується. Нехай $G = (l_{k*k}|P_{k*(n-k)})$, тоді $H = (P_0^T P_1^T \dots P_{k-1}^T|l_{n-k})$ і перфорація будь-якого стовбця із матриці l_{n-k} призводить к видаленню строки з таким ж номером. Техніка перфорації перевірочних розрядів досить ефективна в системах з параметричною адаптацією по коду.

Це процес, передбачаючий зменшення числа інформаційних символів без зміни довжини коду, тобто $n = const$, що призводить до зменшення кількості строк в породжувальній матриці *G k-I,n(x)* і відповідно до зменшення деяких кодових комбінацій.

Процедура поповнення коду за рахунок збільшення кількості інформаційних символів без збільшення довжини коду, що призводить до збільшення меншого розряду породжувальної матриці $G\ k, n(x) => G\ k + i, n(x)$. Код в такому випадку поповняється новими кодовими комбінаціями.

Збільшення довжини кодової послідовності відбувається за рахунок додавання нових інформаційних символів, що призводить до збільшення обох розмірів породжувальної матриці $G\ k, n(x) => G\ k + 1, n + 1(x)$. Якщо почати з циклічного коду, для якого $P(x)$ ділиться на $(x - 1)$, то процедура ділиться на два етапа. Спочатку код поповнють до кода, отриманого поліномом $p(x)/(x - 1)$, що відповідає додаванню до кода слова(послідовності) цілком одиниць. Далі отриманий код необхідно розширити, додавши перевірку на парність.

Процедура виконується за рахунок зменшення кількості інформаційних символів, що призводить до зменшення обох розмірів породжувальної матриці $G\ k, n(x) => G\ k - l, n - l(x)$.

Таблиця 1 — Модифікація коду

Операція	Дія	Параметри
Вкорочення	Вилучення інф. та перев. символів	n-p, k-p
Поповнення	Додавання інф. та перев. символів	n+p, k+p
Розширення	Додавання перев. символів	n, k+p
Перфорація	Вилучення перев. символів	n, k-p
Доповнення	Додавання кодових символів	n+p, k
Викидання	Вилучення код. симв.	n-p, k

2 МОДИФІКАЦИЯ ЦИЛІЧНИХ КОДІВ НА ОСНОВІ ТЕОРІЇ ЛІНІЙНИХ АВТОМАТІВ

2.1 Абстрактна модель автоматів

Цифрові пристрої з m тригерами ($m > 1$), стан виходів яких залежить не тільки від значень вхідних сигналів у даний момент часу, а й від стану використовуваних тригерів у даний та попередні моменти часу, називаються цифровими автоматами.

У загальному плані цифровим автоматом називається пристрій, який формує ряд вихідних дискретних сигналів у відповідності до вхідної бінарної комбінації сигналів. Найпростішим з таких автоматів є перетворювач кодів, і він називається комбінаційним.

Оскільки кількість внутрішніх станів залежить від кількості використовуваних тригерів, то цифрові автомати поділяються на скінченні, що мають обмежену кількість станів, і автомати з нескінченною кількістю станів, або послідовнісні машини.

Будь-який цифровий автомат є сукупністю елементів пам’яті (тригерів) та комбінаційних схем. Тригери містять інформацію про особливості попередньої роботи автомата. Комбінаційні схеми на основі вхідних сигналів та інформації, яка береться з тригерів, формують вихідні сигнали і сигнали для формування нових станів тригерів. Таким чином, однією з особливостей цифрових автоматів є те, що вони мають свої внутрішні стани, від яких залежить реакція на вхідні сигнали.

Автомати можуть бути синхронними, зміна станів яких відбувається в тактові моменти часу, що задаються зовнішнім генератором, а також асинхронними — зміна станів яких відбувається внаслідок дії вхідних сигналів практично без затримки. Асинхронні автомати вважаються більш швидкодіючими і знаходять використання у швидкодіючих інформаційних пристроях вимірювання і керування різноманітними процесами, де необхідна миттєва реакція на зміну вхідних сигналів.

Синхронні автомати в силу специфіки своєї роботи вносять у процес вимірювання або керування затримку, що визначається величиною періоду синхросигналу.

Здебільшого асинхронні автомати будуються на основі асинхронних елементів пам'яті – асинхронних тригерів. Синхронні автомати будуються з використанням синхронних тригерів.

Математичною моделлю цифрового автомата є абстрактний автомат, в якому враховуються вхідні та вихідні сигнали, а також внутрішні стани. Ми будемо розглядати детерміновані автомати, які завжди мають початковий стан Q_0, з якого вони починають працювати при дії вхідних сигналів і при повторі перебору вхідних сигналів повторюють послідовність станів і вихідних сигналів.

Абстрактний автомат задається множиною внутрішніх станів

$$Q = \{Q_1, Q_2, Q_3, \ldots, Q_m\} \text{ (алфавітом станів)}, \quad (1.7)$$

множиною вхідних сигналів

$$X = \{X_1, X_2, X_3, \ldots, X_p\} \text{ (вхідним алфавітом)}, \quad (1.8)$$

множиною вихідних сигналів

$$Y = \{Y_1, Y_2, Y_3, \ldots, Y_k\} \text{ (вихідним алфавітом)} \quad (1.9)$$

і початковим станом Q_0.

Перехід з одного стану в інший визначається функцією переходів f_p , що визначає стан автомата Q_s, в який він переходить з попереднього стану Q_m при дії сигналу X_p :

$$Q_s = f_p(Q_m, X_p) \quad (2.0)$$

Значення виходів автомата задається функцією виходів λ, що залежить від стану автомата Q_m і вхідного сигналу X_p :

$$Y_k = \lambda(Q_m, X_p) \tag{2.1}$$

Абстрактний автомат працює в дискретному часі, який задається цілими позитивними числами $t = 0, 1, 2, \ldots = n$.

У кожний момент дискретного часу t, який зветься тактом, автомат перебуває у деякому стані $Q(t)$ з множини станів автомата Q.

У початковий момент часу ($t = 0$) він завжди знаходиться в стані $Q(0) = Q_0$. Вважається, що реакція автомата на вхідні сигнали не залежить від інтервалів часу між тактовими моментами.

У момент t, знаходячись у стані $Q(t)$, автомат сприймає на своєму вході сигнали, які називаються буквами (літерами) вхідного алфавіту $X_p \in X$. У відповідності до функції переходів f_p, він перейде у стан $Q_s(t)$, що описується алфавітом станів, тобто $Q_s(t) \in Q$.

Аналогічно, у відповідності до функції виходів λ, на виході отримується сигнал Y_k, де $Y_k \in Y$.

Скінченна множина букв вхідного алфавіту, вихідного алфавіту і станів називається словами. Для скінченного автомату кількість символів цих алфавітів обмежена. Повний перебір символів вхідного слова повинен привести автомат до початкового стану Q_0. Автомат, який завжди починає працювати з початкового стану, називається ініціальним.

Поняття станів в описі автоматів пов’язане з необхідністю враховувати типи і характер попередніх сигналів, тобто таких, які діяли на один або декілька тактів раніше. Стани автомата і є тими відповідними елементами пам’яті, що характеризують попередні сигнали. Введення поняття станів дозволяє усунути час як явну змінну і визначати вихідний сигнал як функцію внутрішнього стану і входу в тактовий момент часу. Такий спосіб опису автоматів має великі переваги перед іншими для асинхронних пристроїв, в яких інтервали часу між

сусідніми тактами можуть мати довільні значення, що значно відрізняються між собою. З такої точки зору комбінаційні пристрої відносяться до автоматів, в яких вихід не залежить від попередніх сигналів і повністю визначається комбінацією вхідних сигналів.

2.2 Представлення циклічних кодів на основі лінійних автоматів

Відомі різні способи представлення циклічних кодів: поліноміальне, матричне, через корінь породжувального багаточлена [11]. Зручним способом представлення циклічних кодів є їхнє представлення за допомогою математичного апарату лінійних послідовних схем (ЛПС) [12].

Лінійна послідовнісна схема Λ — це кінцевий автомат лінійного типу(лінійний автомат), який над полем Галуа $GF(q)$ описується функцією станів (переходів)

$$S(t + 1) = A \times S(t) + B \times U(t), GF(q) \quad (2.2)$$

та функцією виходів

$$Y(t) = C \times S(t) + D \times U(t), GF(q), \quad (2.3)$$

де t — дискретний час;

A, B, C, D — характеристичні матриці ЛПС;

S, U, Y — вектори станів, вхідного та вихідного слова.

Вибір характеристичних матриць ЛПС визначається вимогою r— керованості ЛПС, тобто можливості переходу з будь-якого стану S_i у стан S_j не більше ніж за r тактів роботи автомата.

