МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

С.В. Денбновецький, І.В. Мельник, Л.Д. Писаренко

КОДУВАННЯ СИГНАЛІВ В ЕЛЕКТРОННИХ СИСТЕМАХ. ЧАСТИНА 3. СПОСОБИ КОДУВАННЯ СИГНАЛІВ.

ТОМ 1. НАТУРАЛЬНІ, ЕФЕКТИВНІ ТА ЛІНІЙНІ КОДИ

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського як навчальний посібник для студентів, які навчаються за спеціальністю 171 «Електроніка» за освітніми програмами «Електронні пристрої та системи» та «Електронні прилади та пристрої»

Київ
КПІ ім. Ігоря Сікорського
2021

Рецензенти *Забара С.С.*, доктор технічних наук, професор, завідуючий кафедрою інформаційних технологій та програмування Інституту комп'ютерних технологій Відкритого міжнародного університету розвитку людини „Україна", лауреат Державної премії СРСР, лауреат Державної премії України

Прокопенко І.Г., доктор технічних наук, професор, професор кафедри авіаційних радіоелектронних комплексів факультету аеронавігації, електроніки та телекомунікацій Національного авіаційного університету України

Відповідальний редактор *Бідюк П.І.*, доктор технічних наук, професор

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 5 від 14.01.2021 р.) за поданням Вченої ради факультету електроніки (протокол № 12/2020 від 21.12.2020 р.)

Електронне мережне навчальне видання

Денбновецький Станіслав Володимирович, д-р. техн. наук, проф.
Мельник Ігор Віталійович, д-р. техн. наук, проф.
Писаренко Леонід Дмитрович, д-р. техн. наук, проф.

КОДУВАННЯ СИГНАЛІВ В ЕЛЕКТРОННИХ СИСТЕМАХ. ЧАСТИНА 3. СПОСОБИ КОДУВАННЯ СИГНАЛІВ. ТОМ 1. НАТУРАЛЬНІ, ЕФЕКТИВНІ ТА ЛІНІЙНІ КОДИ

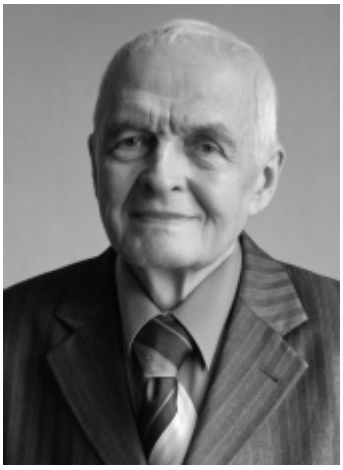
Кодування сигналів в електронних системах. Частина 3. Способи кодування сигналів: Том 1. Натуральні, ефективні та лінійні коди [Електронний ресурс] : навч. посіб. для студ. спеціальності 171 «Електроніка», освітньої програми «Електронні прилади та пристрої» / С.В. Денбновецький, І.В. Мельник, Л.Д. Писаренко ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 6,32 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2021.

У навчальному посібнику розглядаються стандартні способи кодування сигналів у сучасних електронних системах. У першому розділі розглянуті та систематизовані способи подання цифрових сигналів у каналах зв'язку, наведені частотні спектри сигналів для різних методів кодування. Окремо розглянуті способи кодування інформації в комп'ютерах, зокрема коди ASCII, коди для запису від'ємних та дрібних чисел, рефлексні коди та алгоритми скремблювання. Також приділяється увага ефективним способам кодування сигналів у безпроводових мережах, зокрема розглядається границя Велча та коди Голда як приклад ітеративних кодів. Розглянуті ефективні надмірні коди, зокрема коди Шеннона – Фано та Хаффмена. У другому розділі посібника розглянуті завадостійкі коди, зокрема код Хеммінга та циклічний код, а у третьому розділі – групові коди, зокрема коди БЧХ та Ріда – Соломона, а також згорткові коди, коди Вітербі та турбокоди. У четвертому розділі розглядаються способи кодування, які використовуються для стиснення інформації та її криптографічного захисту. Відмінною рисою та несумнівною перевагою посібника є досконале описання алгоритмів формування різноманітних цифрових кодів. Ці алгоритми реалізовані у комп'ютерних програмах, наведених у додатках.

Посібник призначений для студентів, які навчаються за спеціальністю «Електроніка», може бути корисним для студентів, які навчаються за спеціальностями «Телекомунікації» та «Комп'ютерна інженерія», а також для магістрів, аспірантів, викладачів та науковців відповідних спеціальностей.

Зміст

ВІДОМОСТІ ПРО АВТОРІВ



Денбновецький Станіслав Володимирович

Доктор технічних наук, професор кафедри електронних приладів та пристроїв факультету електроніки НТУУ «КПІ ім. І. Сікорського». Заслужений професор НТУУ «КПІ ім. І. Сікорського», у 1977-2006 роках завідував кафедрою електронних приладів та пристроїв НТУУ «КПІ», заслужений діяч науки і техніки України, відмінник народної освіти України, дійсний член IEEE, академік міжнародної академії інформатизації та академік Академії зв'язку України.

Народився у 1937 р., у 1959 р. закінчив радіотехнічний факультет КПІ за спеціальністю «Електронні прилади». Був аспірантом Інституту Фізики Напівпровідників АН УРСР. Працював у КПІ асистентом, інженером, старшим інженером, старшим викладачем, доцентом, професором.

Розробив ряд спеціалізованих лекцій курсів, зокрема «Електронні прилади», «Інформаційна електроніка», «Електронні системи» та інші. Один з ініціаторів створення на факультеті електроніки підготовки фахівців за системою бакалавр-магістр.

Наукова діяльність пов'язана з дослідженням та розробкою принципів дії електронно-променевих приладів, моделюванням фізичних процесів, з проектуванням та використанням електронно-променевих, плазмових приладів, пристроїв та систем.

Розробив теорію та довів до серійного виробництва низку електронних приладів та пристроїв: електронно-променеві гармати та системи керування електронним променем для обробки матеріалів, наукових досліджень та технічних застосувань; перетворювачі растрів, перетворювачі масштабу часу, прицезійні вимірювачі інтервалів часу та довжини коротких імпульсних сигналів для обробки на ЕОМ, запам'ятовуючі осцилографи, системи від ображення для ЕОМ.

Був головним консультантом трьох докторських та двадцяти чотирьох кандидатських дисертацій по електроніці та електронній техніці.

Член редколегій «Електроніка та зв'язок», «Semiconductor Physics», «Quantum Electronics and Optoelectronics». Працював головою спеціалізованої вченої ради з захисту кандидатських та докторських дисертацій.

Автор понад 400 наукових праць, зокрема 15 монографій та навчальних посібників. Має 98 авторських свідоцтв, підготував 24 кандидатів та 3 доктора технічних наук. Брав участь у роботі багатьох національних та міжнародних конференцій.

Мельник Ігор Віталійович

Доктор технічних наук, професор кафедри електронних пристроїв та систем факультету електроніки НТУУ «КПІ ім. І. Сікорського». Народився у 1966 р у м. Києві. У 1983 р. закінчив середню школу та поступив до Київського політехнічного інституту, який закінчив у 1989 р. за фахом “інженер електронної техніки”. З 1989 року по поточний час працює у Національному технічному університеті України “Київський політехнічний інститут імені І.Сікорсьнкого” на посадах аспіранта, асистента, старшого викладача, доцента та докторанта. Після закінчення аспірантури у 1994 р. захистив кандидатську дисертацію. Розробив ряд спеціалізованих курсів, зокрема: «Електронні пристрої обчислювальної техніки», «Обчислювальні системи та мережі», «Проектування інформаційних електронних систем», «Променеві інформаційно-технологічні системи», «Імовірнісні основи обробки даних». З 2005 по 2008 рік навчався у докторантурі Національного технічного університету України «КПІ». З 2004 по 2009 рік – за сумісництвом доцент кафедри комп’ютерної інженерії Відкритого міжнародного університету розвитку людини „Україна”, а з 2009 по 2016 рік – професор цієї кафедри, де викладав курси «Архітектура комп’ютерних мереж» та «Дослідження та проектування комп’ютерних мереж». У 2009 р. захистив докторську дисертацію. Був науковим керівником кандидатської дисертації по електроніці. Член експертної комісії МОН України з експертизи науково-технічних проектів за фахом «Радіоелектроніка та телекомунікаці»

Як науковець зробив внесок у розвиток теорії високовольтного тліючого розряду, розробив методи комп’ютерного розрахунку та проектування технологічних газорозрядних електронних гармат.

Має понад 200 наукових праць, серед яких більше 50 опубліковано за кордоном, зокрема 10 авторських свідоцтв, 5 патентів України та 8 навчальних посібників. Опубліковані навчальні посібники: «Інформаційні комп'ютерні мережі» (2006), «Комп'ютерні мережі та телекомунікації» (2007), «Система науково-технічних розрахунків MatLab та її використання для розв'язання задач із електроніки», у двох томах (2009), «Проектування та дослідження комп'ютерних мереж» (2010), «Основи проектування безпроводових комп'ютерних мереж» (2011), «Planung und Optimierung von Rechnernetzen» (2011, німецькою мовою), «Основы построения компьютерных сетей» (2013, видана російською мовою в Азербайджані), «Кодування сигналів в електронних системах. Частина 1. Параметри сигналів та каналів зв'язку та методи їхнього оцінювання» (2016), «Кодування сигналів в електронних системах. Частина 2. Математичні основи теорії кодування», у трьох томах (2018), «Основи програмування на мові Python», у двох томах (2020). Брав участь у роботі багатьох національних та міжнародних конференцій. Член організаційних комітетів міжнародних конференцій «Elnano» та «UkrMiCo». Член редакційної колегії журналу «Прикладні проблеми математичного моделювання». Член спеціалізованої вченої ради з захисту кандидатських та докторських дисертацій.

Коло наукових інтересів: комп'ютерне моделювання фізичних процесів та явищ в газорозрядних електронних приладах, комп'ютерне моделювання технологічних процесів, комп'ютерні мережі та INTERNET, методи математичного моделювання електронних пристроїв та систем.

Писаренко Леонід Дмитрович

Доктор технічних наук, професор, кафедри пристроїв та систем факультету електроніки НТУУ «КПІ ім. І. Сікорського». Декан факультету електронної техніки (1992 – 1996), декан факультету електроніки (1996 – 2003), завідувач кафедри електронних приладів та пристроїв (2006 – 2020). Академік Академії наук прикладної радіоелектроніки (1993), заслужений викладач НТУУ «КПІ» (1998). Почесний професор Харківського національного університету радіоелектроніки (2002). Народився у 1945 році. В 1969 з відзнакою закінчив Київський політехнічний інститут. Кандидатську дисертацію захистив в 1975, докторську – в 2006. З 2007 професор кафедри електронних приладів та пристроїв. Заступник голови спеціалізованої вченої ради з захисту кандидатських та докторських дисертацій.

Наукові дослідження присвячено проблемам моделювання в електроніці, зокрема, автоматизованому проектуванню електронних приладів та пристроїв, аналізу процесів обробки інформації стохастичними інформаційними каналами, розробці методів та засобів імовірнісного аналізу інформаційних систем. Створив інструментарій часо-частотного математичного моделювання та аналізу електронних інформаційних каналів в прикладних задачах.

Започаткував видання науково-технічного журналу «Електроніка та зв'язок» (1995), був його першим головним редактором (1995 – 1998), заступник головного редактора та відповідальний редактор (1998 – 2003), член редакційної ради журналу (з 2003).

Має медалі та відзнаки: Подяка Київського міського голови (2002), Почесна грамота ЦК профспілки працівників освіти і науки України (2006), Подяки Міністерства освіти і науки України (2005, 2006 та 2008), Відмінник освіти України (1998).

Автор понад 140 наукових праць, в тому числі: 5 підручників, 15 навчальних посібників, 2 довідника, 2 науково-методичних видання, 17 авторських свідоцтв та патентів СРСР та України. Готує кандидатів та докторів наук. Основні праці: «Основы автоматизированного проектирования электронных приборов» (1987), «Численные методы анализа электронных приборов» (1988), «Математичні аспекти хвильового аналізу» (2001), «Ключові питання теорії цифрової обробки зображень» (2001), «Рекурсивні цифрові фільтри» (2001), «Теорія вейвлетів з елементами фрактального аналізу» (2002), «Вступ до алгебраїчної теорії перешкодостійкого кодування» (2002), «Основи теорії електронних кіл» (2006, 2008, 2013).

Вступ

Навчальний посібник, який пропонується, є продовженням комплексної, багатотомної багаторічної праці авторів, яка присвячена описанню засобів та способів кодування сигналів в електронних системах. У першій частині посібника [1] розглядалися способи формування складних сигналів в електронних системах та можливості їхнього математичного описання з використанням моделей різного рівня складності. Окремо були розглянуті чинники, які впливають на спотворення сигналів в каналах зв'язку та способи погодження параметрів сигналу та каналу зв'язку. Акцент був зроблений на описання способів формування цифрових сигналів, окремо розглядалися випадкові сигнали та способи їхнього математичного описання. Також у першій частині посібника були розглянуті головні положення теорії інформації, які є теоретичним підґрунтям для розвитку сучасної інформаційної електроніки та комп'ютерної техніки, а також для інженерної розробки сучасних цифрових електронних пристроїв.

Сучасні способи кодування цифрових сигналів загалом базуються на використанні положень та теорем дискретної математики. Тому друга частина посібника була присвячена досконалому описанню математичних основ теорії кодування цифрових сигналів. Були розглянуті такі важливі розділи сучасної дискретної математики, як теорія чисел, теорія множин, основи комбінаторики, основи двійкової арифметики та алгебра Буля, теорія скінченних груп та полів Галуа, теорія ґраток, теорія поліномів та твірних функцій, основи матричного аналізу, та теорія векторних просторів. Також окремо розглядалися імовірнісні підходи до аналізу сигналів, зокрема – основи теорії імовірностей, основи теорії черг, основи математичної статистики та статистичної радіотехніки. Також були розглянуті методи теорії штучного інтелекту, які широко використовуються для аналізу параметрів цифрових сигналів та визначення способів формування оптимальних сигнальних конструкцій. Зокрема, була розглянута теорія предикатів, теорія графів, теорія регулярних мов та скінченних автоматів, а

також основи сучасної теорії нечіткої логіки. Було показано, що теорія скінченних автоматів та нечітка логіка є основним теоретичним підґрунтям для подальшого бурхливого розвитку сучасної цифрової електроніки, зокрема програмованих електронних пристроїв та систем.

У третій частині посібника, яка пропонується, розглядаються способи кодування сигналів, які загалом ґрунтуються на базових теоретичних принципах, описаних у першій та другій частинах посібника. Розгляд способів кодування сигналів проведений комплексно, базові принципи кодування розглядаються на фізичному та логічному рівні [1]. У першому розділі розглянуті фізичні основи формування цифрових сигналів та створення відповідних оптимальних сигнальних конструкцій, параметри яких погоджуються із параметрами каналів зв'язку з метою забезпечення мінімального спотворення сигналу, який передається. Розглянуті способи потенціального та імпульсного кодування, а також способи формування цифрових сигналів з амплітудою, частотною та фазовою модуляцією. Особлива увага приділяється методам натурального кодування. Для всіх типів натуральних кодів проведені оцінки спектру сигналу. Серед натуральних кодів окремо розглянуті коди ASCII, рефлексні коди та код Грея, а також надлишкові натуральні коди 4B/5B, 8B/6T та алгоритми скрамблювання. Крім цього, розглядаються натуральні ітеративні та згорткові коди, які формуються на основі теорії дискретних груп. Серед кодів такого типу найбільш відомі коди Голда, які часто використовуються в системах зв'язку для формування ефективних сигнальних конструкцій. Підрозділ 1.5 присвячений описанню ефективних кодів, зокрема розглядаються принципи формування коду Шеннона – Фано та коду Хаффмена. Розглянуті також інші різновиди ефективних кодів, які застосовуються у сучасній цифровій електронній апаратурі, в комп'ютерній техніці та у системах зв'язку.

У другому розділі розглядаються основні теоретичні принципи завадостійкого кодування та прості завадостійкі коди, які знайшли широке

використання у сучасній цифровій електронній апаратурі. Зокрема, ретельно розглянута теорема Шеннона про канал зв'язку із завадами та наведене теоретичне обґрунтування теореми Котельникова – Найквіста [2 – 8]. Під час розгляду відповідних теорем теорії кодування використаний узагальнений математичний апарат теорії інформації, який був описаний у першій частині посібника. Подана узагальнена класифікація завадостійких кодів та показано, що найбільш простими з них, хоча і досить ефективними, є систематичні коди, зокрема лінійні та циклічні кодові конструкції.