Зазвичай застосовуються два різновиди ЛПС. Перший різновид ЛПС описується характеристичними матрицями виду:

$$A_1=\begin{vmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 \\ g_0 & g_1 & g_2 & \dots & g_{r-1} \end{vmatrix}, B = \begin{vmatrix} 0 \\ 0 \\ \dots \\ 0 \\ 1 \end{vmatrix}, C = |1 \quad 0 \quad 0 \quad \dots \quad 0|, D = [0].$$

Другий різновид ЛПС описується характеристичними матрицями виду:

$$A_2=\begin{vmatrix} 0 & 0 & 0 & \dots & g_0 \\ 1 & 0 & 0 & \dots & g_1 \\ 0 & 1 & 0 & \dots & g_2 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & g_{r-1} \end{vmatrix}, B = \begin{vmatrix} 1 \\ 0 \\ 0 \\ \dots \\ 0 \end{vmatrix}, C = |0 \quad 0 \quad \dots \quad 0 \quad 1|, D = [0].$$

Матрицю A_1 першого виду називають також супроводжуючою або матрицею Фібоначчі, а матрицю A_2 другого виду —транспонованою супроводжуючою або матрицею Галуа.

Елементи останнього рядка матриці A_1 з та останнього стовпця матриці A_2 з являють собою коефіцієнти породжувального багаточлена циклічного коду Ω:

$$g(x) = g_0 + g_1 x + g_2 x^2 + \cdots + g_{r-1} x^{x-1} + g_r x^x. \qquad (2.4)$$

Розмірності матриць ЛПС і параметри циклічного коду Ω пов'язані через коефіцієнт r, який для коду дорівнює числу контрольних розрядів кодового вектора $C(x)$ при систематичному кодуванні ($r = n - k$).

При апаратній реалізації ЛПС над полем Галуа GF(2) відповідає регістр зсуву з лінійними зворотними зв'язками (РСЛЗЗ) з l входами, m виходами та r елементами пам'яті.

Одним із окремих випадків ЛПС є автономна ЛПС.

Автономна ЛПС над полем Галуа GF (2) називається така ЛПС, функціонування якої не залежить від вхідних впливів та описується функцією станів (переходів)

$$S(t+)1 = A \times S(t), \quad GF(2), \tag{2.5}$$

та функцією виходів

$$Y\ t)(= C \times S\ t), \quad GF(2) \tag{2.6}$$

Характеристичні матриці B та D автономної ЛПС дорівнюють нулю. Як зазначається в [12], автономна ЛПС не перетворює додані ззовні дії, а генерує періодичну послідовність символів поля Галуа GF(2). Період r-мірної ЛПС визначається видом породжувального багаточлена: для примітивного багаточлена період буде максимальним, тобто $2^r - 1$.

2.3 Операція перфорації циклічних кодів

Операція виколювання (перфорації) символів в сучасних системах завадостійкого кодування відіграє важливу роль.

Введення надлишковості суттєво знижує швидкість коду R, що не завжди вигідно через постійно змінюючихся умов передачі інформації в каналі зв'язку. При високому рівні завад R буде доцільно збільшити, тобто в відношенні $R = k \div n$ необхідно збільшити значення n. В умовах передачі інформації з низьким рівнем завад немає необхідності в передачі всіх надлишкових символів. В свою чергу це означає, що в примітивному варіанті необхідно перейти на кодуючі і декодуючі пристрої з іншими відповідними значеннями n та k. Реалізація такого підходу підіймає питання щодо синхронізації і погодження кодів за параметром k для деяких фіксованих значень n [7].

Перфорація коду полягає в систематичному видаленні з процесу передачі в канал контрольних бітів (символів) з виходу основного кодера [13]. Кількість інформаційних символів не змінюється. В результаті формується виколотий код з більш високою кодовою швидкістю. Отримувач знає матрицю(маску)

перфорації P, яка задає правило видалення вихідних символів. На стороні отримувача виколоті символи, позиції яких відомі, відновлюються як стерті. В системі, яка використовує коди з перфорацією, процес синхронізації становиться більш стабільним, за умови, що був досягнутий необхідний рівень синхронізації по масці перфорації.

Для прикладу закодуємо інформаційне слово $I = 10010000001$ циклічним кодом Хеммінга з породжувальним поліномом $g(x) = 1 + x + x^4$ і виконаємо перфорацію останніх двох бітів.

Типова схема кодера наведена на рис.3

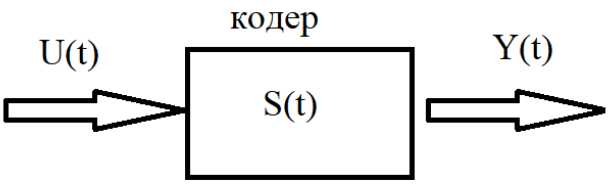


Рисунок 3 — Кодуючий пристрій

На рис. 3 U(t) — вхідний вектор, S(t)—стан автомата,Y(t)—вихідний вектор. Стани автомата обчислюються за формулою 2.7 в залежності від часу t

$$S(t-1) = A * S(t) + B * U(t) \quad (2.7)$$

$S(0) = \begin{vmatrix} 0 \\ 0 \\ 0 \\ 0 \end{vmatrix}$—початковий стан автомата.

$$A = \begin{vmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{vmatrix} \quad B = \begin{vmatrix} 1 \\ 0 \\ 0 \\ 0 \end{vmatrix}$$

Далі обраховуємо S(t) за формулою

$$S(1) = A * S(0) + B * U(1) = \begin{vmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{vmatrix} * \begin{vmatrix} 0 \\ 0 \\ 0 \\ 0 \end{vmatrix} + \begin{vmatrix} 1 \\ 0 \\ 0 \\ 0 \end{vmatrix} * |1| = \begin{vmatrix} 0 \\ 0 \\ 0 \\ 0 \end{vmatrix} + \begin{vmatrix} 1 \\ 0 \\ 0 \\ 0 \end{vmatrix} = \begin{vmatrix} 1 \\ 0 \\ 0 \\ 0 \end{vmatrix}$$

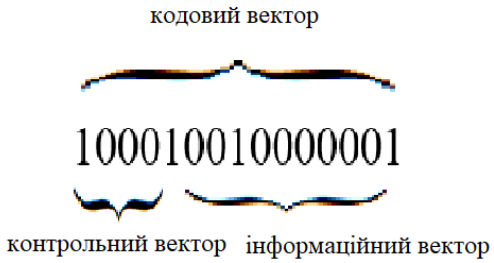
$$S(2) = \begin{vmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{vmatrix} * \begin{vmatrix} 1 \\ 0 \\ 0 \\ 0 \end{vmatrix} + \begin{vmatrix} 1 \\ 0 \\ 0 \\ 0 \end{vmatrix} * |0| = \begin{vmatrix} 0 \\ 1 \\ 0 \\ 0 \end{vmatrix} + \begin{vmatrix} 0 \\ 0 \\ 0 \\ 0 \end{vmatrix} = \begin{vmatrix} 0 \\ 1 \\ 0 \\ 0 \end{vmatrix}$$

$$S(3) = \begin{vmatrix} 0 \\ 0 \\ 1 \\ 0 \end{vmatrix} \; S(4) = \begin{vmatrix} 0 \\ 0 \\ 0 \\ 1 \end{vmatrix} \; S(5) = \begin{vmatrix} 1 \\ 1 \\ 0 \\ 0 \end{vmatrix} \; S(6) = \begin{vmatrix} 0 \\ 1 \\ 1 \\ 0 \end{vmatrix} \; S(7) = \begin{vmatrix} 0 \\ 0 \\ 1 \\ 1 \end{vmatrix} \; S(8) = \begin{vmatrix} 0 \\ 1 \\ 0 \\ 1 \end{vmatrix}$$

$$S(9) = \begin{vmatrix} 1 \\ 1 \\ 1 \\ 0 \end{vmatrix} \; S(10) = \begin{vmatrix} 0 \\ 1 \\ 1 \\ 1 \end{vmatrix} \; S(11) = \begin{vmatrix} 0 \\ 1 \\ 1 \\ 1 \end{vmatrix}$$

$$S`(12) = \begin{vmatrix} 0 \\ 0 \\ 1 \\ 1 \end{vmatrix} + \begin{vmatrix} 1 \\ 1 \\ 0 \\ 0 \end{vmatrix} = \begin{vmatrix} 1 \\ 1 \\ 1 \\ 1 \end{vmatrix} \; S`(13) = \begin{vmatrix} 0 \\ 1 \\ 1 \\ 1 \end{vmatrix} + \begin{vmatrix} 1 \\ 1 \\ 0 \\ 0 \end{vmatrix} = \begin{vmatrix} 1 \\ 0 \\ 1 \\ 1 \end{vmatrix} \; S`(14) = \begin{vmatrix} 0 \\ 1 \\ 0 \\ 1 \end{vmatrix} + \begin{vmatrix} 1 \\ 1 \\ 0 \\ 0 \end{vmatrix} = \begin{vmatrix} 1 \\ 0 \\ 0 \\ 1 \end{vmatrix}$$

$S`(15) = \begin{vmatrix} 0 \\ 1 \\ 0 \\ 0 \end{vmatrix} + \begin{vmatrix} 1 \\ 1 \\ 0 \\ 0 \end{vmatrix} = \begin{vmatrix} 1 \\ 0 \\ 0 \\ 0 \end{vmatrix}$—кодовий вектор дописуємо в кінець інф.в.