Показано, що найбільш відомими з лінійних кодів є код із перевіркою на парність та код із повторенням елементів, хоча ефективність таких кодів, з точки зору можливостей виявлення та виправлення помилок, не є високою.

Окремо розглянуті коди Хеммінга та циклічні коди. Наведені узагальнені правила формування таких кодів, особливості використання яких розглянуті на конкретних прикладах.

У третьому розділі наведено описання блокових та згорткових кодів. Описані теоретичні положення, на яких ґрунтуються алгоритми створення таких кодів. Загалом це є теорія скінченних груп, теорія поліномів, теорія матриць, теорія ґраток та теорія скінченних автоматів, які були ретельно описані у другій частині посібника. Розглянутий алгоритм формування кодів Файра, які можна розглядати як проміжний варіант між циклічними та блоковими кодами. Код Файра побудований на основі алгоритмів формування циклічного коду, хоча він є блоковим. Такий підхід до вивчення матеріалу сприяє швидшому розумінню принципів формування блокових кодів тими читачами, які вже знають, як побудований циклічний код. Найбільш ефективно для формування блокових кодів використовується математичний апарат теорії поліномів.

Далі, у третьому розділі, розглянуті способи формування кодів Боуза – Чоудхурі – Хоквінгема (БЧХ) та кодів Ріда – Соломона. Наведені головні теореми теорії кодування, на яких базуються принципи формування кодів БЧХ. Показано, що коди Ріда – Соломона є різновидом кодів БЧХ та для формування їх послідовностей використовуються ті самі принципи. Але,

оскільки коди Ріда – Соломона мають дуже високу коректувальну здатність та є найбільш ефективними серед кодів БЧХ, вони знайшли найбільше практичне впровадження в електронній апаратурі та у навчальній літературі їх зазвичай розглядають окремо. Розглянуті параметри кодів Ріда – Соломона та проведений порівняльний аналіз різних способів декодування послідовностей цього коду. Показано, що найбільш ефективним серед способів цих способів, з обчислювальної точки зору, є алгоритм Берлекемпа – Мессі та теорема Форні. Проведені теоретичні оцінки коректувальної здатності кодів Ріда – Соломон залежно від параметрів їх.

Також розглянуті принципи формування каскадних та ітеративних кодів та можливості формування кодових конструкцій на основі аналізу спектру сигналів.

Розглянуті основні принципи формування згорткових кодів та показано, що теоретичною основою для їхнього створення є теорія імовірностей та теорія скінченних автоматів, розглянуті у другому та третьому томі другої частини посібника. Показано, що головною перевагою згорткових кодів є можливість формування та дешифрування кодової послідовності у реальному часі. Наведені прості структурні схеми кодерів з різними параметрами. Показано, що головними способами розшифрування послідовностей згорткового коду є алгоритм послідовного декодування та алгоритм Вітербі, пов’язаний з принципом максимальної правдоподібності. Наведена велика кількість прикладів щодо формування та декодування послідовностей згорткового коду. Введено поняття про матрицю станів та показано, як така матриця може бути використана для ефективної реалізації алгоритму послідовного декодування послідовностей згорткового коду за умови відомого алгоритму кодування. Показано, як для оцінки коректувальної здатності згорткового коду використовується теорія скінченних автоматів та теорія поліномів. Проведений порівняльний аналіз ручних та комп’ютерних методів оцінки коректувальної здатності згорткового коду з використанням теорії поліномів.

Як окремий підклас каскадних та згорткових кодів розглянуті турбокоди, способи розшифрування яких побудовані на принципі максимальної правдоподібності. Цей тип кодів, запропонований на початку дев'яностих років минулого століття, сьогодні вважається найбільш ефективним та широко впроваджується в сучасній цифровій електронній апаратурі та в системах зв'язку.

Натуральні, ефективні, лінійні та каскадні коди розглянуті у першому томі посібника, а блокові та згорткові коди – у другому. Третій том посібника буде присвячений алгоритмам стиснення та шифрування інформації, а також описанню відповідних способів формування цифрових сигналів.

Навчальний посібник, який пропонується, має вельми велике практичне значення для інженерної освіти в галузі електроніки. Всі теоретичні міркування підтверджені з практичної точки зору як наведеними принциповими електричними схемами відповідних кодувальних та декодувальних пристроїв, так і ретельним аналізом відповідних алгоритмів кодування та декодування цифрових сигналів. Несумлінною перевагою посібника є велика кількість програмних додатків, в яких реалізовані алгоритми кодування та декодування для різних кодових послідовностей. Всі програми написані мовою програмування системи науково-технічних розрахунків MatLab та відповідні програмні коди досконало описані в тексті посібника. Такі комп'ютерні програми можуть бути ефективно використані на лабораторному практикумі для наочного пояснення студентам принципів формування кодових послідовностей та їхнього розшифровування. Тобто, на основі кодів комп'ютерних програм, наведених у додатках А – Р, можуть бути побудовані цікаві цикли лабораторних робіт з курсів «Теорія сигналів» та «Теорія інформації».

Для контролю знань студентів та для закріплення ними вивченого матеріалу наприкінці кожного розділу наведено велика кількість контрольних питань та завдань.

Что, опять поврежденья на трассе?
Что, реле там с ячейкой шалят?
Всё равно, буду ждать, я согласен
Начинать каждый вечер с нуля.

Владимир Высоцкий
«Ноль семь»

Розділ 1 Принципи кодування інформації. Натуральні та ефективні коди

У цьому розділі розглядаються способи кодування двійкових та багатопозиційних сигналів. Окремо виділені способи кодування електричних сигналів у каналах зв'язку та способи запису двійкової інформації. Серед способів кодування електричних сигналів розглянуті потенціальне та імпульсне кодування, окремо розглядаються манчестерські коди, біполярне кодування із альтернативною інверсією, потенціальний код з інверсією за умови одиниці, а також коди B8ZS, HDB3 та MLT-3. Серед багаторівневих систем кодування розглянутий чотирерівневий потенціальний код 2B1Q . Розглядаються способи формування сигналів із фазовою модуляцією та квадратурною амплітудною модуляцією. Розглянута також диференціальна модуляція як спосіб формування двійкових сигналів, відмічені її переваги та недоліки. Серед різновидів двійкових кодів розглянуті код ASCII, який використовується в комп'ютерних системах, коди від'ємних чисел із доповненням до одиниці, код Грея, код 4B/5B та алгоритми скремблювання. Також розглядаються способи кодування сигналів у безпроводових мережах з використанням ітеративних кодів. Тут окрема увага приділяється т-послідовностям та кодам Голда. Розглянуті ефективні двійкові коди та теорема Шеннона про канал зв'язку без завад як основа теорії ефективного кодування. Як приклади ефективних кодів наведені способи побудови коду Шеннона – Фано та коду Хаффмена. Розглянуті також інші способи кодування, які використовуються у сучасних обчислювальних та телекомунікаційних мережах.

1.1 Загальні основи теорії кодування

1.1.1 Поняття про кодування сигналів

Взагалі під час кодування сигналів вирішується цілий ряд практичних задач, пов'язаних із обробленням та передаванням інформації в електронних системах. Перша із задач кодування – це подання повідомлень через таку

систему символів, яка забезпечує простоту, надійність та ефективність апаратної реалізації всіх електронних пристроїв. У системах зв'язку зазвичай стоїть інша задача, яка вирішується за допомогою кодування, і полягає вона у тому, що необхідно забезпечити найкраще узгодження між параметрами джерела повідомлень, приймача та каналу зв'язку, а також безпомилкове розпізнавання сигналу за умови наявності завад. Тому кодер на передавальній стороні та декодер на приймальній стороні є обов'язковими пристроями у структурі приймально-передавальної системи. Структурна схема такої системи наведена у другому розділі першої частини посібника [1]. Тобто, можна розглядати спосіб кодування сигналу як погодження його параметрів із параметрами лінії зв'язку та електронної системи, де цей сигнал використовується. З цієї точки зору засоби модуляції сигналів, які були розглянуті у розділі 5 першої частини посібника [1], також можна розглядати як один із способів його кодування. Серед способів кодування аналогових сигналів у системах зв'язку та в електронних системах розглядають також квадратурну модуляцію та різницеве кодування. Метод квадратурної модуляції сигналів розглядався у підрозділі 5.5.2 першої частини посібника [1]. Методи різницевого кодування основані на формуванні різницевого сигналу між сигналами кількох каналів системи, в результаті чого передається сумарний сигнал і різниця між сумарним сигналом та сигналом відповідного каналу. Способи різницевого кодування сигналів у каналах зв'язку, разом із квадратурною модуляцією, широко використовуються у сучасних системах телебачення та стереофонічного радіомовлення [51, 60]. Проте, як відмічалось у першій частині посібника у підрозділі 5.5.2, сигнали з квадратурною модуляцією не є ортогональними у комплексному просторі, і тому вони не можуть бути використані у системах без синхронізації [1]. Взагалі, як відмічалось у першій частині посібника, узгодження параметрів приймально-передавального обладнання та каналу зв'язку підвищує надійність роботи всієї системи та ефективність її використання.

Іншим способом кодування сигналів, який підвищує завадостійкість інформаційних систем, є дискретизація та цифрове подання, яке розглядалось у розділі 6 першої частини посібника [1]. І, в решті решт, однією із задач кодування сигналів в електронних системах є підвищення надійності передавання інформації в умовах сильних завад. Ця задача зазвичай вирішується через використання завадостійких кодів, які будуть досконало

розглянуті у розділах 2 та 3 цієї частини посібника.

Можна сказати, що під кодуванням у загальному розумінні цього слова розуміють перетворення інформаційних повідомлень в електричний або оптичний сигнал. Як під час передавання, так і під час збереження та обробки інформації значні переваги надає дискретна форма подання сигналів. Дискретні сигнали та способи їхнього формування, а також головні переваги та недоліки дискретного подання інформації, розглядалися у розділі 6 першої частини посібника [1]. З іншого боку, як було відмічено у розділі 5 першої частини посібника [1], модуляцію також можна вважати одним із способів кодування, а цей спосіб обробки використовується як для аналогових, так і для цифрових сигналів. У зв'язку з цим у теорії інформації існує поняття кодування у вузькому розумінні цього слова, під яким розуміють подання дискретних повідомлень у вигляді відповідних сполучень символів. Сукупність правил, відповідно до яких здійснюється формування коду, називається способом, або алгоритмом кодування. З точки зору теорії інформації, яка розглядалась у підрозділі 1.1 першої частини посібника, задача кодування полягає у тому, що послідовність символів від джерела повідомлення замінюють послідовністю кодових символів [2 – 6, 8, 10].

Оскільки кодовані сигнали мають низку переваг, мету кодування зазвичай не можна визначити однозначно. По-перше, часто намагаються подати повідомлення у такий системі символів, яка забезпечує простоту та надійність електронних пристроїв, а також їх ефективність. Друга важлива мета кодування полягає в тому, щоб забезпечити найкраще погодження фізичних властивостей джерела повідомлення, каналу зв'язку та приймача. Це зазвичай дозволяє скоротити час передавання інформації та підвищує ефективність систем зв'язку. Крім цього, існують способи кодування, які дозволяють забезпечити високу достовірність передавання інформації за умови наявності завад [2 – 6]. Для оцінки імовірності розпізнавання сигналів в умовах завад використовують поняття про сигнально-кодові конструкції, які розглядалися у розділах 4 та 5 першої частини посібника [1].

Але треба мати на увазі, що будь-яке кодування змінює структуру початкового сигналу, і за цих умов місткість інформації у закодованому повідомленні повинна залишатися незмінною відносно початкового повідомлення. Для проведення відповідних оцінок використовують математичний апарат теорії інформації та поняття ентропії, які розглядались у підрозділі 1.1 першої частини посібника [1]. Зокрема, одним із способів кодування двійкової інформації є ефективні коди, сутність яких полягає у тому, що символи, які мають більшу імовірність появи в інформаційній системі, кодуються коротшою послідовністю бітів і за рахунок цього мають меншу тривалість передавання та менше завантажують канал зв'язку. Методи інформаційних оцінок, які можуть бути використані для побудови ефективних кодів, розглядались у підрозділі 4.1 першої частини посібника [1]. Приклади ефективних кодів, які використовуються в інформаційних системах, розглядатимуться у підрозділі 1.5.

Як елементарні сигнали під час формування кодів розглядають одиночні імпульси змінного струму, або радіоімпульси, та одиночні імпульси постійного струму, або відеоімпульси [1 – 6]. Способи формування та аналізу радіоімпульсів розглядалися у розділі 5 першої частини посібника, а способи аналізу відеоімпульсів – відповідно у розділі 6 [1]. Елементарним сигналом може бути також пауза між імпульсами, або комбінація паузи та імпульсу. Такі сигнали розрізняють за їхніми параметрами, які називають кодовими ознаками. Відповідно до параметрів сигналів, які розглядалися у підрозділі 3.5 першої частини посібника [1], кодовими ознаками можуть бути полярність, амплітуда сигналу, його тривалість або частота. Для радіоімпульсів кодовою ознакою може бути також фаза сигналу. Спосіб фазової кодоімпульсної модуляції був описаний у підрозділах 5.3 та 5.6 першої частини посібника. Відповідні способи формування кодованих електричних сигналів розглядатимуться у підрозділі 1.2.3. Під час формування кодованих сигналів ефективно використовують також засоби синхронізації, які розглядалися у підрозділі 4.4 першої частини посібника [1].

Ефективність використання кодованих сигналів в електронних системах легко зрозуміти на такому наочному прикладі. Як спосіб кодування інформації можна розглядати мову, якою люди спілкуються між собою. У той же час, одну й ту ж саму інформацію можна подати по-різному. Можна дати точне, навіть математичне описання події, про яку Ви розповідаєте слухачеві, а можна пояснити свою думку інакше, припустимо, через образне уявлення, художньо. І різні люди, відповідно до того, як сформоване їхнє мислення, по-різному Вас зрозуміють. Якщо Ваш співрозмовник вивчав математику та краще розуміє точні описання, йому треба пояснювати свою думку через точні міркування. А якщо він займається образотворчим мистецтвом, йому краще навести доречний образний приклад. Наприклад, можна сказати: «Ми зустрілись ввечері біля шостої години», а можна: «Ми зустрілись весняним вечором, сонце заходило, у парку розцвіли каштани, і легкий вітерець ніжно торкав їх зелені листя та рожеві свічки, які грали своїми райдужними кольорами під променями теплого весняного сонця». Кожне з цих повідомлень несе свою інформацію, відрізняється лише форма її подання, і як Вас зрозуміє співрозмовник – це залежить від того, з ким Ви спілкуєтесь і що бажаєте розповісти. Крім мовного, можливі інші способи обміну інформацією між людьми, наприклад, це можуть бути жести або малюнки. Легко також проілюструвати ефективність кодування інформації за умови наявності завад. Наприклад, якщо студенти на занятті сильно шумлять, а викладач говорить тихо, вони не зможуть його зрозуміти. Проте, якщо викладач зробить на дошці рисунок або напише математичну формулу, студенти зможуть прийняти, записати та запам’ятати подану інформацію навіть за таких несприятливих умов. Інший приклад. Якщо людина сидить в навушниках, Ви не зможете з нею розмовляти, для цього необхідно спочатку взяти її за руку або за плече, і тим самим дати цій людині зрозуміти, що Ви звертаєтесь до неї. Всі наведені вище приклади у загальному випадку можна розглядати як різні способи кодування інформації. Наприклад, якщо Ви берете за руку Вашого співрозмовника, який слухає музику, це можна

розглядати як відповідний сигнал до нього: «Увага! Слухай мене!». До речі, як ще одну, досить нестандартну форму кодування інформації, можна розглядати мову глухонімих людей.

Зрозуміло також, що якщо Ви користаєтесь такою мовою, яку не розуміє Ваш співрозмовник, наприклад, китайською, Ви не зможете передати йому необхідну інформацію без перекладача. Тобто, способи кодування інформації в електронних системах та системах зв'язку для передавального та приймального пристроїв завжди мають бути погодженими. Нагадаємо ще раз типовий приклад, пов'язаний із різними способами кодування адрес та необхідністю встановлення однозначного зв'язку між сформованими кодами, який вже був наведений у підрозділі 1.4 другої частини посібника [48].