При перфорації коду виколюються останні два біти в кодовому векторі, вони будуть позначатися як стерті і не будуть передаватися в канал зв’язку.

XX0010010000001

Обчислимо синдром помилки:

$$x + 0 = x$$
$$x + 1 = \bar{x} \qquad (2.8)$$

$$S(13) = \begin{vmatrix}1\\0\\1\\1\end{vmatrix}; S(14) = \begin{vmatrix}\text{x}\\1\\0\\1\end{vmatrix} + \begin{vmatrix}1\\1\\0\\0\end{vmatrix} = \begin{vmatrix}x+1\\0\\0\\1\end{vmatrix} = \begin{vmatrix}\bar{x}\\0\\0\\1\end{vmatrix};$$

$$S(15) = \begin{vmatrix}0\\\bar{x}\\0\\0\end{vmatrix} + \begin{vmatrix}1\\1\\0\\0\end{vmatrix} + \begin{vmatrix}\text{x}\\1\\1\\1\end{vmatrix} = \begin{vmatrix}\bar{x}\\\text{x}\\0\\0\end{vmatrix};$$

Прирівнюємо до нульового контрольного синдрому:

$$\begin{vmatrix}\bar{x}\\\text{x}\\0\\0\end{vmatrix} = \begin{vmatrix}0\\0\\0\\0\end{vmatrix}$$

Синдром буде нульовим, якщо

$$\bar{x} = 0 \rightarrow x_2 = 1 \text{ розряд } 15 = 1$$
$$x = 0 \rightarrow x_1 = 0 \text{ розряд } 14 = 0$$
$$0 = 0$$
$$0 = 0$$

2.4 Операція вкорочення коду

Зазвичай вкорочення коду полягає в заповненні h—інформаційних символів нулями. В результаті виходить вкорочений циклічний $(n-h, k-h)$ — код. На стороні отримувача можна добавити h нулів у відповідні позиції і виконати декодування отриманого кодового слова звичайним способом [10].

В результаті вкорочення ми отримаємо код з такою самою мінімальною кодовою відстанню. Окрім збереження корегувальних властивостей вкорочені циклічні коди зберігають і більшість інших властивостей початкового коду, мають такі ж самі схеми кодування та декодування. Єдиною суттєвою відмінністю вкороченого коду буде те, що циклічний зсув кодової комбінації не буде завжди давати в результаті чергову дозволену кодову комбінацію. Тому вкорочені циклічні коди не є строго циклічними і їх часто називають псевдоциклічними.

Вкорочені циклічні коди зберігають основні властивості класичних кодів, до яких належать наступні:

— вкорочені циклічні коди утворюються дільниками бінома $x^n + 1$— поліномами, що породжують $G(X)$, такими ж, як у повних циклічних кодів;

— вкорочені циклічні коди відносяться до класу лінійних (групових) кодів, для яких сума дозволених кодових комбінацій коду також є дозволеною кодовою комбінацією;

— вкорочені циклічні коди має таку ж саму мінімальну кодову відстань, як і у початкового коду, і таке само число перевірочних;

— вкорочені циклічні коди виправляє таку ж саму кількість помилок, як і циклічного коду, тобто має таку ж кратність виявлених і виправлених помилок.

Специфіку побудови вкорочених циклічних кодів розглянемо на прикладі.

В цьому прикладі нам потрібно передати повідомлення, закодоване стандартним кодом МТК—2 з числом інформаційних символів $k = 5$. Необхідно забезпечити виправлення одноразової помилки в кодовому слові в одержувача повідомлення.

Одноразова помилка виправляється за мінімальної кодової відстані $d_{min} = 3$. Цьому значенню задовольняють коди Хеммінгу (7,4), (15,11), (31,26)... (див.табл.1). Код (7,4) з числом інформаційних символів $k = 4$ та числом перевірочних символів p=3 не задовольняє умові прикладу при необхідності передачі $k = 5$.

Цю умовою задовольняє наступний по порядку код Хеммінга (15,11), якщо із загальної кількості символів $n = 15$ і числа інформативних символів $k = 11$ відняти одне й те число $\ell = 6$ (виключення перших ℓ стовпців і ℓ рядків, що породжує код матриці з розмірністю $k \times n$).

Отримуємо вкорочений циклічний код (9, 5), що задовольняє умові прикладу $k = 5$, $d_{min} = 3$, з числом перевірочних символів $r = 4$.

Для опорного циклічного коду (15,11) біном $x^n + 1$ розкладається на наступні незвідні поліноми:

$$x_{15} + 1 = (x + 1) * (x_2 + x + 1) * (x_4 + x + 1) *$$

$(x_4 + x_3 + 1) * (x_4 + x_3 + x_2 + x + 1)$,з яких для побудови коду $r = 4$ можна вибрати будь-який із трьох останніх. Виберемо як породжувальний поліном $G(X) = x_4 + x + 1$ і на основі матриці цього циклічного коду(15,11) покажемо, як здійснюється відсікання(рис. 5):

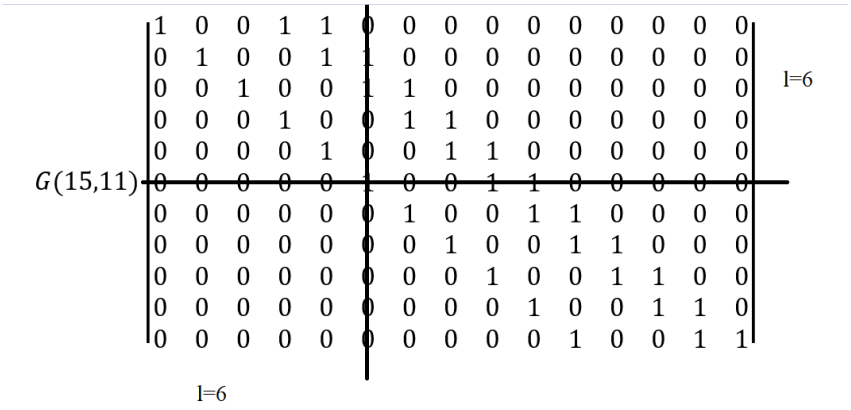


Рисунок 4 — Відсікання матриці

$$G(9,5) = \begin{vmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{vmatrix}$$ —порізана матриця (9,5)

Приведемо усічену матрицю $G(9,5)$ до канонічного виду шляхом відповідного додавання рядків і отримаємо відповідні рівняння перевірки на парність при по-елементному формуванні усіченого коду (9, 5):

$$G(9,5) = \begin{vmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{vmatrix}$$ —канонічний вигляд матриці

Рівняння перевірки на парність:

$$r_1 = i_2 \oplus i_3;$$

$$r_2 = i_1 \oplus i_3 \oplus i_4;$$

$$r_3 = i_2 \oplus i_4 \oplus i_5;$$

$$r_4 = i_1 \oplus i_2 \oplus i_5.$$

3 Програмно-апаратна реалізація методів модифікації кодів

3.1 Структурний склад лінійних послідовнісних схем

Циклічність перестановок для формування дозволених кодових комбінацій циклічного коду лежить в основі техніки побудови кодера і декодера циклічних кодів. Ця техніка застосовує зсувні регістри у вигляді тригерних ланцюжків із тими чи іншими зворотними зв'язками. Такі зсувні регістри називають також багататактними лінійними послідовнісними схемами (ЛПС) та лінійними кодовими фільтрами Хафмена, який першим розпочав вивчення ЛПС з погляду лінійних фільтрів. До речі, Д. Хафмен є і автором принципу, що полягає в тому, що "дві точки зору краще, ніж одна", який одержав широке застосування в даний час.

При побудові ЛПС використовується 3 види елементарних пристроїв [14]:

— суматор, що має, як правило, два входи та один вихід, причому для двійкових кодів додавання здійснюється за модулем 2;

— запам’ятовуючий пристрій, що має один вхід і один вихід і являє собою одну тригерну комірку (один розряд) зсувного регістра;

— пристрій множення на постійну величину, що має один вхід та один вихід. Ці пристрої зображуються на схемах, як показано на рис. 5

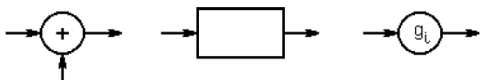


Рисунок 5 — Пристрої ЛПС

Лінійними послідовними схемами з кінцевим числом станів називаються будь-які схеми, що містять кінцеве число суматорів, пристроїв пам'яті та пристроїв множення на константу, з'єднаних будь-яким допустимим способом.