Відомо, що в комп'ютерних мережах введені два різновиди кодування адрес, а саме числові та доменні адреси вузлів мережі [15 – 23]. Це пов'язано з тим, що комп'ютерам, як електронним пристроям, простіше організовувати зв'язок через числові адреси, а люди краще розуміють та пам'ятають імена та образи, які з цими іменами пов'язані. Тому під час написання електронних листів або у разі звернення до Internet люди вказують доменні адреси відповідних вузлів мережі, які під час організації зв'язку та пошуку транзитних вузлів мережі автоматично перетворюються на числові [15 – 23]. Для цього використовується служба доменних імен DNS (абревіатура англійського словосполучення **D**omain **N**ame **S**ervice). Принцип роботи цієї служби, головною метою якої є забезпечення адекватності та взаємозв'язку числових та доменних адрес, легко зрозуміти на такому цікавому та доречному прикладі [15 – 23]. Уявіть собі, що Ви всіх своїх друзів, коли спілкуєтесь з ними, стали би ідентифікувати не за прізвищами та іменами, а за телефонними номерами. Саме так працюють сервери комп'ютерних мереж, коли надсилають повідомлення один одному. Наведемо іронічний приклад, у якому спробуємо перевести звичайну телефонну розмову на „мову серверів комп'ютерних мереж", тобто, здійснити перекодування доменних адрес на числові. Припустимо, Ви телефонуєте своєму другові і кажете:

«Привіт, Васю, це я, Сергій! Слухай, Васю! Ти з Петром давно спілкувався? А ти не знаєш, де він зараз? Я ніяк не можу йому додому додзвонитися. Справа у тому, що до мене вчора зателефонувала Наталка, і просила йому передати, щоб він терміново подзвонив Світлані. А, він зараз у Андрія? Це точно? Добре, я сам Андрію зателефоную. До побачення». Такий же самий діалог перекладений на «мову серверів», з іншим способом кодування, проходив би трохи іншим чином, оскільки сервери замість імен використовували би числові адреси, якими у нашому випадку є телефонні номери. Тобто, наведена розмова була б приблизно такою: «Привіт, 534-12-78, це я, 238-56-18! Слухай, 534-12-78! Ти з 454-98-72 давно спілкувався? А ти не знаєш, де він зараз? Я ніяк не можу йому додому додзвонитися. Справа у тому, що мені вчора зателефонувала 234-12-54, і просила йому передати, щоб він терміново подзвонив 275-13-90. А, він зараз у 512-47-95? Це точно? Добре, я сам 512-47-95 зателефоную. До побачення!». У цьому прикладі двох розмов, крім різної системи кодування, є ще одна важлива деталь. Якщо Ви культурна людина, то на початку розмови обов'язково привітаєте свого друга або колегу, а в кінці розмови прощаєтеся з ним. Крім того, після вітання, на початку телефонної розмови, Ви обов'язково представляєтеся, навіть коли спілкуєтеся із знайомою людиною. І все це не просто норми етикету, вони значно спрощують спілкування, розмова стає коротшою і несе більшу кількість інформації. Ваш співрозмовник відразу розуміє, хто з ним хоче спілкуватися, коли починається розмова, коли вона закінчується. Тому такий саме принцип використовується під час організації зв'язку для кодування повідомлень в електронних системах, зокрема в комп'ютерних мережах.

Розрізняють також способи кодування інформації для різних рівнів еталонної мережної моделі OSI, яка розглядалася у підрозділі 2.3 першої частини посібника [1]. Зокрема, у теорії комп'ютерних мереж розрізняють методи кодування із встановленням з'єднання між початковим та кінцевим вузлами та без встановлення з'єднання. Зазвичай такі способи кодування інформації називаються протоколами передавання даних. Встановлення

з’єднання між кінцевими вузлами зазвичай здійснюється на транспортному або вищих рівнях ієрархії моделі OSI, а на фізичному та канальному рівні більш важливим є передавання сигналів між транзитними вузлами без спотворень інформації [15 – 23]. Для цього використовуються способи подання двійкових сигналів та натуральні коди, які розглядатимуться у підрозділі 1.2 цієї частини посібника. Для передачі цифрових сигналів через канал зв’язку в умовах завад використовують завадостійкі коди, які розглядатимуться у розділах 2 та 3 цієї частини посібника. Під час встановлення з’єднання важливими є також способи стиснення та шифрування інформації [22, 23, 30, 33], які розглядатимуться у розділі 4 цієї частини посібника.

Відповідно до наведених вище міркувань, можна дати таке визначення процесу кодування сигналів в електронних системах.

Визначення 1.1. Кодуванням сигналу називається зміна параметрів сигналу відповідно до характеристик електронної системи та каналу зв’язку з метою зменшення імовірності його спотворення та спрощення його розпізнавання на приймальній стороні, за умови, що засоби передавання, приймання та обробки сигналу мають бути оптимальними та ефективними.

Відповідно до способу кодування, згідно із структурою системи зв’язку, яка наведена у першій частині посібника [1], на приймальній стороні здійснюється декодування сигналу з метою його оновлення.

Згідно із наведеним визначенням розрізняють способи кодування аналогових сигналів та способи кодування цифрових сигналів. Ефективним способом кодування аналогових сигналів є їхня модуляція, яка розглядалась у розділі 5 першої частини посібника [1]. Проте іншим, не менш ефективним способом кодування аналогових сигналів є їхня дискретизація та переведення в цифрову форму, ці способи обробки сигналів розглядалися у підрозділі 6 першої частини посібника. Зокрема, переваги та недоліки цифрової форми подання сигналів розглядалися у першій частині у підрозділі 6.1 [1].

Для цифрових двійкових та багатопозиційних сигналів використовують

способи кодування електричних сигналів у каналах зв’язку, які дозволяють зменшити імовірність спотворення цифрового сигналу. Відповідні способи кодування двійкових та багатопозиційних електричних сигналів у каналах зв’язку розглядатимуться у підрозділі 1.2 цієї частини посібника.

В свою чергу, існують відповідні способи подання цифрової інформації та запису цифрових повідомлень, і ці способи також зменшують імовірність спотворення інформації в цифрових електронних системах. Ці способи кодування пов’язані із тим, що послідовності слідування нулів та одиниць змінюються за відповідними законами, і деякі із таких змін мають високу імовірність, а деякі з них є малоймовірними. Такі способи кодування цифрової інформації пов’язані також із алгоритмами функціонування цифрових електронних систем. Вони розглядатимуться у підрозділі 1.3. Зокрема, до способів кодування цифрової інформації належать коди Грея та алгоритми скрамблювання. Для розгляду принципів функціонування комп’ютерних систем важливим є також розуміння головних принципів побудови коду ASCII та кодів із доповненням до одиниці для від’ємних чисел, які розглядатимуться у підрозділах 1.3.3 та 1.3.4 цієї частини посібника.

Серед способів кодування цифрової інформації розрізняють натуральні числові коди та завадостійкі коди, які також називають надлишковими. У натуральних кодах двійкових чисел використовується та ж сама кількість розрядів, що і для початкового числа, тобто, вони не несуть додаткової інформації. Проте натуральні коди більше пристосовані до алгоритмів функціонування електронних систем і тому з використанням таких способів кодування інформації імовірність спотворення цифрових сигналів є меншою. Крім того, використання таких кодів зазвичай веде до економії енергії, яку споживають електронні системи. Наприклад, до таких способів кодування відносяться код від’ємних чисел із доповненням до одиниці та код Грея, а також алгоритми скрамблювання, які розглядатимуться у підрозділі 1.3.6 цієї частини посібника. Узагальнена класифікація способів кодування сигналів наведена на рис. 1.1.

Рис. 1.1 Узагальнена класифікація способів кодування сигналів в електронних системах

Способи побудови завадостійких кодів, а також алгоритми виявлення та пошуку помилок, будуть розглянуті у розділах 2 та 3 цієї частини посібника. Загальні математичні основи теорії завадостійкого кодування були розглянуті у розділах 4 та 5 другої частини посібника [48].

1.1.2 Узагальнена класифікація цифрових кодів

Розглянемо узагальнену класифікацію цифрових кодів, яка наведена у вигляді діаграми на рис. 1.2. В цілому, за законами кодування, які використовуються, у теорії інформації розрізняють два класи кодів: арифметичні, або числові, та комбінаторні коди. Комбінаторні коди будуються за законами теорії сполучень. Наприклад, є абетка із трьох символів: *a*, *b* та *c*. Якщо одне елементарне повідомлення може містити лише 2 символи, із введених нами трьох символів можна створити 6 таких елементарних повідомлень: *aa*, *bb*, *cc*, *ab*, *ac*, *bc*, *ac*. У разі збільшенні можливої кількості символів у елементарному повідомленні кількість повідомлень також збільшується за відомим з комбінаторики законом для кількості сполучень із повтореннями [48, 54]:

$$f_k^r = C_{k+r-1}^{k-1} = \frac{(k+r-1)!}{r!(k-1)!}, \qquad (1.1)$$

де k – кількість елементів, r – можлива кількість сполучень, C_{k+r-1}^{k-1} – кількість сполучень без повторень із (k + r - 1) елементів по (k - 1) елемент. Найчастіше у технічних засобах передавання та обробки інформації використовують числові коди, але у сучасних засобах кодування іноді впроваджені і більш складні комбінаторні алгоритми [8, 24, 28, 29, 38].

Рис. 1.2 Узагальнена класифікація кодів

Цифрові коди діляться за кількістю елементарних символів, які у них

використовуються. За цим критерієм розрізняють *одиничні*, *двійкові* та *багатопозиційні* коди [2 – 10]. У одиничних кодах для подання інформації завжди використовують один і той же елементарний символ, або імпульс, і тому кодові комбінації відрізняються одна від одної лише кількістю символів. Одиночні коди є найпростішими, але вони мають і низку недоліків, зокрема низьку завадостійкість та невизначеність у часовому діапазоні. Саме з використанням одиничних кодів побудовані алгоритми роботи машини Тюрінга, розглянутої у підрозділі 7.3.3 другої частини посібника [50].

Найбільше поширення у сучасних інформаційних системах отримали двійкові коди. Це пов'язано з тим, що формування двійкових сигналів та їхнє дешифрування легко здійснюється апаратно цифровими пристроями, які мають два стабільних стани. Крім того, сьогодні існує безліч апаратних пристроїв пам'яті, які дозволяють зберігати та обробляти інформацію саме у двійковій формі [1 – 5]. Слід також мати на увазі простоту оброблення двійкової інформації через здійснення відповідних арифметичних та логічних операцій, оскільки саме двійкова арифметика у найбільшій мірі розвинена та пристосована для розв'язування прикладних задач з використанням комп'ютерної техніки. Способи кодування двійкових сигналів цілком відповідають теорії скінченних автоматів, розглянутій у підрозділі 7.3 другої частини посібника. Багатопозиційні коди є досить складними, але й високоефективними, тому вони знаходять все більше впровадження у сучасних системах зв'язку та інформаційних електронних системах. Способи формування багатопозиційних кодів розглядатимуться у підрозділі 1.4 та у розділі 3 цієї частини посібника.

Розрізняють *натуральні* (англійський термін – natural code) та *коректувальні* двійкові коди, коректувальні коди іноді також називають *завадостійкими* (англійський термін – **F**orward **E**rror **C**orrection, **FEC**). Класифікація коректувальних кодів є досить складною (рис. 1.2) внаслідок великої кількості наявних способів їхнього подання. Приклади побудови коректувальних кодів будуть наведені у розділах 2 та 3 цієї частини

посібника.

Як видно із наведеної діаграми, натуральні коди розділяють на рівномірні (англійський термін – equal-length code) та нерівномірні (англійський термін – variable-length code). Для рівномірних двійкових кодів характерним є передавання повідомлень однаковою кількістю елементарних символів, нулів або одиниць. Прикладами рівномірних кодів є відомий телеграфний код Боде, а також комп'ютерний код ASCII. У разі використання нерівномірних кодів число елементарних символів в процесі передавання елементів алфавіту є змінним. Найвідомішим прикладом такого типу кодів є телеграфний код Морзе. Зрозуміло, що головною особливістю нерівномірних кодів є необхідність передавання спеціальних розділювальних символів між елементами абетки. Наприклад, у морзянці такими символами є короткі паузи між буквами та довгі між словами. Іноді в електронних системах, у разі використання нерівномірних кодів, використовують засоби синхронізації повідомлень, які розглядалися у підрозділі 4.4 першої частини посібника [1]. До нерівномірних кодів також відносяться ефективні коди, які розглядатимуться у підрозділі 1.5, проте у сучасних алгоритмах ефективного кодування засоби синхронізації не використовуються.

Головною задачею ефективного кодування є мінімальне завантаження каналу зв'язку згідно із інформаційними оцінками, наведеними у підрозділі 1.1 та розділі 4 першої частини посібника [1]. Згідно із цією задачею кодування коди елементів повідомлення мають різну розрядність, і, відповідно, різну тривалість передавання. Головні принципи побудови ефективних кодів розглядатимуться у підрозділі 1.5.2 цієї частини посібника. Головним теоретичним підґрунтям для оцінки параметрів ефективних кодів є теорема Шеннона для каналу без завад. Основними типами ефективних кодів, які використовуються у сучасних системах зв'язку, є код Шеннона – Фано та код Хаффмена, які розглядатимуться у підрозділах 1.5.4 та 1.5.5 цієї частини посібника.

Як рівномірні, так і нерівномірні коди мають свої переваги та недоліки.

Рівномірні коди відрізняються більшою упорядкованістю, і тому за допомогою таких кодів простіше забезпечити передавання даних без помилок. Це пов’язано із двома причинами. По-перше, рівномірність кількості символів для кожного елементу абетки завжди забезпечує їхнє синхронне передавання, і у більшості випадків немає потреби впроваджувати додаткову часову синхронізацію між приймальним та передавальним обладнанням. По-друге, рівномірний код зазвичай забезпечує високу завадостійкість, оскільки відсутність будь-яких елементів символу, що передається, призводить до зміни часу його передавання, а ця неточність відразу легко фіксується приймальним обладнанням. Проте нерівномірні цифрові коди також мають свої переваги. Вони є більш економічними, і за умови їх використання можна передати інформацію за менший час. Тому їх у сучасних системах передавання інформації дуже широко використовують для стохастичного кодування, сутність якого полягає у тому, що найбільш ймовірні символи алфавіту кодуються короткими послідовностями, що дозволяє значно підвищити швидкість передавання довгих повідомлень. Саме таким чином формуються ефективні цифрові коди, які розглядатимуться у підрозділі 1.5 цієї частини посібника.

Як рівномірні, так і нерівномірні коди знайшли впровадження у стандартах сучасних систем зв’язку та в електронних системах, проте існують відповідні особливості щодо їх використання. Рівномірні коди, через їх високу упорядкованість та орієнтованість на коректну роботу із апаратним забезпеченням, зазвичай використовують на фізичному та канальному рівнях моделі OSI, розглянутої у підрозділі 2.4.2 першої частини посібника [1]. Проте мережний та вищий рівні, які формують інформаційні пакети з урахуванням статистики передавання даних та кількості інформації, яка передається, зазвичай застосовують засоби нерівномірного кодування, особливо у тих випадках, коли розмір пакету є змінною величиною [15 – 23]. Наприклад, саме таким чином працюють мережі стандарту Ethernet та більшість стандартів систем безпроводового зв’язку [15 – 23]. Більш

досконало стандарти побудови комп'ютерних мереж розглядатимуться у четвертій частині посібника.

За формою подання у пристроях оброблення інформації та у каналах передавання даних розрізняють паралельні та послідовні двійкові коди. У разі послідовної форми подання елементарні сигнали, із яких створюється кодова комбінація, надсилаються однією лінією зв'язку послідовно у часі, і як правило, вони відділяються певним часовим інтервалом. Саме таким чином здійснюється передавання пакетів у стандарті проводових комп'ютерних мереж Ethernet. У разі використання паралельної форми подання інформації елементарні сигнали передаються одночасно, і це здійснюється або через використання відповідної кількості каналів зв'язку, або через зміну параметрів сигналів, наприклад їх частоти або фази. Приклади сигналів із багатопозиційною фазовою кодоімпульсною модуляцією були розглянуті у підрозділі 5.4 першої частини посібника [1]. Якщо у проводових системах передавання інформації використання паралельних систем зв'язку пов'язане із великими матеріальними затратами на створення багатоканальних високоякісних ліній передавання, а також складних систем мультиплексування та демультиплексування даних у погоджених потоках, то у системах безпроводового зв'язку паралельне передавання інформації значно спрощується за рахунок використання принципу розділення радіоканалів за частотою або за часом. Майже всі способи розділення сигналів, які розглядалися у розділі 7 першої частини посібника [1], використовуються сьогодні в електронних засобах кодування сигналів. Наприклад, у комп'ютерних системах багатопозиційна кодоімпульсна фазова модуляція використовується у стандартах USB, Bluetooth, а у безпроводових комп'ютерних мережах – у стандартах Wi-Fi та WiMAX. Більш досконало стандарти сучасних інформаційних електронних систем та систем зв'язку розглядатимуться у четвертій частині посібника.