У бінарному випадку суматор являє собою логічний елемент «виключне АБО», а пристрій пам'яті є пристроєм затримки (D-тригером). Пристрої затримки, включені послідовно, становлять зсувний регістр, у комірках якого вихідний символ збігається з вхідним символом попереднього часу. До

зсувного регістра підводиться шина зсуву, з допомогою якої тактовими імпульсами здійснюється просування по розрядах зсувного регістра записаної кодової інформації. Як правило, шина зсуву не показується на схемах ЛПС.

При формуванні та обробці двійкових циклічних кодів введення в схему ЛПС помножувача на константу, що дорівнює 1, еквівалентно введенню додаткового з'єднання, а помножувач на константу, що дорівнює 0, відповідає відсутності такого з'єднання.

Передбачається, що на вхід зсувного, що входить до складу ЛПС, кодова комбінація подається послідовно, з періодичністю, що дорівнює періоду проходження тактового імпульсу в шині зсуву. Аналогічно, послідовно у часі, з'являються кодові символи на виході зсувного регістра. Коли входом або виходом є багаточлен, що представляє при двійковій обробці набір «1» і «0», то на вхідному або вихідному кінці зсувного регістра з'являються лише коефіцієнти («1» або «0»), починаючи з коефіцієнтів вищих порядків. Це обумовлюється тим, що при розподілі у дільника спочатку мають бути опрацьовані коефіцієнти вищих порядків.

3.2 Множення поліномів на основі ЛПС

Схема, зображена на рис. 5.1, використовується для множення будь-якого полінома на вході

$$A(X) = a_0 + a_1 x + a_2 x_2 + \ldots + a_k x_k \qquad (2.9)$$

на фіксований породжувальний поліном:

$$G(X) = g_0 + g_1 x + g_2 x_2 + \ldots + g_r x_r. \qquad (3.0)$$

Передбачається, що всі розряди зсувного регістра містять нулі, а на вхід коефіцієнти полінома A(X) надходять, починаючи з коефіцієнтів вищих порядків (зі старших розрядів), після чого слідує r нулів.

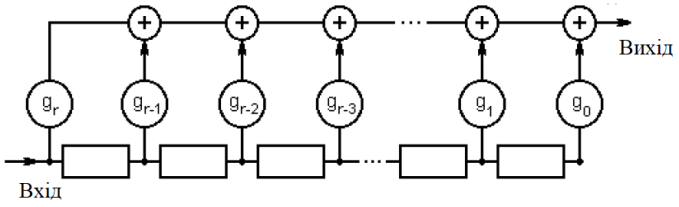


Рисунок 5.1 — ЛПС для множення поліномів

Добуток поліномів:

$$A(X) \cdot G(X) = a_0\, g_0 + (a_0\, g_1 + a_1\, g_0)x + \ldots + a_k g_r X_k + r. \qquad (3.1)$$

Коли на вході ЛПС з'являється перший (старший) коефіцієнт полінома A(X), то він помножиться в першому пристрої множення на g_r і з'явиться на виході вже як результат перемноження $a_k g_r$, прослідкувавши "транзитом" через усі схеми підсумовування за модулем 2. Крім того, a_k запишеться в першому розряді зсувного регістра, а всі інші розряди утримуватимуть нулі. Через одиницю часу, з появою в шині зсуву 2-го тактового імпульсу, на вході з'явиться a_{k-1}, який перемножиться з g_r і додається в першій схемі підсумовування за модулем 2 $a_k g_{r-1}$, сформувавши на виході суму $a_{k-1} g_r + a_k g_{r-1}$, Тобто другий коефіцієнт твору $A\,(X)\,G\,(X)$. Подальші операції здійснюються аналогічним чином. Після $r + k$ зрушень зсувного регістра повністю обнулиться і на виході з'явиться значення $a_0 g_0$, що дорівнює першому коефіцієнту множення (3.1), так що множення на виході ЛПС послідовно виходить у повному складі.

Коефіцієнти добутку формуються безпосередньо в зсувному регістрі. Після того, як перший символ подається на вхід, на виході з'являється останній коефіцієнт (3.8) $a_k\ g_r$, а розряди регістра містять лише нулі.

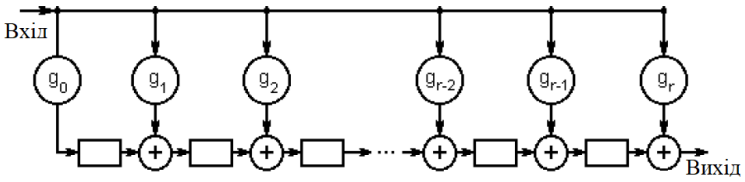


Рисунок 5.2 — Другий варіант ЛПС множення поліномів

Після одного зсуву комірки регістра містять елементи $a_k g_0, a_k\ g_1, \ldots, a_k\ g_{r-1}$, а вхід дорівнює a_{k-1}. У цьому вихід регістра дорівнює $a_k\ g_{r-1}\ +\ a_{k-1}\ g_r$, т. е. дорівнює другому коефіцієнту (3.1). Після появи чергового тактового імпульсу в шині зсуву на виході з'являється третій коефіцієнт (4.32). Подальші операції здійснюються аналогічним чином.

Схеми множення можуть мати більше одного входу, якщо додати до ЛПС, зображеної на рис. 5.2 другу шину з ланцюжком пристроїв множення, пов'язаних з відповідними схемами підсумовування за модулем 2. Тоді схема реалізовуватиме процедуру підсумовування добутку двох пар поліномів

$$C(X)\ =\ A_1(X)\cdot G_1(X)\ +\ A_2(X)\cdot G_2(X), \qquad (3.2)$$

причому запам’ятовуючий пристрій буде лише один.

3.3 Ділення поліномів на основі ЛПС

Схема для ділення полінома $A(X) = a_0 + a_1 x + a_2 x_2 + \cdots +\ a_k x_k$ на поліномі $G\ (X) =\ g_0 +\ g_1 x + g_2\ x_2 + \cdots + g_r x_r$ представлена на рис. 5.3. Динамічний запам’ятовуючий пристрій у вигляді зсувного регістра спочатку має містити всі нулі. Для поділу поліномів він охоплений зворотним зв'язком, т. е. вихід регістра з'єднується з входом. Для підкреслення протилежного напрямку шини зворотного зв'язку коефіцієнт помножувача позначається як gr_r^{-1}.

Для перших r - зсувів, тобто до тих пір, поки перший вхідний символ не досягне кінця, вихід приймає значення, рівні "0". Після цього на виході

з'являється перший ненульовий вихід, який дорівнює $a_K * g_r^{-1}$ — першому коефіцієнту частки. Для кожного коефіцієнта частки g_j необхідно відняти від діленого поліном $G(X)$.

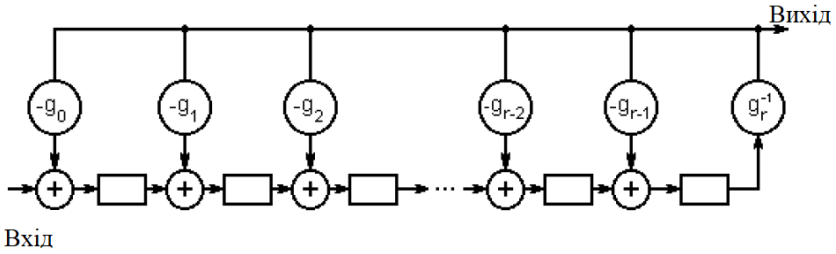


Рисунок 5.3 — Перший варіант ЛПС для ділення поліномів

Це віднімання здійснюється за допомогою зворотного зв'язку. Після k зрушень на виході з'явиться частка від поділу, а залишок від поділу буде в РЗ.

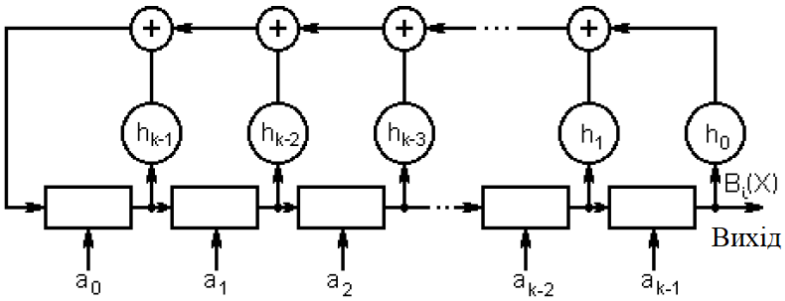


Рисунок 5.4 — Другий варіант ділення поліномів на базі ЛПС

При побудові кодера циклічного коду, а також генераторів різних кодових послідовностей, зокрема, послідовностей максимальної довжини (М-послідовностей), застосовується в ряді випадків так званий генераторний поліном H(X). Цей поліном називають також перевірним, якщо він виходить при розподілі бінома $1 + x^n$ на породжуваьлний поліном

$$G(X): H(X) = \frac{1+x^n}{G(X)} \quad (4)$$

При використанні цієї схеми в якості кодера циклічного коду вихідну кодову комбінацію A(X) паралельно одночасно записують в k розрядів зсувного регістра.