Зрозуміло, що у разі реалізації технологій багатоканального зв'язку у безпроводових мережах слід враховувати закони інтерференції радіохвиль,

розглянуті у підрозділі 2.3.3 першої частини посібника [1]. Проте використання сучасних частотних та лінійних фільтрів [1, 49] зазвичай дозволяє розділяти сигнали різної частоти, а для уникнення помилок застосовуються методи розділення сигналів в ортогональному просторі, які розглядалися у підрозділах 4.6 та 5.5 першої частини посібника [1], а також методи завадостійкого кодування, які розглядатимуться у розділах 2 та 3 цієї частини посібника.

Головними параметрами натуральних цифрових кодів є розрядність коду n та кількість символів абетки N [1, 2]. Саме ці параметри визначають, скільки рівнів сигналу може відобразити відповідний цифровий код. У загальному випадку для багатопозиційного коду кількість станів, які відображуються, визначається як:

$$N = m^n, \tag{1.2}$$

де m – кількість сигналів для кодування інформації, n – розрядність коду. Зрозуміло, що формула (1.2) є наслідком формули для кількості розміщень із повтореннями (1.1), яка відома із основ комбінаторики [61, 62, 74, 75]. Відповідно, для двійкових сигналів:

$$N = 2^n. \tag{1.3}$$

Завадостійкі коди відрізняються від натуральних тим, що кількість розрядів в них є більшою, ніж у початковому числі, і за рахунок цього такі коди дозволяють знаходити та виправляти помилки в інформації, яка передається. Тому розряди у завадостійких кодах діляться на *інформаційні*, які використовуються для передавання необхідної інформації, та *контрольні*, які використовуються для виявлення та виправлення помилок. Взагалі завадостійкі коди є одним із найбільш ефективних засобів забезпечення високої імовірності безпомилкового передавання інформації за умови наявності завад у каналі зв’язку. Теорія завадостійкого кодування базується на двох головних теоретичних засадах. Перша із них пов’язана із інформаційними оцінками, які у загальному вигляді розглядалися у першій частині посібника. Тут головним принципом, на якому базується вся теорія

завадостійкого кодування, є теорема Шеннона про передавання інформації у каналі зв’язку за умови наявності завад. Ця теорема розглядатиметься у підрозділі 2.1 цієї частини посібника. Другою теоретичною підставою для формування завадостійких кодів є математичні засади, зокрема теорія чисел, теорія дискретних груп та поліномів. Теорія чисел та поліномів використовується для формування рівномірних числових кодів, а теорія груп – для блокових, або групових. Відповідний математичний апарат, який використовується під час побудови завадостійких кодів, розглядався у другій частині посібника у розділах 2, 3, 4 та 5 [48]. Більш повна класифікація завадостійких кодів та головні їх параметри розглядатимуться у підрозділі 2.1 цієї частини посібника.

Серед завадостійких кодів розрізняють коди, які виявляють помилки та коди, які їх виправляють. Серед кодів, які виявляють помилки, розрізняють коди із перевіркою на парність, кореляційний манчестерський код, інверсний код, уніполярний код із поверненням до нуля та уніполярний код без повернення до нуля [2 – 6, 8]. Проте слід відзначити, що на відміну від кодів із перевіркою на парність манчестерські та уніполярні коди не мають контрольних розрядів, і їх можна розглядати просто як спосіб кодування сигналів у каналі зв’язку, якому відповідає не завадостійке, а натуральне кодування. Тому уніполярні та манчестерські коди розглядатимуться у підрозділах 1.2.4 та 1.2.5 цієї частини посібника, а код із перевіркою на парність – у розділі 2.

Як видно із діаграми, яка наведена на рис. 1.2, завадостійкі коди діляться на блокові те неперервні [2 – 6, 8, 10]. У блокових кодах кожному символу повідомлення відповідає послідовність інформаційних і контрольних символів, і кількість цих символів є завжди незмінною. У цьому разі кодування та декодування кожного блоку здійснюється незалежно. Залежно від способів внесення надлишкових розрядів розрізняють рівномірні та нерівномірні блокові коди. На практиці частіше використовуються рівномірні блокові коди, у яких як положення, так і призначення

контрольних символів завжди є однаковими. Роль інформаційних і контрольних символів у рівномірних блокових кодах є завжди чітко визначеною. Тому код кожного символу має однакову розрядність *n* і в таких системах зв'язку простіше використовувати засоби синхронізації, описані у підрозділі 4.4 першої частини посібника [1]. У нерівномірних блокових кодах чітке ділення символів на інформаційні та контрольні відсутнє, тому кількість символів на елемент абетки, а також розрядність коду *n*, можуть бути змінною величиною.

Рівномірні, або роздільні блокові коди, згідно із діаграмою, наведеною на рис. 1.2, діляться на систематичні коди (англійський термін – systematic code) та несистематичні, із яких найбільш поширеними є систематичні коди [2 – 6, 8]. В систематичних кодах, які також називаються лінійними, контрольні та інформаційні символи не тільки знаходяться на відповідних позиціях кодової комбінації, але й правила формування контрольних символів коду також є обмеженими і базуються на операціях лінійної алгебри. Відповідний математичний апарат розглядався у розділах 4 та 5 другої частини посібника. Серед систематичних кодів найбільш часто використовуються код Хеммінга, циклічний код, коди Боуза – Чоудхрі – Хоквінгема (БЧХ) та код Ріда – Соломона, які розглядатимуться у розділах 2 та 3 цієї частини посібника.

Ітеративні коди (англійський термін – iterative code), які також називають багатомірними, є різновидом систематичних (рис. 1.2). Відмінною рисою ітеративних кодів є те, що в них операція кодування здійснюється над сукупністю інформаційних символів, а не над одним символом. Тому лінійні операції в ітеративних кодах проводяться не над окремими числами, а над дискретними групами. Відповідний математичний апарат дискретних груп Галуа розглядався у другій частині загального посібника. До ітеративних кодів належать коди БЧХ та код Ріда – Соломона, які розглядатимуться у розділі 3 цієї частини посібника, а також коди Голда, які будуть розглянуті у підрозділі 1.4.

Згідно із діаграмою, наведеною на рис. 1.2, різновидом рівномірних кодів є неперервні коди. Основою побудови неперервних кодів є те, що операції кодування та декодування послідовностей символів здійснюються неперервно у часі, без їх ділення на окремі блоки. Формування контрольних символів у неперервних кодах здійснюється з використанням рекурентних співвідношень, тому такі коди також називають рекурентними, або ланцюговими. Основні алгоритми формування рівномірних лінійних кодів були розглянуті у п'ятому розділі другої частини посібника [48].

У теорії кодування, як окремий випадок неперервних кодів, розглядаються згорткові коди (англійський термін – convolution code), які, згідно із діаграмою, наведеною на рис. 1.2, належать до групи неперервних кодів [2 – 6, 8]. Відмінною рисою згорткового коду є те, що кодер містить один або декілька розрядів пам'яті для збереження інформаційних символів. Тобто запам'ятовування інформації, яка передається, є одним із головним принципів формування згорткового коду, а процеси кодування та декодування здійснюються неперервно. Відмінною рисою побудови алгоритмів згорткового коду є те, що розрядність кодової послідовності, яка формується, залежить не лише від кількості інформаційних розрядів, що передавалися протягом останнього відрізку часу, але і від кількості розрядів, які були передані у попередні інтервали часу та зберігаються у пам'яті кодера. Принципи побудови згорткових кодів головним чином базуються на методах теорії ймовірностей. Відповідний математичний апарат розглядався у другій частині цього посібника, а приклади алгоритмів побудови згорткових кодів будуть наведені у підрозділі 3.6 цього посібника. Серед згорткових кодів найбільш поширеними є коди Вітербі та турбокоди [2 – 6, 30, 33].

Щодо методів оброблення інформації та її збереження, то тут сьогодні використовують переважно паралельні коди. Наприклад, загальновідомо, що елементарна одиниця інформації в комп'ютері під час проведення обчислень становить не 1 біт, а 1 байт, а сьогодні у персональних комп'ютерах використовують мікропроцесори із розрядністю 32 або 64 [44 – 46].

Використання методів паралельного передавання та паралельної обробки інформації значно підвищує швидкодію та ефективність сучасних електронних систем [45]. Проте вагомою альтернативою паралельних систем обробки інформації є використання багатопозиційних кодів. Способи пошуку інформації у рядках із великою кількістю символів розглядалися у підрозділі 1.4 другої частини посібника [48]. Відповідні стандарти побудови сучасних систем зв'язку розглядатимуться у четвертій частині посібника.

1.2 Способи кодування цифрових сигналів в електронних системах та каналах зв'язку

1.2.1 Потенціальне та імпульсне кодування

Існує два головних способи подання цифрових сигналів, відповідно до яких розрізняють способи їхнього кодування у каналах зв'язку. Одним із таких способів, який був розглянутий у підрозділі 5.4 першої частини посібника, є амплітудна, частотна та фазова кодоімпульсна модуляція [1]. Як було показано, сутність цього способу кодування сигналів полягає в тому, що для формування двійкового сигналу змінюється амплітуда, частота або фаза синусоїдального сигналу, який передається через канал зв'язку. Такі сигнали також називають радіоімпульсами [1 – 5]. Зазвичай такий спосіб кодування двійкових сигналів використовують в тому випадку, коли для їхнього передавання застосовують існуючі аналогові канали зв'язку. Наприклад, саме таким чином організований синхронний модемний зв'язок між комп'ютерами [35, 36, 38]. Недолік способу синхронного зв'язку між комп'ютерами через телефонний канал полягає в тому, що, як було відмічено у підрозділі 4.4 першої частини посібника, зв'язок у комп'ютерних мережах бажано організовувати за принципом комутації пакетів, а не за принципом комутації каналів, який притаманний телефонії. Щоб проілюструвати незручність синхронного модемного зв'язку між комп'ютерами, наведемо такий доречний приклад. Припустимо, що користувач, використовуючи

синхронний модемний зв'язок, хоче прочитати електронну пошту та відповісти на всі листи. Для цього йому необхідно, по перш за все, відключити телефон та підключити свій комп'ютер до телефонної лінії зв'язку. Тобто, за умови використання синхронного модемного зв'язку, на той час, коли користувач буде працювати з електронною поштою, телефон має бути відключеним. Другий недолік полягає в тому, що час приймання електронної пошти та відсилання відповідей на отримані електронні листи, навіть за умови не дуже високої швидкості передавання інформації 64 кБіт/с, є дуже незначним, порівняно із тим часом, протягом якого користувач буде писати ці відповіді, а канал зв'язку у цей час також буде зайнятим. Тобто, як відмічалось у підрозділі 4.4 першої частини посібника, використання принципу комутації каналів для комп'ютерного зв'язку не є неефективним з тієї причини, що час зайнятості каналу зазвичай є незначним, а необхідність зв'язку виникає спонтанно у випадкові моменти часу. Для забезпечення синхронного модемного зв'язку використовувався протокол передавання даних X25, проте сьогодні такий спосіб передавання інформації вже майже не вживається [15 – 23, 35].

Із-за вказаних вище причин для передавання даних у комп'ютерних мережах був обраний принцип CSMA/CD, оснований на використанні великою кількістю робочих станцій одного каналу зв'язку та на часовому розділенні сигналів TDM [28, 29, 34, 38, 58]. Цей принцип був описаний у розділі 7 першої частини посібника [1].

Тобто, сьогодні синхронний модемний зв'язок між комп'ютерами використовується досить рідко, проте, з урахуванням переходу на цифрову телефонію, широке впровадження знаходить *асинхронний модемний зв'язок* (англійський термін – **A**symmetric **D**igital **S**ubscriber **L**ine, **ADSL**) [15 – 23]. Асинхронному модемному зв'язку не притаманні описані вище недоліки, користувач комп'ютерної мережі може не відключати телефон від лінії зв'язку, а сама лінія зв'язку буде використовуватися комп'ютером лише в тому випадку, коли іде передавання або приймання інформації. Для

кодування цифрової інформації у телефонних каналах зв'язку використовуються методи кодоімпульсної модуляції, проте головним принципом організації зв'язку тут є не комутація каналів, а комутація пакетів [1, 15 – 23].

Цифрові сигнали з кодоімпульсною модуляцією сьогодні використовуються не лише для модемного зв'язку, а також у таких технологіях передавання інформації, як USB та Bluetooth. Вони також знаходять використання у безпроводових мережах [28, 29, 47]. У цих технологіях передавання інформації використовується спосіб багатопозиційної фазової кодоімпульсної модуляції сигналів, який відрізняється тим, що за рахунок великої кількості рівнів кодування він забезпечує високошвидкісне передавання інформації однією лінією зв'язку за умови підвищеної захищеності від завад. Переваги цього способу кодування багатопозиційних сигналів були розглянуті у підрозділі 5.5.2 першої частини посібника [1].

У разі використання цифрових каналів зв'язку застосовується цифрова форма подання сигналів. За умови застосування цифрових сигналів загалом використовується два головних типа кодування – потенціальне та імпульсне. Форми сигналів, які формуються для їхнього передавання цифровою лінією зв'язку за умови використання потенціального та імпульсного кодування, наведені на рис. 1.3. Більш простим є потенціальне кодування, у разі використання якого зазвичай одиниці відповідає високий рівень напруги, а нулю – низький (рис. 1.3, а). Недоліком такого засобу подання є складність розпізнання нульового значення сигналу за умови впливу потужної завади. У разі використання імпульсного кодування (рис 1.3, б) для подання цифрової інформації застосовуються імпульси однієї або різної полярності. Такий спосіб кодування є більш завадостійким, але він потребує використання цифрових ліній зв'язку з більшою пропускною здатністю. Такі сигнали також називають відеоімпульсами [1 – 5, 8].

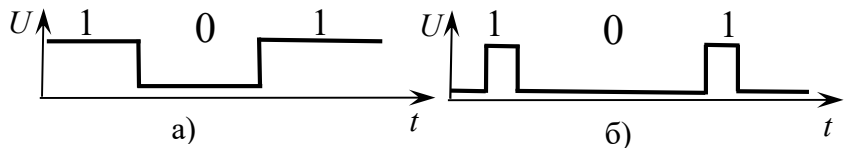


Рис. 1.3. Форма подання цифрової інформації у разі використання потенціального (а) та імпульсного (б) кодування

Недоліком способу полярного кодування є необхідність використання додаткових засобів синхронізації, які були описані у підрозділі 4.3 першої частини посібника. На відміну від цього, одним із можливих варіантів імпульсного кодування є внесення відповідних засобів синхронізації у послідовність імпульсів, яка формується. Одним із таких способів кодування сигналів є манчестерські коди, які будуть розглянуті у підрозділі 1.2.4. Крім цього, потенціальні та імпульсні коди у випадку, коли для кодування одиниці використовуються високий рівень сигналу, а для кодування нуля – низький рівень, як це показано на рис. 1.3, мають ще один суттєвий недолік. Він полягає в тому, що такі сигнали завжди мають у своєму спектрі постійну складову, фільтрація якої на вихідних ємностях передавального пристрою та на вхідних ємностях приймача завжди призводить до суттєвих спотворень форми сигналу і, відповідно, до спотворень інформації, яка передається через канал зв’язку згідно із структурною схемою, наведеною на рис. 2.41 першої частини посібника [1]. Фільтрувальні ємності завжди ставлять у приймальних та передавальних пристроях для уникнення впливу завад від джерела живлення на канал зв’язку. Тому у реальних цифрових системах частіше використовують інший спосіб кодування, згідно з яким рівню одиниці відповідає позитивне значення потенціалу $U_{\max}$, а значенню нуля – негативне значення потенціалу $U_{\min}$. Такі сигнали також значно легше розпізнавати на

фоні завад, оскільки, згідно з оцінками, наведеними у розділі 4 першої частини посібника, вони мають більшу середню потужність. Відповідні часові діаграми для потенціального коду наведені на рис. 1.4, а, а для імпульсного – на рис. 1.4, б.

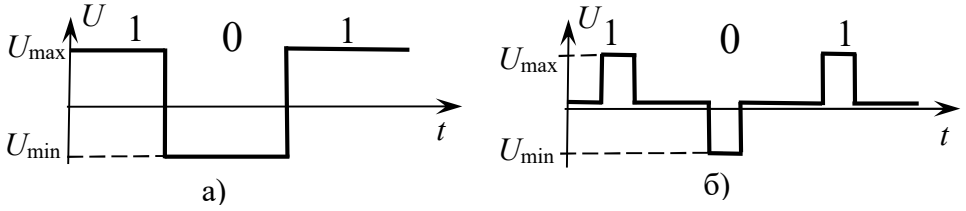


Рис. 1.4. Форма подання цифрової інформації у разі використання дворівневого потенціального (а) та імпульсного (б) кодування із зміною полярності сигналу

Далі будуть розглянуті способи формування потенціальних та імпульсних кодів сигналів, які використовуються в сучасних електронних системах, а також способи кодоімпульсної модуляції сигналів.