З першим тактом на вихід буде виданий коефіцієнт $b_{n-1} = a_{k-1}$, відбудеться зсув вправо в регістрі, і в комірку пам'яті, що звільнилася, буде записано обчислене значення перевірочного біта $r_{n-k-1} = h_0\, a_{k-1} + h_1\, ak_{k-2} + \ldots + h_{k-1}a_0$. На другому такті на виході буде зчитано коефіцієнт $b_{n-2} = a_{k-2}$, відбудеться зсув, і в першу комірку регістра, що звільнився, запишеться другий перевірочний біт $r_{n-k-2} = h_0\, a_{k-2} + h_1\, ak_{k-3} + \ldots + h_{k-1}r_{n-k-1}$. Через $n-k$ тактів будуть обчислені всі $n-k$ перевірочних символів $r_0, r_1, \ldots, r_{n-k-1}$ і записані в регістр. Після k тактів, тобто після виведення на вихід всіх інформаційних символів, стануть виводитися перевірочні символи в тому порядку, якому вони обчислювалися. На виході формується блоковий код. Після k тактів процес кодування однієї комбінації $A_i(X)$ закінчується, і регістр приймає початковий стан. Для кодування наступної комбінації необхідно стерти $A_i(X)$, ввести в регістр нову $A_j(X)$ та повторити цикл із n тактів.

3.4 Кодуючі та декодуючі пристрої для циклічного коду Хеммінга

Для побудови кодера за класичною схемою розподілу, оскільки кодування шляхом обчислення залишку "загалом" вимагає попереднього виконання операції множення на оператор зсуву X_r та складання полінома-залишку з поліномом-добутком $A_i(X) \cdot X_r$ (4.15), потрібно попередньо видозмінити структуру схеми. Для виконання операції множення слід розмістити суматор, на який підключений вхід наприкінці регістра перед зворотним зв'язком g_{r-1}. Таке підключення входу еквівалентно множенню на X_r, так як виключається затримка на r тактових імпульсів.

Для виконання операції складання залишку $R_i(X)$ з поліномом $A_i(X) \cdot X_r$ (4.15) необхідно вихід кодера підключити до одного з входів схеми логічного

складання (АБО), до другого входу якої підключається вхід схеми кодової комбінації $A_i(X)$ (старшим розрядом уперед).

Розглянемо роботу схеми на конкретному прикладі.

В цьому прикладі побудуємо схему кодера, що забезпечує кодування циклічного коду (n=7,k=4) з породжуючим поліномом $G(X) = 1 + X + X^3$ шляхом визначення перевірочної групи методом поділу поліномів та визначення залишку $R(X)$. Прослідкувати за тактами процес кодування та стан елементів схеми при кодуванні вихідного полінома $A_i(X) = 1 + X_3 \sim 1001$. Схема кодера для умов прикладу наведена на рис. 5.5, стан комірок та виходу схеми за тактами — у табл. 5.

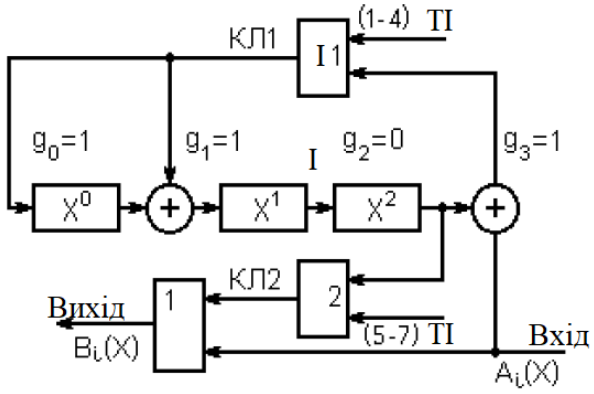


Рисунок 5.5 — Схема кодера

Поряд із вищезазначеними особливостями побудови схеми, кодера доповнено двома ключовими схемами, роль яких виконують схеми логічного множення I_1 та I_2, відповідно. Протягом перших $k = 4$ тактів другий вхід схеми I_1 надходять тактові імпульси, забезпечуючи проходження символів від вихідного суматора в шину зворотного зв'язку регістра. Починаючи з 5-го по 7-й такт, на другий вхід схеми I_1 не надходять, і зворотний зв'язок розривається. У цей час надходять тактові імпульси на другий вхід схеми I_2 завдяки чому вихід регістра підключається до виходу всього кодера, забезпечуючи видачу залишку від поділу кодової комбінації $A_i(X)$ на породжувальний поліном $G(X)$ на вихід для підстановки перевірочних символів до $A_i(X)$.

З табл.2 видно, що після 4-го такта у зсувному регстрі утворюється залишок 011, тобто. $R(X) = X + X_2$, а протягом n тактів на вихід надходить кодова комбінація $0111001 \sim X + X_2 + X_3 + X_6$.

При апаратній реалізації декодерів циклічного коду для визначення синдрому використовують схему, яка здійснює процедуру ділення полінома на поліном (див. рис.5.3).

Таблиця 2 — Стан комірок зсувного регістра та виходи схеми по тактам.

Номер такту	Вхід	СТАН					
		комірки			ключів		
		X^0	X^1	X^2	Вихід	КП1	КП2
0	--	0	0	0	--	Замкнуто	Розімкнено
1	1	1	1	0	1		
2	0	0	1	1	0		
3	0	1	1	1	0		
4	1	0	1	1	1		
5	--	0	0	1	1	Розімкнено	Замкнуто
6	--	0	0	0	1		
7	--	0	0	0	0		

При побудові декодера слід додатково включати запам’ятовуючий пристрій на k елементів і схему запиту залишку при розподілі. Ця схема складається із схеми логічного складання (АБО) на r входів та схеми логічного множення (І) на два входи. Зсувні регістри і зворотні зв'язки повинні відповідати структурі полінома $G(X)$, що породжує, тобто число комірок регістра має бути рівним r, а замкнутий зворотний зв'язок повинен відповідати ненульовим коефіцієнтам полінома $G(X)$.

Побудуємо схему кодера для циклічного коду Хеммінга (7, 4) з породжуючим поліномом $G(X) = 1 + X + X_3$, і за тактами зсувних імпульсів простежимо за його роботою. Схема кодера буде вирішувати проблему виявлення помилок.

На рис.5.6 наведена схема кодера, в табл.6 представлені стани комірок регістра при декодуванні вхідної кодової комбінації $B_i(X) = X + X_{2+}\mathrm{X}_3 + \mathrm{X}_6 \sim 111001$, що зчитується без помилок. Декодуючий пристрій наступним чином. Кодова комбінація $B_i(X)$ старшим розрядом вперед надходить на

зсувний регістр для визначення залишку при розподілі і запам'ятовуючий пристрій на k елементів через відкриту схему I_1, яка через k тактів закривається, так як припиняється подача з синхронізатора тактових імпульсів на один із входів схеми I_1.

Таблиця 3 — Стан комірок зсувного регістра та виходи схеми по тактам.

Вхід B(X)	Номер такту	Стан комірок			Вихід ЗР
		X^0	X^1	X^2	
--	Поч. стан	0	0	0	--
1	1	1	0	0	0
0	2	0	1	0	0
0	3	0	0	1	0
1	4	0	1	0	1
1	5	1	0	1	0
1	6	0	0	0	1
0	7	0	0	0	0

При цьому в запам'ятовуючий пристрій записується k інформаційних символів кодової комбінації $B_i(X)$, що приймається.

В зсувний регістр надходять усі n елементів $B_i(X)$, і після n тактів відбувається опитування стану комірок регістра шляхом подачі циклового імпульсу з синхронізатора на схему I_2. Якщо $R(X) \neq 0$, то на виході схеми I_2 імпульс не з'явиться і зчитування із запам'ятовуючого пристрою прийнятих інформаційних символів не відбудеться. Якщо $R(X) = 0$, то з'явившийся на виході I_2 імпульс зчитує $A_i(X)$ на вихід і видає чотири інформаційні біти одержувачу повідомлень.