1.2.2 Цифрові сигнали із фазовою модуляцією та їхні спектри

Перед вивченням цього підрозділу необхідно повторити розділ 5 першої частини посібника, а також підрозділи 1.2.3.2 та 2.4 другої частини посібника

Розглянемо тепер спектри двійкових сигналів із фазовою модуляцією, які у загальному вигляді аналізувалися у розділі 5 першої частини посібника. Зокрема, спектри таких сигналів наведені у першій частині посібника на рис. 5.11 та 5.12 [1].

Спочатку розглянемо сигнал $f(t)$ як потенціальний код для двійкової числової послідовності 10010110. Відповідний сигнал для уніполярного коду NRZ наведений на рис. 1.5, а, а для біполярного коду – на рис. 1.5, б.

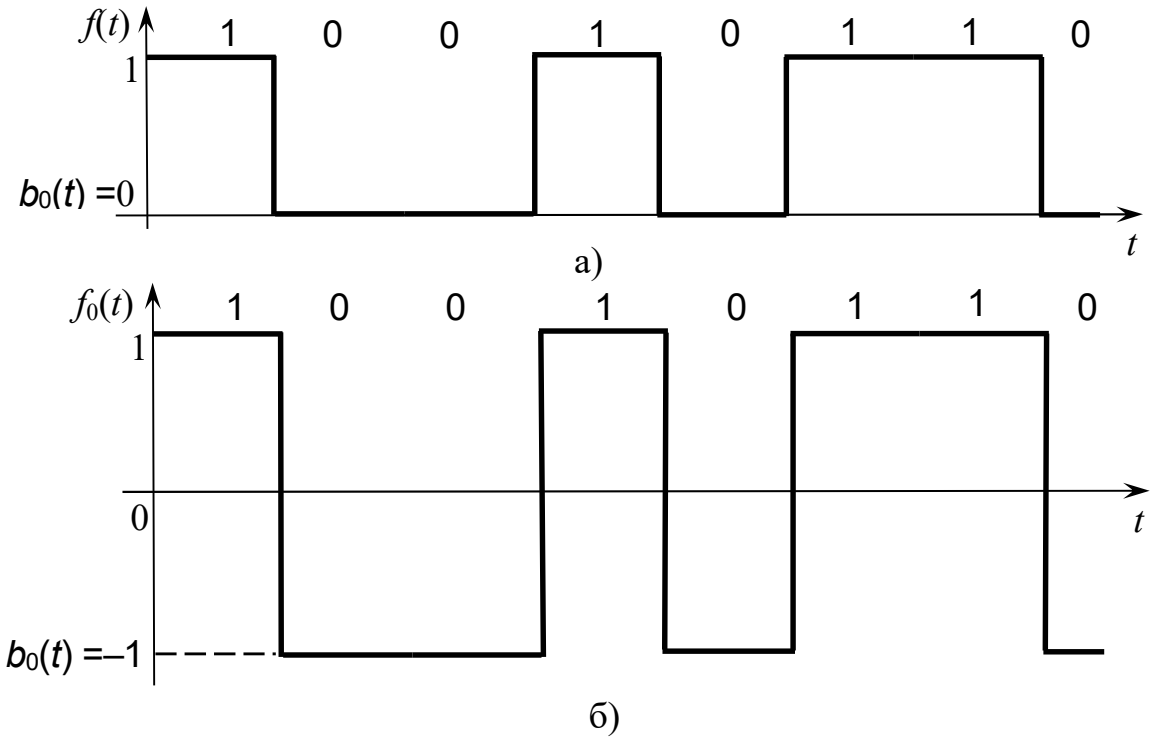


Рис. 1.5 Уніполярний (а) та біполярний (б) сигнали для кодової послідовності 10010110

Зрозуміло, що для уніполярного сигналу $b_0(t) = 0$, а для біполярного – $b_0(t) = -1$. Подамо цифровий сигнал $b_0(t)$ на фазовий модулятор, який забезпечує зсув фази на π. Відповідна схема кодування сигналу наведена на рис. 1.6 [67].

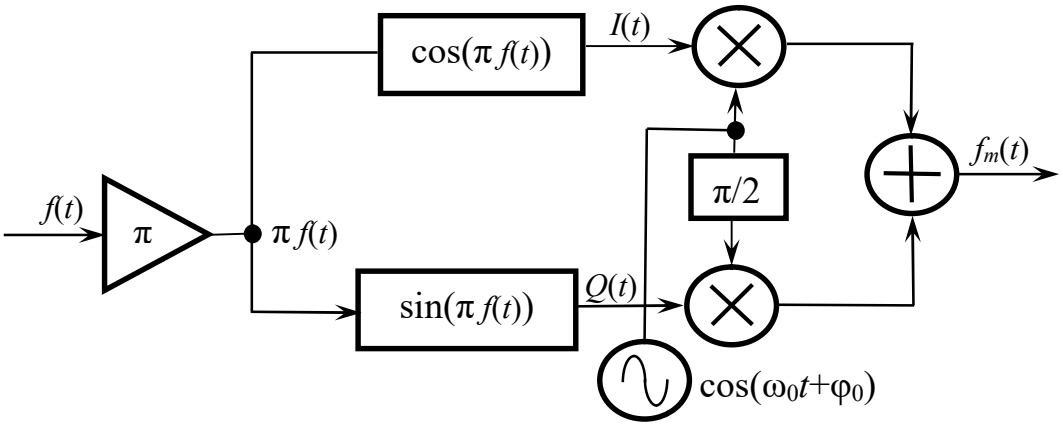


Рис. 1.6 Формування модульованого сигналу $f_m(t)$ з використанням схеми фазового модулятора

Згідно із схемою модулятора, яка наведена на рис. 1.6, для модульованого сигналу $f_m(t)$ можна записати співвідношення [1, 26]:

$$f_m(t) = I(t)\cos(\omega_0 t+\varphi_0) - Q(t)\sin(\omega_0 t+\varphi_0)= f_0(t)\cos(\omega_0 t+\varphi_0). \quad (1.4)$$

Тоді структурна схема фазового модулятора спрощується і її можна подати у вигляді, наведеному на рис. 1.7 [37].

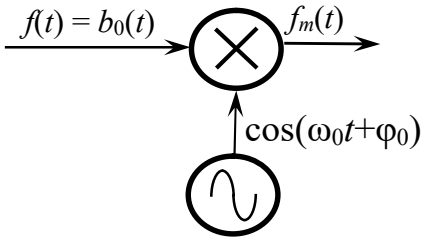


Рис. 1.7 Спрощена структурна схема фазового модулятора

Будемо вважати, що тривалість передавання одного біта інформації складає $T = 1/B_r$, де B_r – швидкість передавання інформації. Початковий сигнал $b_0(t)$, який модулюється, множиться на несне коливання $\cos(\omega_0 t+\varphi_0)$, в результаті створюється модульований за фазою сигнал із зміною фази на π. Відповідний графік для часової форми сигналу із фазовою модуляцією був наведений на рис. 5.5, в, а фазова діаграма для цього сигналу – на рис. 5.8, а першої частини посібника [1]. Усереднений за великий час передавання інформації частотний спектр сигналу із фазовою модуляцією для швидкості передавання інформації $B_r = 20$ кбіт/с наведений на рис. 1.8 [37]. Із рис. 1.8 видно, що частотний спектр сигналу із фазовою модуляцією має головний, широкий пелюсток, який відповідає частоті синусоїдального сигналу $F_0 = \dfrac{2\pi}{\omega_0}$, та бокові пелюстки, амплітуда яких асимптотично зменшується із віддаленням від центрального пелюстка. Відповідні спектри для меандрового сигналу із кодоімпульсною фазовою модуляцією були розраховані у прикладі 5.7 першої частини посібника та наведені на рис. 5.11 та 5.12 [1]. Головні параметри сигналу із фазовою модуляцією наведені на рис. 1.8. Зрозуміло, що ширина кожного пелюстка частотного спектру складає $2B_r$.

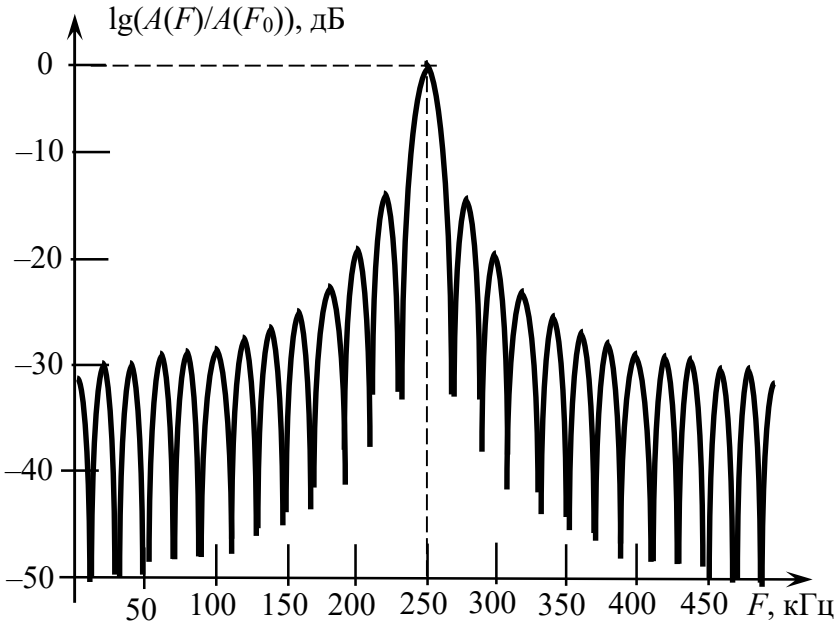


Рис. 1.8 Усереднений спектр сигналу із фазовою модуляцією

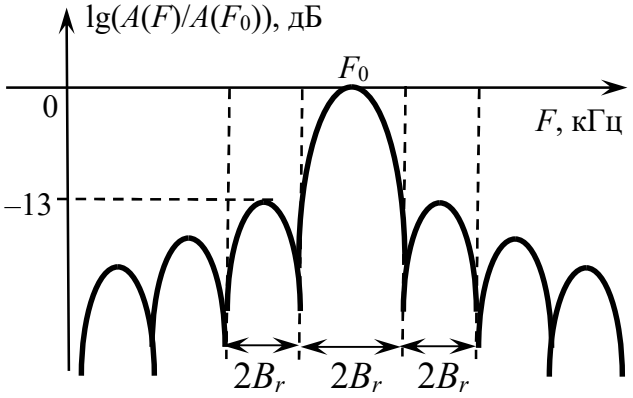


Рис. 1.9 Параметри частотного спектру сигналу із фазовою модуляцією

Недоліком двійкових сигналів із фазовою модуляцією є необхідність когерентного приймання сигналів для визначення зсуву їх фази. Припустимо, що, як на рис. 5.8, а першої частини посібника [1], значенню 0 відповідає фаза $\varphi = 0$, а значенню 1 – фаза $\varphi = \pi$. Відповідна фазова діаграма такого сигналу наведена на рис. 1.10, а. На рис. 1.10 видно, що якщо зсув фази цифрового сигналу під час його приймання є більшим за $\frac{\pi}{2}$, сигнал не

спотворюється (рис. 1.10, б), проте у разі, якщо зсув фази перевищує $\frac{\pi}{2}$, сигнали одиниці та нуля на приймальній стороні приймаються хибно (рис. 1.10, в).

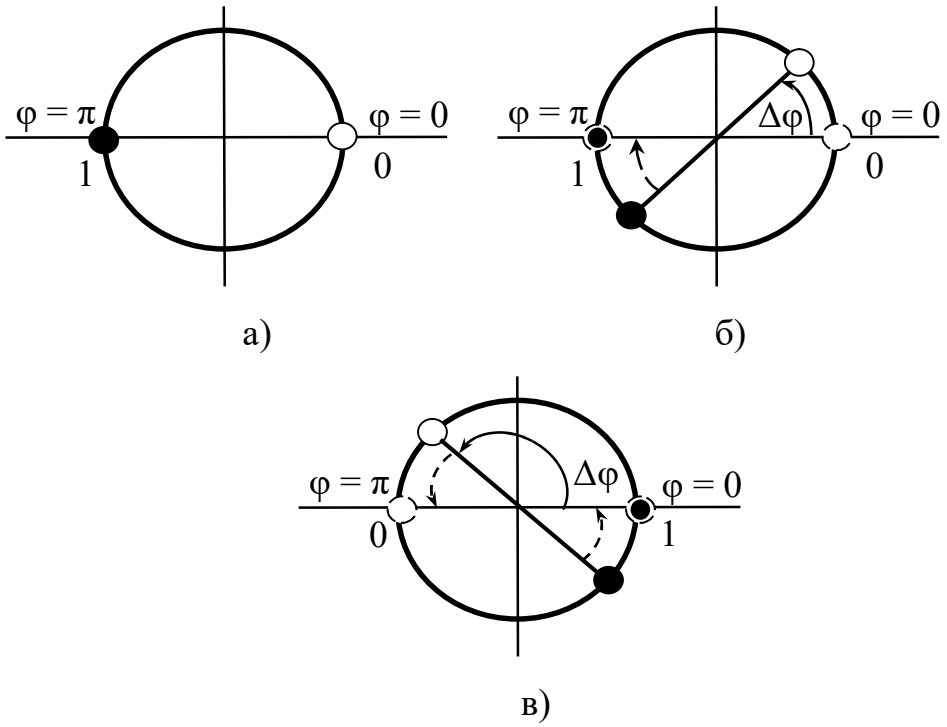


Рис. 1.10 Ілюстрація правильного та хибного приймання сигналу із фазовою модуляцією за умови зсуву фази на приймальній стороні

Це пов’язано із поняттям про відстань між сигналами, яке було розглянуто у підрозділі 4.6 першої частини посібника [1], а також у розділі 5 другої частини посібника [48]. Зрозуміло, що у випадку, наведеному на рис. 1.10, в, відстань від спотвореного сигналу одиниці до реального сигналу нуля є меншою, ніж до реального сигналу одиниці. На фазових діаграмах, наведених на рис. 1.10, суцільними лініями показані значення для прийнятого сигналу, а пунктирними – для обізнаного сигналу. Слід відзначити, що отримані висновки для сигналів із фазовою модуляцією співпадають із теоретичними міркуваннями про те, що, як було відмічено у підрозділі 6.1 першої частини посібника, погіршення роботи цифрових систем зазвичай

здійснюється стрибкоподібно і має не поступовий, а пороговий характер.

Для уникнення хибного приймання інформаційних сигналів у системах зв'язку та інформаційних системах використовують диференціальну фазову модуляцію (англійський термін **D**ifferential **B**inary **P**hase **S**hift**–K**eying, **DBFSK**). Сутність диференціальної фазової модуляції полягає в тому, що кодуються не інформаційні біти, а їх зміна. Структурна схема цифрового каналу передавання інформації, в якому використовується спосіб диференціальної фазової модуляції сигналу, наведена на рис. 1.11 [37].

Із схеми системи зв'язку з диференціальною фазовою модуляцією (рис. 1.11) видно, що початковий інформаційний потік $f(t)$ проходить через схему диференціального кодування, після чого здійснюється фазова модуляція двійкового сигналу, а на приймальній стороні сигнал демодулюється з використанням некогерентного фазового демодулятора та диференціального декодера. В результаті на виході системи формується модульований інформаційний потік з диференційною фазовою модуляцією $f_{DPM}(t)$.