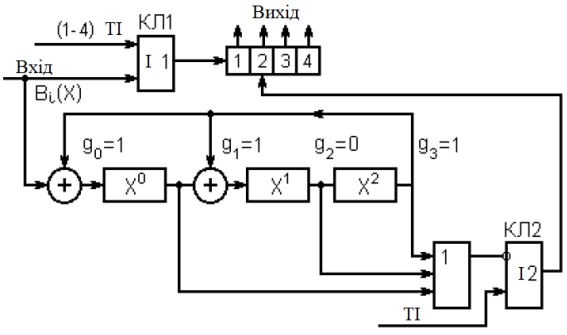


Рисунок 5.6 — Схема декодера

3.5 Принципи побудови декодера для циклічних кодів із виправленням помилок

Декодування прийнятих комбінацій циклічного коду можна проводити різними методами. Поряд із синдромним методом декодування, заснованим на обчисленні залишку від поділу прийнятої комбінації на код породжувальний поліном, існує цілий ряд інших методів, що спрощують процедуру декодування і не вимагають зберігання в пам'яті декодера великого числа синдромів при обробці довгих кодів. Для довгих циклічних кодів розроблені спеціальні ітеративні процедури декодування з виправленням кількох помилок, наприклад, метод Берле-Кемпа або досконаліший ітеративний алгоритм Тренча-Берлекемпа-Мессі (ТБМ-метод), що оперує з поліномами над полями Галуа.

Декодуючі пристрої для кодів, призначених лише для виявлення помилок, суттєво не відрізняються від схем кодера. У них додається лише буферний регістр для зберігання прийнятого повідомлення на час проведення операції ділення. Якщо залишок-синдром при діленні виходить нульовим, що свідчить про відсутність помилки, то інформація з буферного регістру зчитується в дешифратор повідомлення отримувача сигналу. Якщо залишок виявлено, що свідчить про наявність помилки, то інформація в буферному регістрі знищується і на сторону, що передає, до джерела сигналу посилається імпульс запиту повторної передачі по зворотному каналу зв'язку.

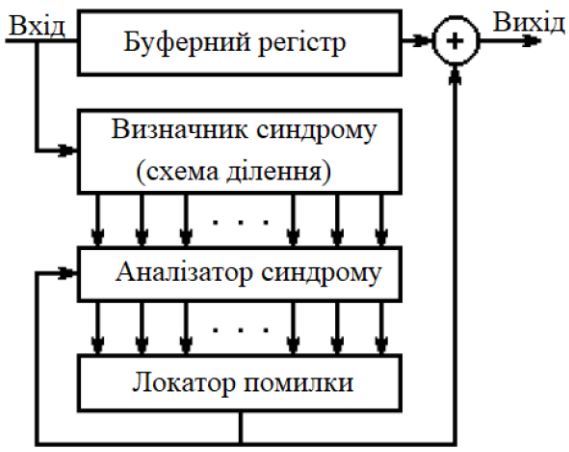


Рисунок 5.7 — Схема декодера з виправленням помилок

Символи кодової комбінації, яка підлягає декодуванню, що, можливо, містить помилку, послідовно, починаючи зі старшого розряду, вводяться в n-розрядний буферний регістр зсуву і одночасно в схему визначника синдрому, де за n тактів поділу визначається залишок, який у разі синхронної, безперервної передачі кодових комбінацій відразу ж переписується в аналогічний регістр схеми аналізатора синдрому.

До складу схеми аналізатора синдрому може входити постійний запам’ятовуючий пристрій, в якому записані всі можливі конфігурації синдромів з відповідними їм шумовими векторами. Кодові комбінації шумових векторів (4.16) містять "поодинокі" символи на тих позиціях, які в процесі передачі повідомлення каналом зв'язку виявилися спотвореними перешкодами.

Локатор помилок (визначник місця помилок) представляє собою комбінаторно-логічну схему, що видає на вихід одиничні символи в ті моменти часу, коли кожен з помилкових символів прийнятої кодової комбінації займає в буферному регістрі крайню праву комірку. При наступному тактовому зсуві локатор помилки (детектор помилки) формує символ "1", який надходить на суматор корекції, що є схемою додавання по модулю 2, де виправляється спотворений символ.

Одночасно через ланцюг зворотного зв’язку з виходу локатора помилки подається одиничний символ на аналізатор синдрому, що у ряді конкретних схемних рішень побудови аналізатора спрощує його побудову на основі ЛПС без використання постійного запам’ятовуючого пристрою. Складність аналізатора синдрому і локатора помилки залежить від гарантованого числа помилок, що виправляються і виявляються. Найпростіші схемні рішення виходять при обробці кодів, розрахованих на виправлення поодиноких помилок.

З розгляду логіки роботи структурної схеми декодера (рис. 5.7), найскладнішою його частиною є необхідність запам'ятовування заздалегідь обчислених синдромних поліномів і відповідних їм векторів помилок. Перевагою циклічного коду є те, що аналізатор синдрому можна значно

спростити, скориставшись алгебраїчною структурою коду для пошуку зв'язків між синдромами при числі помилок, що виправляються $g_i > 1$. Спираючись на ці зв'язки, можна запам'ятати в постійний запам’ятовуючий пристрій лише поліноми помилок, що відповідають деяким типовим синдромним поліномам, а обчислення інших здійснити за допомогою простих обчислювальних алгоритмів.

3.6 Програмна реалізація перфорації циклічного коду.

Для виконання поставленої задачі було використано могу програмування Java. У програмі кодується матричним способ інформаційне слово, виконується перфорація бітів кодової послідовності і знаходження виколотих бітів.

Конструктор класу Coder() зображен на рис. задається інформаційне слово, характеристичні матриці A та B.

```
public Coder(String  info_word, int[][] A, int[][] B) {
    String [] result=info_word.split( regex: "");
    this.inf_word = new int [result.length];
    for (int i = 0; i < result.length; i++) {
        inf_word[i]=Integer.parseInt(result[i]);
    }
    this.A = A;
    this.B = B;
}
```

Рисунок 6 — Конструктор класу Coder.

За допомогою функції sSeguence() (рис. 7) обраховуються стани лінійного автомата за формулою (2.7).

```
public int[] sSequence () {
    int[] codedWord = new int[inf_word.length + A.length];
    System.arraycopy(inf_word,  srcPos: 0, codedWord,  destPos: 4, inf_word.length);
    for (int i = codedWord.length - 1; i >= 0; i--) {
        S = CodeUtils.sumFunction(CodeUtils.multiplyFun(A, tempS), CodeUtils.multiplyOne(B,
        tempS = S;
    }
    int i = 0;
    for (int[] m : tempS) {
        codedWord[i] = m[0];
        i++;
    }
    System.out.println("Отримана кодова  послідовність:\n");
    System.out.println(Arrays.toString(codedWord));
    return codedWord;
}
```

Рисунок 7 — Структура функції sSeguence ().

Множення матриць (рис. 8) реалізовано в статичному методі multiplyFun() класу CodesUtils .

```
public static int[][] multiplyFun(int[][] A, int[][] B) {
    int[] matrixA;
    int[] matrixB;
    matrixA = matrixSize(A);
    matrixB = matrixSize(B);
    int c[][] = new int[4][1];
    for (int i = 0; i < matrixA[0]; ++i) {
        for (int j = 0; j < matrixB[1]; ++j) {
            for (int k = 0; k < matrixA[1]; ++k) {
                c[i][j] ^= A[i][k] & B[k][j];
            }
        }
    }
    return c;
}
```

Рисунок 8 — Структура функції multiplyFun().

Сумування матриць(рис. 9) представлено в методі sumFunction() класу CodesUtils.

```
public static int[][] multiplyFun(int[][] A, int[][] B) {
    int[] matrixA;
    int[] matrixB;
    matrixA = matrixSize(A);
    matrixB = matrixSize(B);
    int c[][] = new int[4][1];
    for (int i = 0; i < matrixA[0]; ++i) {
        for (int j = 0; j < matrixB[1]; ++j) {
            for (int k = 0; k < matrixA[1]; ++k) {
                c[i][j] ^= A[i][k] & B[k][j];
            }
        }
    }
    return c;
}
```

Рисунок 9 — Структура функції sumFunction ().

Функція perforateSequence() (рис. 10) класу Perforation виконує операцію перфорації кодового вектора .

```
public static int[] perforateSequence(int [] code,int perforateCount){
    System.out.println("Виколюєм перші два біта");
    int [] newArray=new int [code.length-perforateCount];
    System.arraycopy(code, srcPos: 2,newArray, destPos: 0,newArray.length);
    System.out.println("Perforated code:\n");
    System.out.println(Arrays.toString(newArray));
    return newArray;
}
```

Рисунок 10 — Структура функції perforateSequence()

Конструктор класу Decoder() (рис. 11)приймає в конструкторі характеристичні матриці A, B, кодовий вектор та кількість виколотих в ньому бітів.

```
public Decoder(int[][] A, int[][] B, int[] codedSequence, int perforateCount) {

    this.codedSequence = new int[codedSequence.length + perforateCount];
    for (int i = 0; i < perforateCount; i++) {
        this.codedSequence[i] = 0;
    }
    int count = 2;
    for (int i = 0; i < codedSequence.length; i++) {
        this.codedSequence[count] = codedSequence[i];
        count++;
    }
    this.perforateCount = perforateCount;
    this.A = A;
    this.B = B;
}
```

Рисунок 11 — Конструктор класу Decoder

В методі resolveSequence() (рис.12)обраховується синдром помилки, який потім прирівнюються до нульового контрольного синдрому і відновлюються виколоті біти.

```
public String[] resolveSequence() {
    for (int i = codedSequence.length - 1; i >= 0; i--) {
        S = CodeUtils.sumFunction(CodeUtils.multiplyFun(A, tempS), CodeUtils.multiplyOne(B, co
        tempS = S;
    }
    String[] stringSequence = new String[codedSequence.length];
    for (int i = 0; i < stringSequence.length; i++) {
        stringSequence[i] = Integer.toString(codedSequence[i]);
    }
    finalS = solvePerforation(finalS);
    int i = 0;
    for (String[] m : finalS) {
        stringSequence[i] = m[0];
        i++;
    }
    return stringSequence;
}
```

Рисунок 12 — Структура функції resolveSequence()

Виколоті біти знаходяться відповідно до формули (2.8) (рис.13)

```
if(sum[j][i].equals("x+1")){
    sum[j][i]="!x";
}
if(sum[j][i].equals("x+0")){
    sum[j][i]="x";
}
```

Рисунок 13— Програмна реалізація формули (2.8).

```
public String[][] solvePerforation(String[][] code) {
    for (String[] m : code) {
        if (m[0].equals("!x")) {
            m[0] = "1";
        }
        if (m[0].equals("x")) {
            m[0] = "0";
        }
    }
    System.out.println("Синдром буде нульовим, якщо: ");
    CodeUtils.showStringMatrix(code);
    return code;
}
```

Рисунок 14 — Структура функції solvePerforation().

Алгоритм роботи програми.

З клавіатури задається інформаційний вектор через консоль програми;

```
Введіть інформаційний вектор :

10010000001
```

Рисунок 15 — Введення інформаційного вектора.

В кодері обчислюється кодова послідовність і виводиться на екран;

```
Отримана кодова  послідовність:

[1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1]
```

Рисунок 16 — Виведення кодової послідовності.

Виколюються біти в контрольному векторі;

```
Виколюєм перші два біта
Perforated code:

[0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1]
```

Рисунок 17 — Перфорація контрольного вектора.

Обчислення в декодері контрольного вектора і обчислення виколотих бітів.

```
Синдром буде нульовим, якщо:
1
0
0
0
[1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1]
```

Рисунок 18 — Відновлення виколотих бітів.

ВИСНОВОК

У бакалаврській дипломній роботі було проведено аналіз існуючих завадостійких кодів в результаті якого було прийняте рішення використовувати саме циклічні коди в даній роботі.

Було розглянуто існуючі методи модифікації параметрів циклічних кодів і розроблено метод перфорації на основі лінійних автоматів та представлено метод розв’язання системних лінійних рівнянь в полях Галуа. Також представлена програмна реалізація перфорації циклічного кода Хеммінга при кодуванні та декодуванні на основі лінійних автоматів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Введення в теорію завадостійкого / Давыдов А.В., Мальцев. А.А. ― Нижній Новгород: Нижегородский госуниверситет, 2014. – 123 с.

2. Коди та устрій завадостійкого кодування / Корольов А. Науково― практичне видання. ―Мн.: Бестпринт, 2002. – 268 с.

3. Мистецтво завадостійкого кодування / Р.Моралес М: Техносфера. 2005. ― 320 с.

4. Електронні системи / Й.Й. Білинський К.В Огородник М.Ю. Юкиш Навчальний посібник Вінниця ВНТУ 2011 – 209с.

5. Завадостійке кодування в телекомунікаційних системах / Банкет В.Л. Іващенко П.В. Навч. Посіб. Одеса 2011 104.с.

6. Теорія кодування / Касамі Т. Токура Вид―во: М: Мир, 1978 576 с.

7. Основи м’якого декодування надлишкових кодів в стираючому каналі зв’язку / Гладкіх А.А. - Ульяновськ: УлДТУ, 2010. ― 379 с.

8. Кодування інформації / Березюк Н.Г. ― Харків, 1978.―252 с.

9. Основи теорії завадостійких кодів / Васильєв К.К., Новосільцев Д.Д., Смирнов В.М. – Ульяновськ: УлДТУ, 2000. ― 91 с.

10. Основи теорії завадостійких кодів / Васильєв К.К., Новосільцев Д.Д., Смирнов В.М. – Ульяновськ: УлДТУ, 2000. ― 91 с.

11. Паралельне декодування вкорочених циклічних кодів / Семеренко В.П. – ВНТУ 2012. ― 40с.

12. Теорія і практика кодів,контролюючих помилки / Блейхут Р. 1986. 576 с.

13. Лінійні послідовнісні машини: / А. Гілл. – М.: Наука, 1974. – 288 с.

14. Цифровий зв’язок / Джон. Прокіс; 2000. ― 800 с.

15. Завадостійкі циклічні коди / Нікітін Г.І., 2003. ― 33с.

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
проф., д.т.н.. Азаров О.Д..
" " 2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання бакалаврської дипломної роботи
"Програмно-апаратні засоби модифікації циклічних кодів"
08-23.БДР.026.00.000 ТЗ

Науковий керівник: доцент к.т.н.
Семеренко В.П.

Студент групи 2КІ-18б
Мартинов П.Г.

м. Вінниця – 2022

1 Підстава для використання бакалаврської кваліфікаційної роботи (БДР)

1.1 Актуальність розробки полягає у необхідності вирішення проблеми покращення швидкості коду за допомогою перфорації коду.

1.2 Наказ про затвердження теми бакалаврської дипломної роботи.

2 Мета і призначення БДР

2.1 Мета проекту — збільшення швидкості передачі даних завдяки модифікації параметрів завадостійкого коду.

2.2 Призначення розробки — виконання бакалаврського дипломного проекту із подальшим впровадженням та розвитком продукту.

3 Вихідні дані для виконання БДР

3.1 Аналіз завадостійких та циклічних кодів.

3.2 Розгляд принципу операції модифікації коду за допомогою перфорації.

3.2 Розробка алгоритму перфорації коду.

4 Вимоги до виконання БДР

Головна вимога — розробити зручний інтерфейс до веб-додатку, використовуючи усі здобуті навчики та знання.

5 Етапи БДР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

6 Матеріали, що подаються до захисту БДР

До захисту подаються: пояснювальна записка БДР, ілюстративні матеріали, протокол попереднього захисту БДР на кафедрі, відгук наукового керівника, анотації до БДР українською та іноземною мовами, довідка про відповідність оформлення БДР діючим вимогам.

Таблиця А.1 — Етапи БДР

№ з/п	Назва етапів виконання комплексної бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі роботи	08.03.22	Аналітичний огляд літературних джерел, розділ 1
2	Характеристика предметної області	17.03-01.04.22	розділ 1
3	Аналіз методів модифікації циклічних кодів	03.04-09.04.22	розділ 1
4	Модифікація циклічних кодів на основі лінійних автоматів	12.04-29.04.22	розділ 2
5	Програмно-апаратна реалізація методів модифікації кодів	01.05-18.05.22	розділ 3
6	Аналіз виконання роботи, висновки, додатки	21.05-25.05.22	Пояснювальна записка
7	Перевірка якості виконання бакалаврського проекту та усунення недоліків	28.05-09.06.22	Пояснювальна записка, графічний матеріал і презентація

7 Порядок контролю виконання та захисту БДР

Виконання етапів графічної та розрахункової документації БДР контролюється науковим керівником згідно зі встановленими термінами. Захист БДР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання БДР

При оформлюванні БДР використовуються:

— ДСТУ 3008 : 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302 : 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

— документами на які посилаються у вище вказаних.