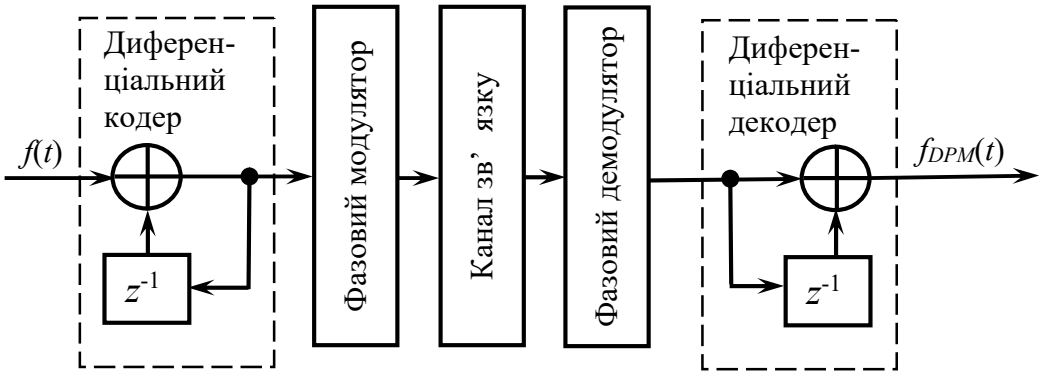
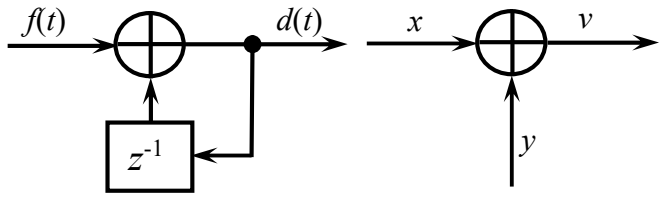


Рис. 1.11 Структурна схема системи передавання інформації з використанням методу диференціальної фазової модуляції

Розглянемо принцип роботи диференційного кодера, схема якого наведена на рис. 1.12.



x	y	v
0	0	0
0	1	1
1	0	1
1	1	0

а) б) в)

Рис. 1.12 Структурна схема диференціального кодера та принцип її роботи

На рис. 1.12, а, видно, що схема диференціального кодера складається із двох блоків: блоку сумування сигналів x та y за модулем 2 (рис. 1.12, б), який виконує операцію «виключного або» XOR [13, 14, 25, 39, 40, 44, 46], та блоку затримки сигналу на 1 біт z^{-1}. Принцип роботи логічної операції XOR легко зрозуміти із таблиці, наведеної на рис. 1.12, в. Якщо два біти, які надходять на вхід блоку XOR, співпадають, результуючий сигнал v дорівнює нулю, а якщо вхідні біти різняться, результуючий сигнал дорівнює одиниці. Логічна схема XOR є стандартною для комп'ютерної логіки і її легко реалізувати як на апаратному рівні [46], так і засобами програмування [39, 40]. Принцип роботи диференціального кодера на прикладі кодової послідовності 011100101 показаний на рис. 1.13.

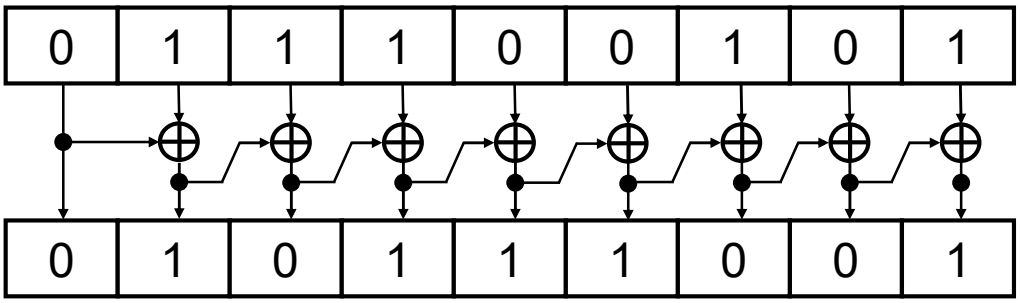


Рис. 1.13 Ілюстрація принципу роботи диференціального кодера

Тобто, як видно із рис. 1.13, початковий бітовий потік 011100101 на виході декодера перетворюється на потік 010111001. Перший біт вихідного

потоку зберігається, а всі наступні біти формуються як результат операції XOR для попереднього вихідного біта та поточного вхідного біта.

Диференціальний декодер виконує зворотну процедуру сумування за модулем 2 поточного та попереднього бітів вхідного потоку, в результаті із закодованої послідовності 010111001 отримуємо початкову послідовність 011100101. Принцип роботи диференціального декодера проілюстрований на рис. 1.14.

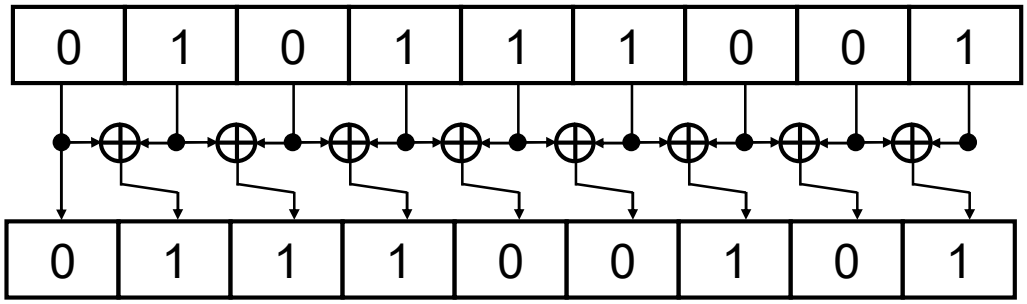


Рис. 1.14 Ілюстрація принципу роботи диференціального декодера

Ефективність використання диференціальної фазової модуляції сигналів легко зрозуміти, якщо розглянути наступний приклад. Припустимо, що вхідний бітовий потік є таким же самим, тобто 011100101, але через зсув фаз у каналі зв'язку виникла помилка і був прийнятий інверсний бітовий потік 101000110. Подамо цю бітову послідовність на вхід диференціального декодера, як показано на рис. 1.15.

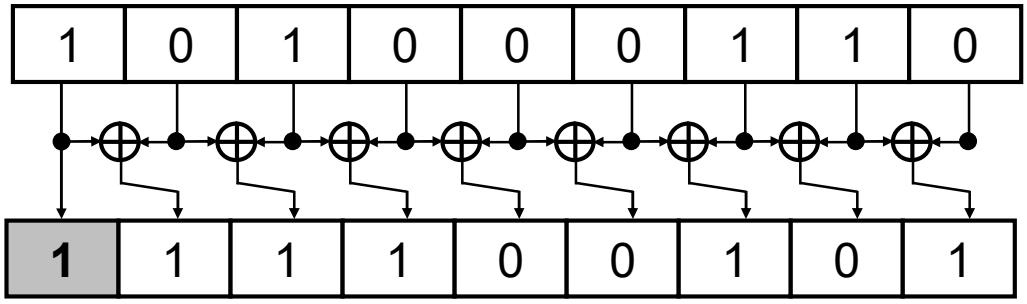


Рис. 1.15 Ілюстрація принципу роботи диференціального декодера за умови інверсії вхідного потоку даних через зсув фази

Порівнявши рис. 1.14 та 1.15 можна зробити висновок, що якщо пропустити спотворений бітовий потік із інверсними даними через диференціальний декодер, початкова інформація майже не спотворюється. Винятком є лише останній біт числа, який на рис. 1.15 зафарбований сірим кольором. Тут символ 0, який був у початковій кодовій послідовності, замінюється на символ 1.

Тобто, використання у приймально-передавальній апаратурі диференціальної фазової модуляції дуже спрощує її апаратну реалізацію та дозволяє уникнути використання складних засобів синхронізації сигналів, описаних у підрозділі 1.4.3 першої частини посібника. Слід відзначити також, що електронні схеми для логічної операції XOR, на основі яких побудовані диференціальні кодери та декодери, є досить простими та швидкодійними цифровими пристроями і їх значно легше реалізовувати та застосовувати у цифровій електронній апаратурі, ніж відповідні засоби синхронізації [37, 38].

Головним недоліком диференціальної фазової модуляції як способу кодування сигналів є збільшення кількості помилок на етапі декодування [37]. У літературі цей ефект називають також «ефектом розмноження помилок» [33, 37]. Із логічної схеми побудови декодера, наведеної на рис. 1.13, зрозуміло, що якщо у одному біті закодованої послідовності виникає помилка, на виході декодера спотворюється не один, а два інформаційних біти. Розглянемо «ефект розмноження помилок» на конкретному прикладі. Припустимо, що, як і в попередніх прикладах, початковий потік даних становить 011100101 і йому відповідає закодований потік 010111001. Проте у цьому разі будемо вважати, що під час передавання закодованої інформації виникла помилка у п’ятому розряді і декодер прийняв спотворену кодову комбінацію 010101001. Тоді на виході декодера будуть спотворені не один, а два розряди початкової кодової комбінації, а саме четвертий та п’ятий розряди. Результат виконання цієї операції над спотвореною кодовою комбінацією показаний на рис. 1.16. Відповідно, замість початкового числа 011100101 маємо спотворену кодову комбінацію 011111101 із двома помилками, у четвертому та п’ятому розрядах.

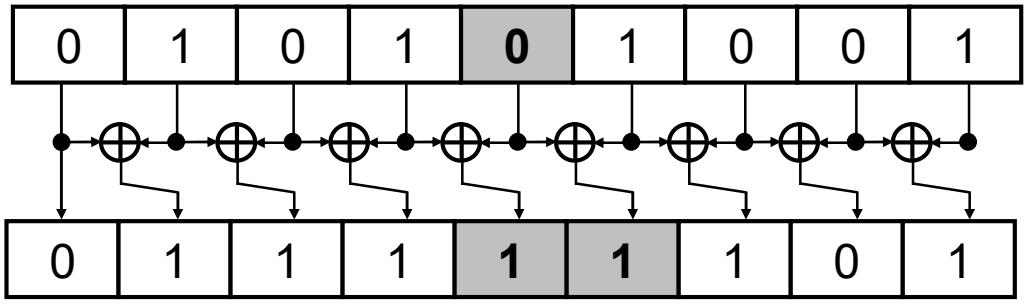


Рис. 1.16 Ілюстрація принципу роботи диференціального декодера у разі виникнення помилки у вхідній бітовій послідовності

Таким чином, двійкові сигнали із фазовою модуляцією мають частотний спектр із шириною пелюстків $2B_r$. Ці сигнали є асинхронними, тому для уникнення помилок розпізнавання сигналу на приймальному обладнанні використовують диференціальну фазову модуляцію, яка дозволяє запобігти помилок розпізнавання фази сигналу. Спосіб кодування, який використовується в диференціальній фазовій модуляції, оснований на сумуванні сигналів за модулем 2, або, інакші кажучи, на використанні логічної функції XOR. Зрозуміло, що в даному випадку однаковий цифровий код для основної і для інверсної кодової послідовності обумовлений властивістю функції XOR. У разі використання цієї логічної функції головне значення має не конкретний біт кодової послідовності, а те, співпадають чи відрізняються сусідні біти цієї послідовності. Наслідком цього і є головний недолік диференціального фазового кодування сигналів. У разі спотворення одного біту закодованої кодової послідовності на виході декодера спотворюється не один, а два розряди початкової кодової комбінації. Уникнути ефекту «розмноження помилок» можна з використанням методів завадостійкого кодування, які будуть розглядатися у розділах 2 та 3 цієї частини посібника.

Слід відзначити, що відмічена властивість логічної функції XOR використовується і для побудови інших логічних кодів двійкових сигналів. Зокрема, саме на використання цієї функції основані алгоритми скрамблювання, які розглядатимуться у підрозділі 1.3.6 цієї частини

посібника.

Імовірність помилки у разі використання фазової модуляції двійкового сигналу обчислюється згідно із теорією сигнально-кодових конструкцій, яка була описана у підрозділі 4.6.2 першої частини посібника [1] та у підрозділі 6.5 другої частини посібника [49]. Відповідне співвідношення можна записати у вигляді [1]:

$$p = \frac{1 - \Phi\left(\sqrt{2}\,\frac{E_б}{N_0}\right)}{2}, \qquad (1.5)$$

де $E_б$ – енергія одного біту сигналу, N_0 – густина спектру шуму, $\Phi(x)$ – функція помилок Гауса. Типова залежність імовірності помилки від співвідношення потужності сигналу із кодоімпульсною фазовою модуляцією до потужності шуму наведена на рис. 1.17 [47]. Код програми, написаний мовою програмування системи MatLab, призначеної для розрахунку імовірності бітової помилки з використанням співвідношення (1.5), наведений у додатку А. Методи оцінки бітових помилок для різних способів модуляції цифрових сигналів розглядалися у підрозділі 6.5 другої частини посібника [49].

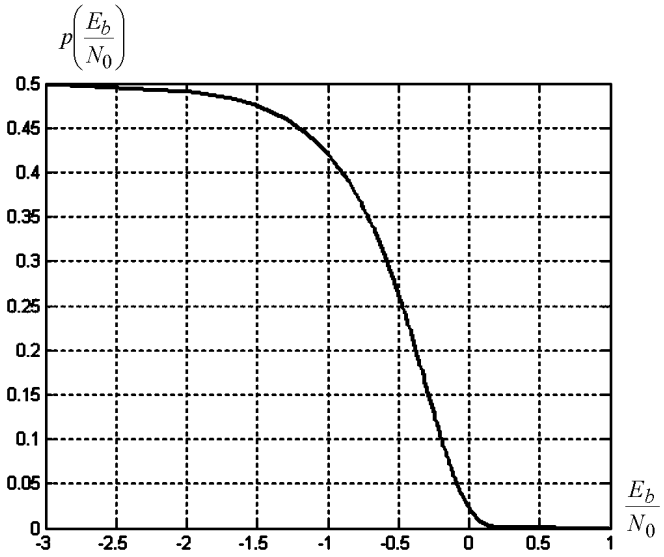


Рис. 1.17 Залежність імовірності бітової помилки від співвідношення потужності сигналу до потужності шуму у децибелах для сигналу із кодоімпульсною фазовою модуляцією

Із рис. 1.13 та 1.14 ясно видно, що відношення між вхідними та

вихідними бітовими послідовностями різницевих кодів є рекурентними. Тому запишемо алгоритми формування та декодування різницевих кодів у математичній формі, що надає можливість реалізувати ці алгоритми у вигляді комп'ютерної програми. Будемо вважати, що X – вектор вхідної бітової послідовності, а Y – вихідної. Із рис. 1.13 зрозуміло, що у цьому разі алгоритм формування різницевого коду можна записати наступним чином:

$$y_1 = x_1;\ y_2 = x_1 \oplus x_2;\ \ y_i = y_{i-1} \oplus x_i,\ i > 3;\ i = 1{:}n. \qquad (1.6)$$

де i – поточна ітерація, n – кількість елементів вхідного вектора X.

Під час описання рекурентних алгоритмів теорії кодування сигналів іноді у різних літературних джерелах існує розбіжність у позначенні вхідних та вихідних векторів кодека та декодека. У технічній літературі з кодування сигналів [30, 33, 55 – 57] для алгоритмів кодування та декодування позначення векторів зберігається. Тобто, вважається, що для кодека вектор X відображається на вектор Y ($X \rightarrow Y$), а для декодека – вектор Y відображається на вектор X ($Y \rightarrow X$). З іншого боку, у математичній літературі [52, 53] зазвичай дотримуються інших позначень. Вважається, що X це завжди початковий вектор, а Y – вектор, на який він відображається, і ці позначення зберігаються як для кодека, так і для декодека. Якщо дотримуватися таких позначень, для перевірки правильності роботи двох пристроїв, кодека та декодека, відносно початкового вектора, необхідно перед декодуванням зробити заміну змінних $X = Y$, а після декодування – зворотну заміну $Y = X$. Проте у цьому разі на виникає питань щодо порядку змінних, оскільки вектор X завжди є аргументом, а вектор Y – функцією цього аргументу. З іншого боку, у разі збереження імен векторів роботу декодека можна розглядати як зворотну функцію до функції кодування, тобто:

$$Y = F(X), \qquad X = F^{-1}(Y) = F^{-1}(F(X)), \qquad (1.7)$$

де $F(X)$ – функція кодування, $F^{-1}(Y)$ – функція декодування. Зазвичай, саме такий підхід із збереженням імен змінних є більш зручним для інженерів електронної техніки, які використовують математичний апарат теорії кодування для проведення своїх прикладних досліджень. Тому надалі у

цьому посібнику, якщо це не буде обумовлено окремо, автори будуть дотримуватись саме таких позначень вхідних та вихідних змінних. Проте під час аналізу особливостей роботи програмних засобів часто зручніше вхідну змінну у будь-якому разі позначати літерою *X*, а вихідну – літерою *Y*. Відношення між елементами множин та їхня однозначність є предметом окремого розділу дискретної математики, вони розглядалися у підрозділі 2.4 першої частини посібника.

Для задачі декодування різницевої послідовності у подальших міркуваннях будемо вважати, що вхідною послідовністю декодека є вектор *Y*, а вихідною, у разі відсутності помилки декодування, має бути початковий вектор *X*. Тобто, будемо користуватися інженерним підходом, основаним на використанні співвідношення (1.7), а не строго математичним підходом, який базується на збереженні імен змінних для функції та її аргументу. Аналіз розбіжностей між вихідним вектором декодека X' та вхідним вектором кодувального пристрою *X* через наявність помилок у роботі електронної апаратури та завад у каналі зв'язку є окремим питанням теорії завадостійкого кодування, яке буде розглянуто у розділі 3.