ДОДАТОК Б

Лістинг класу Coder для кодування кодової послідовності

```
import java.util.Arrays;
public class Coder {
   int[] inf_word;
   int[][] A; int[][] B; int[][] S;
   int[][] tempS = {{0},{0},{0},{0},};
   public Coder(String  info_word, int[][] A, int[][] B) {
      String [] result=info_word.split("");
      this.inf_word = new int [result.length];
      for (int i = 0; i < result.length; i++)
         inf_word[i]=Integer.parseInt(result[i]);
      this.A = A; this.B = B;
   }
   public int[] sSequence () {
      int[] codedWord = new int[inf_word.length + A.length];
      System.arraycopy(inf_word, 0, codedWord, 4, inf_word.length);
      for (int i = codedWord.length - 1; i >= 0; i--) {
         S = CodeUtils.sumFunction(CodeUtils.multiplyFun(A, tempS),
CodeUtils.multiplyOne(B, codedWord[i]));
         tempS = S;       }
      int i = 0;
      for (int[] m : tempS) {
         codedWord[i] = m[0];
         i++;       }

System.out.println("Отримана кодова послідовність:\n"+Arrays.toString(codedW
ord));
      return codedWord;    }
```

ДОДАТОК В

Лістинг класу CodeUtils для виконування операцій над матрицями.

```
public class CodeUtils {
    public static int[][] multiplyOne(int[][] A, int B) {
        int[][] n = {
                {0},
                {0},
                {0},
                {0},
        };
        if (B == 1) {
            return A;
        }
        return n;
    }

    public static int[][] multiplyFun(int[][] A, int[][] B) {
        int[] matrixA;
        int[] matrixB;
        matrixA = matrixSize(A);
        matrixB = matrixSize(B);
        int c[][] = new int[4][1];
        for (int i = 0; i < matrixA[0]; ++i) {
            for (int j = 0; j < matrixB[1]; ++j) {
                for (int k = 0; k < matrixA[1]; ++k) {
                    c[i][j] ^= A[i][k] & B[k][j];
                }
            }
        }
```

```
    return c;
}

public static int[][] sumFunction(int[][] A, int[][] B) {
    int[] matrixA;
    int[] matrixB;
    matrixA = matrixSize(A);
    matrixB = matrixSize(B);
    int c[][] = new int[4][1];
    for (int j = 0; j < matrixB[0]; ++j) {
        for (int i = 0; i < matrixB[1]; ++i) {
            c[j][i] = A[j][i] ^ B[j][i];
        }
    }
    return c;
}
public static int[] matrixSize(int[][] matrix) {
    int[] x = new int[2];
    int line = 0;
    int length = 0;
    for (int[] m : matrix) {
        line += 1;
        length = m.length;
    }
    x[0] = line;
    x[1] = length;
    return x;
}
public static int[] StringMatrixSize(String[][] matrix) {
    int[] x = new int[2];
    int line = 0;
```

```
        int length = 0;
        for (String[] m : matrix) {
            line += 1;
            length = m.length;
        }
        x[0] = line;
        x[1] = length;
        return x;
    }

    public static void showMatrix(int[][] matrix) {
        int l = matrixSize(matrix)[0];
        int s = matrixSize(matrix)[1];
        for (int i = 0; i < l; i++) {
            for (int j = 0; j < s; j++) {
                System.out.print(matrix[i][j] + "\t");
            }
            System.out.println();
        }
    }
    public static void showStringMatrix(String[][] matrix) {
        int l = StringMatrixSize(matrix)[0];
        int s = StringMatrixSize(matrix)[1];
        for (int i = 0; i < l; i++) {
            for (int j = 0; j < s; j++) {
                System.out.print(matrix[i][j] + "\t");
            }
            System.out.println();
        }
    }
}
```

ДОДАТОК Г

Лістинг класу Decoder для декодування кодової послідовності з виколотоими кодовими векторами

```
import java.util.Arrays;
public class Decoder {
    String[][] finalS = {
            {"!x"},
            {"x"},
            {"0"},
            {"0"},
    };
    int[][] tempS = {
            {0},
            {0},
            {0},
            {0},
    };
    int[][] S;
    int[][] A;
    int[][] B;
    int[] codedSequence;
    int perforateCount;

    public Decoder(int[][] A, int[][] B, int[] codedSequence, int perforateCount) {

        this.codedSequence = new int[codedSequence.length + perforateCount];
        for (int i = 0; i < perforateCount; i++) {
            this.codedSequence[i] = 0;
        }
        int count = 2;
```

```
        for (int i = 0; i < codedSequence.length; i++) {
            this.codedSequence[count] = codedSequence[i];
            count++;
        }
        this.perforateCount = perforateCount;
        this.A = A;
        this.B = B;
    }

    public String[] resolveSequence() {
        for (int i = codedSequence.length - 1; i >= 0; i--) {
            S = CodeUtils.sumFunction(CodeUtils.multiplyFun(A, tempS),
CodeUtils.multiplyOne(B, codedSequence[i]));
            tempS = S;
        }
        String[] stringSequence = new String[codedSequence.length];
        for (int i = 0; i < stringSequence.length; i++) {
            stringSequence[i] = Integer.toString(codedSequence[i]);
        }
        finalS = solvePerforation(finalS);
        int i = 0;
        for (String[] m : finalS) {
            stringSequence[i] = m[0];
            i++;
        }
        return stringSequence;
    }

    public String[][] solvePerforation(String[][] code) {
        for (String[] m : code) {
            if (m[0].equals("!x")) {
```

```
                m[0] = "1";
            }
            if (m[0].equals("x")) {
                m[0] = "0";
            }
        }
        System.out.println("Синдром буде нульовим, якщо: ");
        CodeUtils.showStringMatrix(code);
        return code;
    }

    public int[] getCodedSequence() {
        return codedSequence;
    }
}
```

ДОДАТОК Д

Лістинг основного коду програми

```
import java.util.Arrays;
import java.util.Scanner;
public class Main {
    static int[][] A = {
            {0, 0, 0, 1},
            {1, 0, 0, 1},
            {0, 1, 0, 0},
            {0, 0, 1, 0},
    };


    static int[][] B = {
            {1},
            {0},
            {0},
            {0},
    };

    public static void main(String[] args) {
        System.out.println("Введіть інформаційний вектор :\n");
        Scanner in = new Scanner(System.in);
        String codeWord = in.nextLine();
        System.out.println("Введений інформаційний вектор:\n" + codeWord);

        Coder coder = new Coder(codeWord, A, B);
        int[] codedSequence = coder.sSequence();

       int [] perforatedCode=Perforation.perforateSequence(codedSequence, 2);
```

```
        Decoder decoder = new Decoder(A,B,perforatedCode,2);

        System.out.println( Arrays.toString(decoder.resolveSequence()));


    }
}
```

ДОДАТОК Е

Лістинг класу для виконання операцій з виколотими бітами в матрицях

```
import java.util.Arrays;
public class Perforation {

    public static int[][] StringToInt(String[][] A) {
        int colums = 0;
        int lines = 0;
        for (String[] m : A) {
            lines++;
            colums = m.length;
        }
        int[][] tempArr = new int[lines][colums];
        int c = 0;
        for (String[] m : A) {
            for (int i = 0; i < colums; i++) {
                tempArr[c][i] = Integer.parseInt(m[i]);
                c++;
            }
        }
        return tempArr;
    }
    public static int[] perforateSequence(int [] code,int perforateCount){
        System.out.println("Виколюєм перші два біта");
        int [] newArray=new int [code.length-perforateCount];
        System.arraycopy(code,2,newArray,0,newArray.length);
        System.out.println("Perforated code:\n");
        System.out.println(Arrays.toString(newArray));
        return newArray;
```

```
    }
    public static String[][] perforationSum(String[][] A, String[][] B) {
        int[][] matrixB = StringToInt(B);
        int[] matrixSize = CodeUtils.matrixSize(matrixB);
        String[][] sum = new String[4][1];
        for (int j = 0; j < matrixSize[0]; ++j) {
            for (int i = 0; i < matrixSize[1]; ++i) {
                if (Character.isDigit(A[j][i].charAt(0))) {
                    sum[j][i] = Integer.toString(Integer.parseInt(A[j][i]) ^
Integer.parseInt(B[j][i]));
                } else {
                    sum[j][i] = A[j][i] + "+" + B[j][i];
                    if(sum[j][i].equals("x+1")){
                        sum[j][i]="!x";
                    }
                    if(sum[j][i].equals("x+0")){
                        sum[j][i]="x";
                    }
                }
            }
        }
        return sum;
    }
    public static String [][] findX(String [][]A){
        for(String [] m:A){
            if(m[0].equals("!x")){
                m[0]="1";
            }
            if(m[0].equals("x")){
                m[0]="0";
            }
```

```
    }
    return A;

}

public static String[][] perforationMultiplyFun(String[][] A, String[][] B) {
    int[] matrixA = CodeUtils.StringMatrixSize(A);
    int[] matrixB = CodeUtils.StringMatrixSize(B);
    int[] matrixSize = CodeUtils.StringMatrixSize(B);
    String sum[][] = {
            {"0"},
            {"0"},
            {"0"},
            {"0"},
    };
    for (int i = 0; i < matrixA[0]; ++i) {
        for (int j = 0; j < matrixB[1]; ++j) {
            for (int k = 0; k < matrixA[1]; ++k) {
                if (Character.isDigit(A[i][k].charAt(0)) &&
Character.isDigit(B[k][j].charAt(0))) {
                    sum[i][j] = Integer.toString(Integer.parseInt(sum[i][j]) ^
                            Integer.parseInt(A[i][k]) & Integer.parseInt(B[k][j]));
                } else {

                    sum[i][j] = sum[i][j] + "+" + A[i][k] + "*" + B[k][j];
                }
            }
        }
    }
    return sum;
}}
```

ДОДАТОК Ж

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА
НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: програмно апаратні засоби модифікації циклічних кодів

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра обчислювальної техніки

(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 81,5% Схожість 18,5%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

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

Автор роботи _____ Мартинов П.Г.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Семеренко В.П.
(підпис) (прізвище, ініціали)