Вважаючи, що для різницевого декодувального пристрою вхідним є вектор *Y*, а вихідним має бути вектор *X*, алгоритм декодування, який наочно проілюстрований на рис. 1.14, записується наступним чином:

$$x_1 = y_1;\ x_2 = y_1 \oplus y_2;\ \ x_i = y_{i-1} \oplus y_i,\ i > 3;\ i = 1{:}n. \qquad (1.8)$$

Програма diffcode, в якій реалізовані алгоритми різницевого кодування та декодування двійкових послідовностей, задані рекурентними співвідношеннями (1.6) та (1.8), наведена у додатку Б. Програма написана мовою програмування системи MatLab. Відмінною рисою цієї програми є те, що вона створена засобами матричного програмування, розглянутими у навчальних посібниках [13, 14]. Теоретичним підґрунтям цього методу програмування є використання типових арифметико-логічних функцій виду:

$$F_{ал}(x) = F_1(x)\cdot L_1(x) + F_2(x)\cdot L_2(x) + F_3(x)\cdot L_3(x) + ... + F_n(x)\cdot L_n(x), \qquad (1.9)$$

де $F_n(x)$ – арифметичні функції, $L_n(x)$ – логічні функції. Також концепції матричного програмування притаманно застосування спеціальних процедур для формування елементів рекурентних векторів та матриць recvect та recmat [13, 14]. Проте у навчальних посібниках [13, 14] розглядалися особливості матричних обчислень лише для дій із дійсними та комплексними числами, а рекурентні обчислення для двійкових послідовностей у теорії кодування цифрових сигналів мають певні особливості. Розглянемо головні з них.

1. Функції $F_n(x)$ у співвідношенні (1.9) будуються на логічних діях над двійковими послідовностями, основаними на алгебрі Буля, розглянутій у підрозділі 1.2.3.2 другої частини посібника. Зокрема, у рекурентних послідовностях (1.6) та (1.8) використовується логічна функція «виключного або». Тому для двійкової арифметики суттєва різниця між функціями $F_n(x)$ та $L_n(x)$ є іншою, ніж для числових операцій над десятковими числами і для класичних методів функціонального аналізу [11 – 14]. Ця різниця полягає у тому, що функції $F_n(x)$ використовуються для описання логічних дій над бітовими послідовностями, а функції $L_n(x)$ – для описання відповідних логічних умов, за якими здійснюються рекурентні обчислення, задані через функції $F_n(x)$. Наприклад, арифметико-логічний вираз

$$F_{\text{ал}}(x) = (i=1) \cdot x_1 + (i=2) \cdot (x_1 \& x_2) \qquad (1.10)$$

означає, що значення першого елемента вихідної кодової послідовності Y дорівнює першому елементу вхідного вектора X, а значення другого елемента визначається як кон'юнкція першого та другого елементів вхідного вектора. Проте слід відзначити, що в іншому логічне співвідношення (1.10) цілком відповідає правилам формування арифметико-логічних функцій (1.9), розглянутим у [13, 14].

2. Ітераційні дії над двійковими послідовностями зазвичай здійснюються лише за необхідною кількістю ітерацій, і це є єдиний параметр, який має контролюватися. Для дій над раціональними числами, крім кількості ітерацій, важливими є також граничні максимальні та мінімальні

значення елементів рекурентного вектора, які обчислюються. Тому у процедурі обчислення елементів рекурентного вектора recvect, яка розглядалась у навчальному посібнику [14], були введені відповідні перевірки обчислених значень рекурентних послідовностей раціональних чисел, які формуються. У разі, якщо в ході проведення обчислень отримані елементи вектора виходять за межі граничних максимальних або мінімальних величин, ітераційний процес примусово обривається з метою уникнення обчислювальних помилок. Також у цій процедурі передбачене припинення обчислювального процесу у разі, якщо досягнута необхідна точність обчислень, що дозволяє у значній мірі скоротити час проведення числових розрахунків. Такі перевірки спрощують використання процедури recvect для розв'язування практичних завдань обчислювальної математики. Для двійкових послідовностей, які розглядаються, такі перевірки є неможливими, проте вкрай необхідною є перевірка коректності значень елементів вхідного двійкового вектора, які мають дорівнювати 0 або 1.

3. Особливість проведення ітераційних обчислень полягає у тому, що значення елементів вихідного вектора, які будуть обчислені на наступних ітераціях, на поточній ітерації можуть бути невідомими. Наприклад, розглянемо наступний арифметико-логічний вираз:

$$F_{\text{ал}}(x) = (i = 1) \cdot \overline{x_1} + (i = 2) \cdot (x_2 \ \& \ y_1), \qquad (1.11)$$

де x – елементи вхідного вектора, y – елементи вхідного вектора. З математичної точки зору така рекурентна послідовність є цілком коректною. Сутність її полягає в тому, що на першій ітерації обчислюється значення першого елемента вихідного вектора $y_1 = \overline{x_1}$, а на другій ітерації другий елемент вихідного вектора y_2 обчислюється через його попередній елемент y_1 та поточний елемент вхідного вектора x_2 з використанням логічної операції «І». Проблема використання у комп'ютерних програмах подібних арифметико-логічних виразів полягає у тому, що комп'ютер на початкових

ітераціях аналізує значення всього виразу (1.11), і тому на першій ітерації видається помилка про те, що іде посилання на ще необчислений елемент вихідного вектора y_1. У процедурі recvect, розглянутій у навчальному посібнику [14], ця проблема була вирішена через введення додаткових початкових елементів вектора та через зміну індексації у арифметико-логічному співвідношенні (1.9), проте у разі обчислення рекурентних двійкових послідовностей такий підхід не є раціональним та вкрай ускладнює роботу програмістів.

У зв'язку з цим, для реалізації рекурентних алгоритмів двійкової арифметики була написана окрема процедура recvectbin, код якої наведений у додатку Б. Відмінними рисами цієї процедури є наступні.

1. Наявність контролю значень елементів вхідної двійкової послідовності, які мають дорівнювати 0 або 1.

2. Відсутність контролю елементів вихідної послідовності, які обчислюються, за мінімальними та максимальними граничними значеннями.

3. Реалізована процедура автоматичного обрізання рядка арифметико-логічної функції для забезпечення коректності роботи обчислювального алгоритму на поточній ітерації. Наприклад, арифметико-логічний вираз (1.11) для уникнення помилкового посилання на значення елемента y_1 на першій ітерації, можна переписати у такій скороченій формі:

$$F_{\text{ал}}(x) = (i = 1) \cdot \overline{x_1}. \qquad (1.12)$$

Тоді під час проведення обчислень на першій ітерації буде використане співвідношення (1.12), а на другій ітерації – співвідношення (1.11). У цьому разі всі логічні операції на обох ітераціях комп'ютер буде виконувати коректно.

З урахуванням співвідношення для арифметико-логічної функції (1.9), перепишемо рекурентний вираз алгоритму різницевого кодування (1.6) наступним чином:

$$\text{diffcd}(x_1, \ldots, x_n) = (i = 1) \cdot x_1 + (i = 2) \cdot (x_1 \oplus x_2) + \qquad (1.13)$$

$$+(i>2)\cdot(y_{i-1}\oplus x_i),\ i=1{:}n,$$

де n – довжина вхідного вектора X.

Відповідно, рекурентний алгоритм декодека, заданий співвідношенням (1.8), через арифметико-логічну функцію (1.9) можна переписати у вигляді:

$$\text{diffdcd}(y_1,\ldots,y_n)=(i=1)\cdot y_1+(i=2)\cdot(y_1\oplus y_2)+ \quad (1.14)$$
$$+(i>2)\cdot(y_{i-1}\oplus y_i),\ i=1{:}n.$$

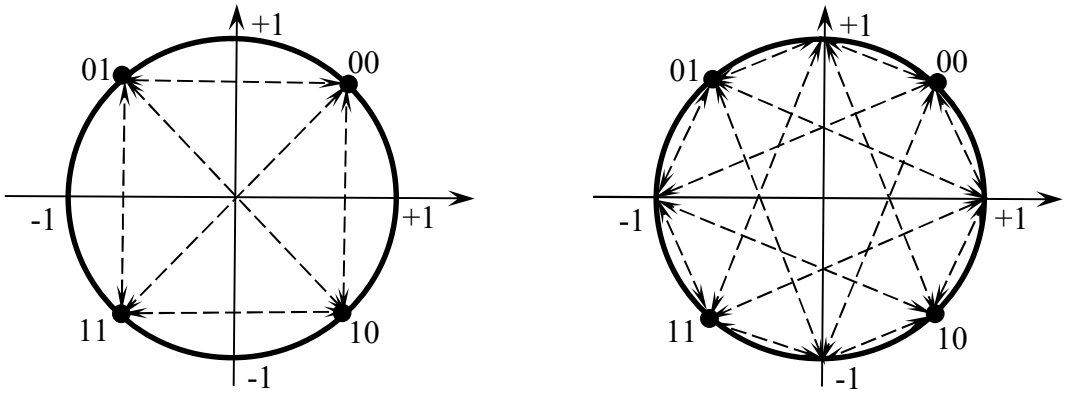
Вихідний вектор функції diffcode, наведеної у додатку В, може бути використаний під час моделювання складних кодувальних електронних систем для подальшої обробки двійкової кодової послідовності, що є важливим для реалізації концепції модульного програмування. Іншою важливою перевагою написаної програми є можливість розв'язування як прямої задачі диференціального кодування, так і зворотної задачі декодування різницевої кодової послідовності. Результати тестування програми diffcode також наведені у додатку Б.

1.2.3 Способи реалізації квадратурної амплітудної модуляції

Перед вивченням цього підрозділу необхідно повторити розділи 4 та 5 першої частини посібника

Узагальнена характеристика методу квадратурної амплітудної модуляції була надана у першій частині посібника у підрозділі 5.5.2. Як було відмічено, сутність цього способу модуляції полягає в тому, що сигнал формується як сума двох сигналів з амплітудною модуляцією, які зсунуті по фазі на кут $\frac{\pi}{2}$. У першій частині посібника на рис. 5.19, в, була наведена діаграма сигналу із квадратурною модуляцією у декартовій системі координат [1]. На рис. 1.18, а, наведена кругова діаграма для такого сигналу. Існує також спосіб диференціальної квадратурної фазової модуляції із зсувом фази $\frac{\pi}{4}$. Додаткове зміщення фази означає, що переходи між символами не проходять через початкову фазу, що зменшує імовірність зміни амплітуди сигналу [28 – 30]. Це суттєво знижує вимоги щодо смуги пропускання каналу зв'язку та

формування імпульсів сигналу. Відповідна фазова діаграма наведена на рис. 1.18, б.



а) б)

Рис. 1.18 Кругова діаграма сигналу із квадратурною амплітудною модуляцією без зсуву фази (а) та із зсувом фази на $\frac{\pi}{4}$ (б)

Тобто, сигнально-кодова конструкція із квадратурною амплітудною модуляцією містить чотири сигнали з амплітудами $+1/\sqrt{2}$ та $-1/\sqrt{2}$ та із фазами, зсунутими на кут $\pi/2$. Такою конструкцією можна закодувати 2 біти інформації. Амплітуда, фаза сигналів, та кодові послідовності, які їм відповідають, наведені у таблиці 1.1.

Таблиця 1.1 – Параметри сигнально-кодової конструкції із квадратурною амплітудною модуляцією

Послідовність бітів	Фазовий зсув	Амплітуда
00	$\pi/4$	$+1/\sqrt{2}$
01	$3\pi/4$	$+1/\sqrt{2}$
11	$-3\pi/4$	$-1/\sqrt{2}$
10	$-\pi/4$	$-1/\sqrt{2}$

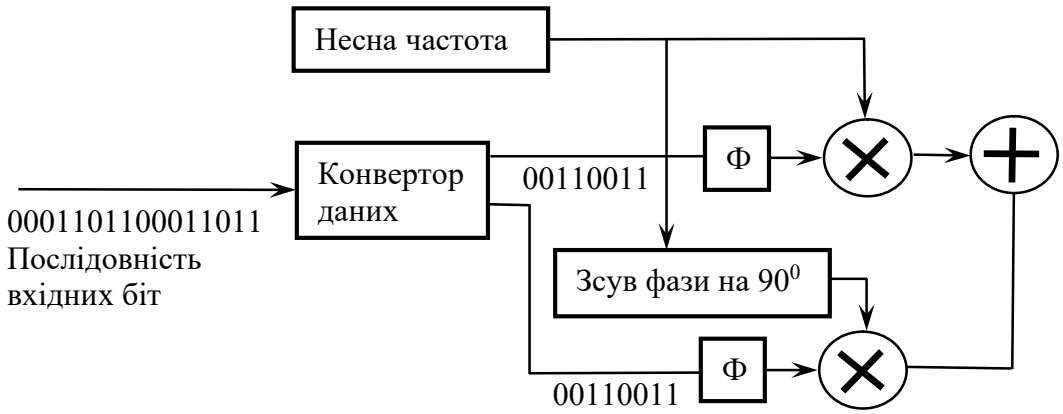
Структурна схема модулятора, який формує сигнал із квадратурною амплітудною модуляцією, наведена на рис. 1.19, а, а відповідна схема

демодулятора – на рис. 1.19, б [28, 29, 37].

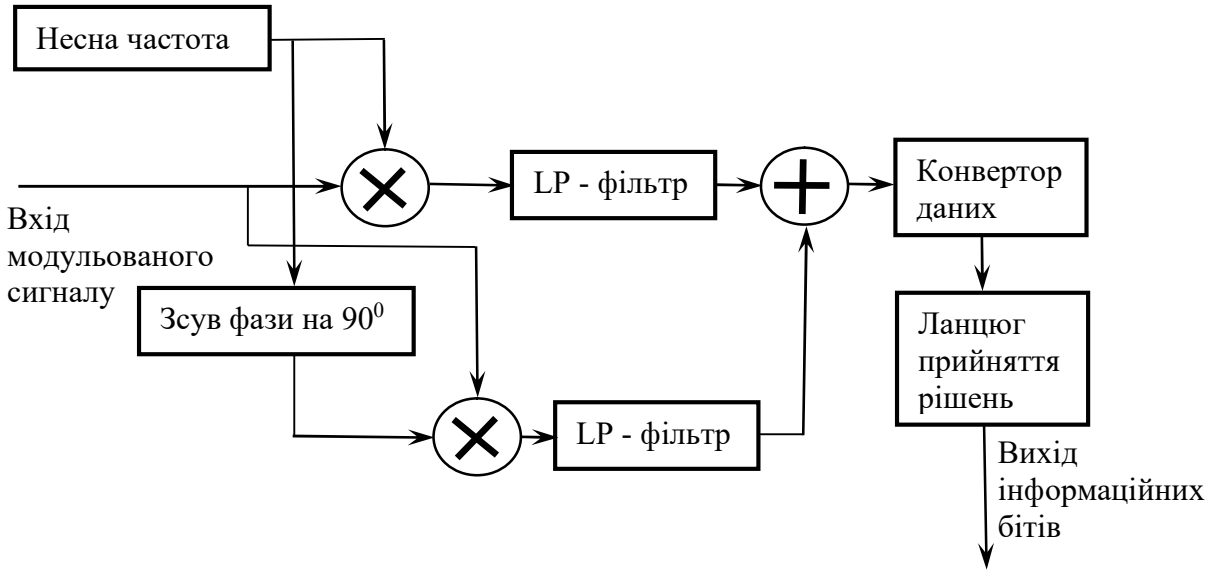
Чистий сигнал з квадратурно-амплітудною модуляцією завжди містить стрибкоподібні переходи на границях бітових компонентів. Виходячи із цього, модулятор містить спеціальні фільтри Ф, які згладжують ці перепади сигналу та формують відповідну форму імпульсу. Як видно з рис. 1.19, а, фільтри розташовані до помножувачів сигналу на несну частоту.

Слід відзначити, що амплітудні та фазові компоненти сигналу з квадратурною амплітудною модуляцією є досить складними для розпізнавання, і в деяких каналах передавання інформації, наприклад у радіоканалах, це приводить до певних труднощів. Як відмічалось у розділі 5.5.2 частини 1 загального посібника [1], розпізнавання сигналів із квадратурною амплітудною модуляцією неможливе без використання засобів синхронізації. Відповідні переваги з цієї точки зору забезпечують інші методи групової модуляції. Два відповідних приклада – це сигнали із зірковою квадратурною модуляцією (англійський термін – Star Quadrature Amplitude Modulation) та сигнали із круговою квадратурною модуляцією (англійський термін – Circular Quadrature Amplitude Modulation) [30, 37]. Проте, з точки зору теорії сигналів, ці два способи модуляції скоріше слід відносити до сигналів із амплітудно-фазовою модуляцією, ніж із квадратурною [26]. Фазова діаграма для зіркової квадратурної модуляції показана на рис. 1.20, а, а для кругової – на рис. 1.20, б.

У разі використання зіркової квадратурної модуляції більша відстань між сигналами забезпечує можливість зменшення потужності сигналу без підвищення імовірності помилок. Зсув фази між сигналами у цій сигнально-кодовій конструкції становить 90^0, що полегшує розпізнавання таких сигналів у дротових каналах зв’язку, де спотворення фази є більш імовірним, ніж спотворення амплітуди сигналу. Тому зіркоподібна діаграма часто використовується у модемних системах зв’язку, відповідні стандарти будуть розглянуті у четвертій частині загального посібника.

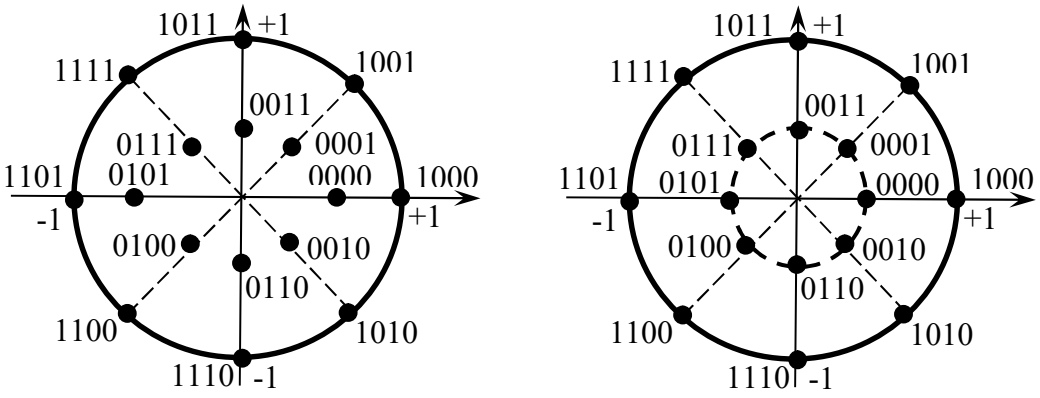


а)



б)

Рис. 1.19 Структурні схеми модулятора (а) та демодулятора (б) для сигналів із квадратурною амплітудною модуляцією



а) б)

Рис. 1.20 Фазові діаграми для зіркової (а) та кругової (б) квадратурної модуляції

Кругова квадратурна модуляція – це аналог багаторозрядної фазової модуляції із визначеною кількістю рівнів сигналу за амплітудою. Для цього виду модуляції використання способу диференціального кодування, який був описаний у підрозділі 1.2.2 цієї частини посібника, дозволяє уникнути необхідності оновлення несної частоти. Як видно з рис. 1.20, способи зіркової та кругової квадратурної модуляції дозволяють кодувати не два, а чотири біти інформації.

Спектр модульованого сигналу із квадратурною модуляцією відповідає спектру сигналу із амплітудною модуляцією, який розглядався у розділі 5 частини 1 загального посібника [1]. З іншого боку, спектр сигналу, який модулюється, визначається із співвідношення (5.67), наведеного у підрозділі 5.6 частини 1 [1]. У теорії цифрових сигналів показано, що чотирирівнева квадратурна амплітудна модуляція має таку ж кодову відстань між сигналами, як і звичайна амплітудна модуляція. Що до багаторівневої квадратурної фазової модуляції, вона має більшу імовірність помилок, ніж двійкова, оскільки, як видно з рис. 1.20, б, сигнальні точки на внутрішньому колі розташовані ближче, ніж на зовнішньому. Проте багаторівнева квадратурна фазова модуляція є більш ефективною, ніж звичайна багаторівнева фазова модуляція [30], особливо за умови великої кількості рівнів m. Наприклад, 64-розрядна амплітудна-фазова модуляція використовується у стандарті USB, а 256-розрядна – у системах цифрового кабельного телебачення [32]. Відповідні стандарти будуть розглянуті у четвертій частині загального посібника.

1.2.4 Імпульсні коди із внутрішнєю синхронізацією

1.2.4.1 Манчестерські коди

Серед імпульсних кодів із самосинхронізацією найчастіше використовуються манчестерські коди. У манчестерських кодах, які відносяться до імпульсних та були використані для передавання інформації у комп’ютерних мережах сімдесятих років XX століття стандарту Ethernet-DIX

[15 – 23], зміна рівню сигналу здійснюється наступним чином. Рівень сигналу змінюється не лише на початку, але і в середині кожного такту, за який передається 1 біт інформації. Зміна рівню на початку такту не несе ніякої інформації, є стохастичною і залежить від попереднього та поточного значення сигналу, а корисну інформацію несе саме зміна сигналу в середині такту передавання двійкового символу, одиниці або нуля. Для манчестерського коду одиниці відповідає перехід з негативного до позитивного рівня сигналу, а нулю – навпаки, перехід від позитивного до негативного рівня.

Іноді у навчальній літературі принцип формування манчестерського коду розглядається дещо інакше, не з точки зору принципу формування сигналу, а з точки зору принципу кодування інформації [2 – 6, 8]. У цьому випадку манчестерський код двійкового числа можна розглядати як кодову послідовність, у якій кожному біту числа, яке кодується, відповідає 2 біти кодової послідовності. Символу 1 відповідає послідовність двох бітів 01, а символу 0 – послідовність 10. Наприклад, числу 11010 відповідає манчестерський код 0101100110. Переваги манчестерської системи кодування полягають у тому, що послідовності символів 11 та 00 є забороненими. З точки зору принципу формування сигналу це означає, що зміна полярності сигналу в середині такту є обов'язковою, і якщо такої зміни полярності не відбувається – це можна розглядати як помилку формування сигналу. Тобто, манчестерський спосіб кодування можна також розглядати як завадостійкий, і ця завадостійкість забезпечується через надлишковість інформації, яка передається. Несумнівною перевагою манчестерського способу кодування є можливість виявлення помилок передавання інформації, проте такі коректувальні властивості сигналу можна забезпечити лише через розширення його частотної смуги та перевантаження ресурсів каналу зв'язку. Взагалі манчестерські коди схожі на код із удвоєнням елементів, який розглядатиметься у підрозділі 2.3 Частотний спектр сигналів із манчестерським кодом розглядатиметься у підрозділі 1.2.9 цієї частини посібника.

Послідовність імпульсів, яка відповідає манчестерському коду, наведена на рис. 1.21, де час $\tau_б$ відповідає бітовій швидкості передавання інформації. Тонкими пунктирними стрілками на рис. 1.21 відмічені перепади сигналу в середині такту, яким і відповідає код значення нуля або одиниці. Із наведеної часової діаграми сигналу зрозуміло, що насправді манчестерський код числа 11010 можна також розглядати як потенціальний код числа 0101100110.

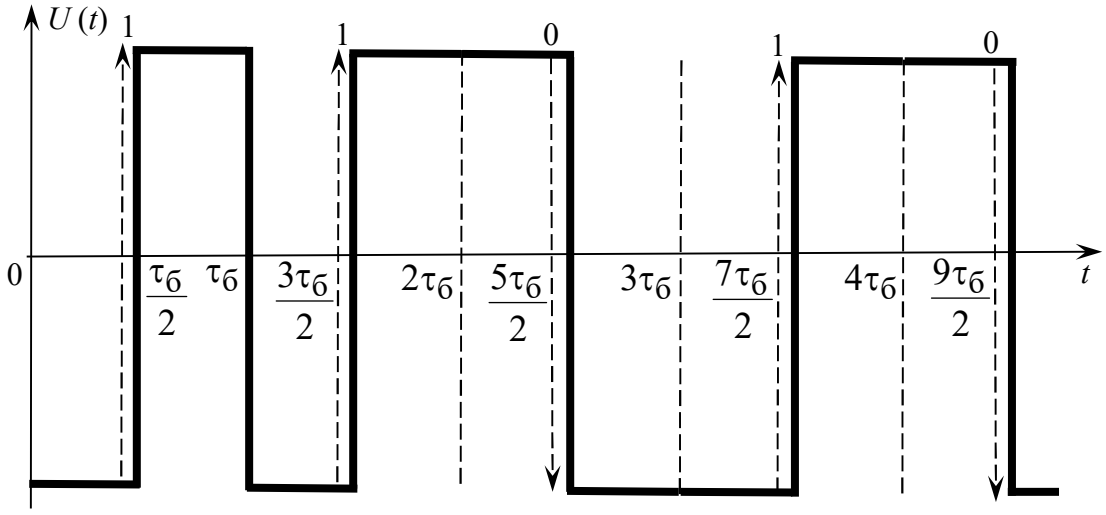


Рис. 1.21 Принцип формування двійкового сигналу із манчестерським кодом

Як було відмічено, манчестерський код дозволяє знаходити помилки передавання сигналу, які пов’язані із відсутністю зміни його полярності у середині такту. Це є важливим у разі передавання сигналу в умовах короткочасної потужної завади. Не можуть бути виявленими такі помилки, коли дія завади є довгочасною і відбувається зміна полярності сигналу у середині такту. У теорії кодування сигналів такі помилки називаються дзеркальними [2 – 6, 8].

Тобто, якщо розглядати манчестерський код як електричний сигнал, він є натуральним, а з точки зору способу кодування та принципу роботи цей код можна вважати завадостійким. Це зайвий раз підтверджує умовність класифікації кодів, яка була розглянута у підрозділі 1.2 цієї частини посібника.

Крім розглянутої у цьому підрозділі класичного манчестерського коду, існують інші способи формування кодів із внутрішньою синхронізацією, які розглядатимуться у наступних підрозділах.

1.2.4.2 Різницеві манчестерські коди

Іншим способом формування манчестерських кодів є різницеве манчестерське кодування, яке було використано в вісімдесятих роках XX століття в комп'ютерних мережах Token Ring [17 – 23]. На відміну від описаних вище стандартних манчестерських кодів, тут зміна полярності сигналу в середині такту не несе ніякої інформації і використовується лише як засіб його самосинхронізації, а значення інформаційного біту, який передається, визначається через зміну сигналу на початку такту. Якщо на початку такту рівень сигналу змінюється, значення інформаційного біту дорівнює нулю, а якщо не змінюється – одиниці. Послідовність імпульсів, яка відповідає різницевому манчестерському коду числа 01001, наведена на рис. 1.22. На цьому рисунку перепади сигналу, які відповідають значенню одиниці, показані тонкими стрілками. Для різницевого манчестерського коду символу 1 відповідає послідовність біт 11, а символу 0 – послідовність 01.

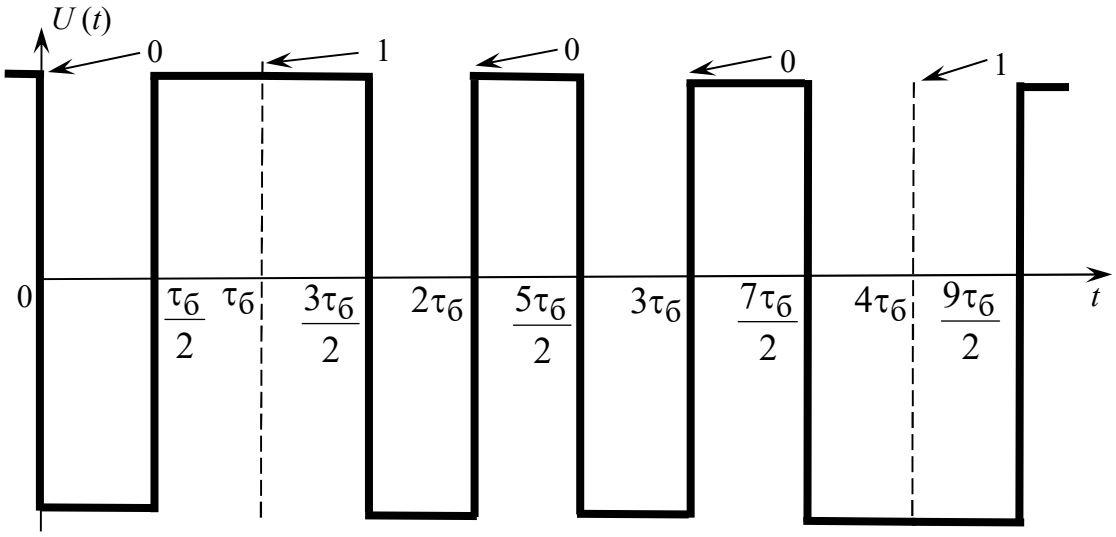


Рис. 1.22 Принцип формування двійкового сигналу за різницевим манчестерським кодом

1.2.4.3 Коди Міллара

Іншими імпульсними кодами із самосинхронізацією є коди Міллара та коди із інверсією кодових посилань [30, 33]. Код Міллара, або біполярний код з модуляцією із затримками, формується наступним чином. Одиницям у кодовій послідовності, як і в манчестерському коді, відповідає зміна полярності сигналу в середині відповідного часового інтервалу, а сигнали нулів формуються без зміни полярності в середині інтервалу, проте, для забезпечення самосинхронізації сигналу, послідовності нулів відповідає зміна його полярності на початку часового інтервалу (рис. 1.23).

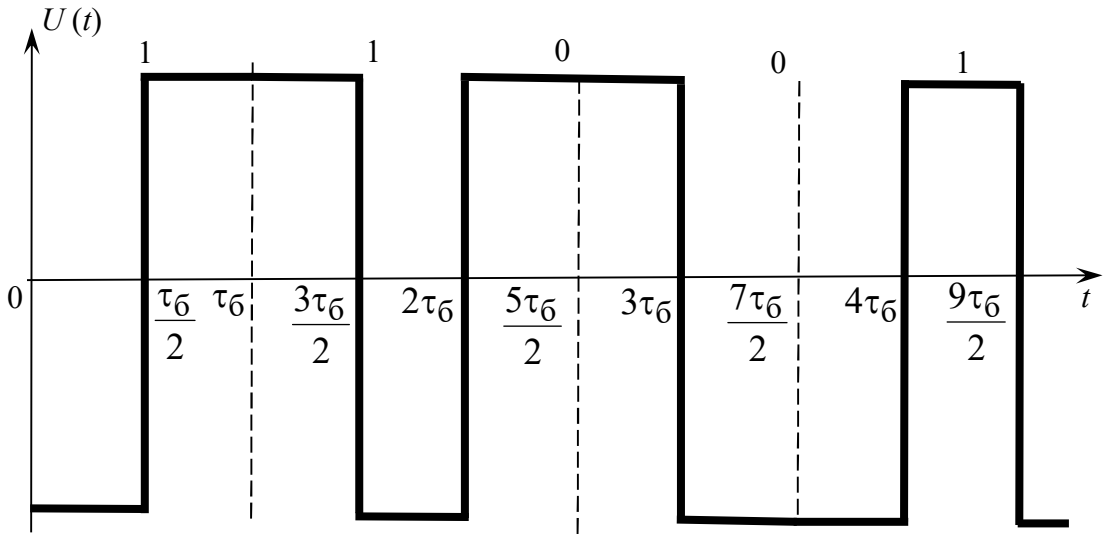


Рис. 1.23 Принцип формування двійкового сигналу за кодом Міллара

Для кодів Міллара, на відміну від манчестерських кодів, напрямок зміни полярності, з верхнього на нижній або з нижнього на верхній рівень сигналу, не має значення з точки зору інформації, яка передається. Часова діаграма для двійкового сигналу, сформованого з використанням коду Міллара, наведена на рис. 1.23. Для коду Міллара символу 1 відповідає послідовність біт 01 а символу 0 – послідовність 10.

1.2.4.4 Коди із інверсією кодових посилань

Формування імпульсного коду із інверсією кодових посилань (англійський термін Coded Mark Inversion, CMI) багато в чому схоже на принцип формування коду Міллара, проте у цьому разі кодування здійснюється зворотним чином. А саме, на відміну від коду Міллара, у цьому коді нулі кодуються через зміну полярності сигналу в середині часового інтервалу, а одиниці – через її зміну на початку часового інтервалу [30, 33]. Часова діаграма для двійкового сигналу, сформованого з використанням коду із інверсією кодових посилань, наведена на рис. 1.24. У коді із інверсією кодових посилань символу 1 відповідає послідовність біт 00, а символу 0 – послідовність 11.

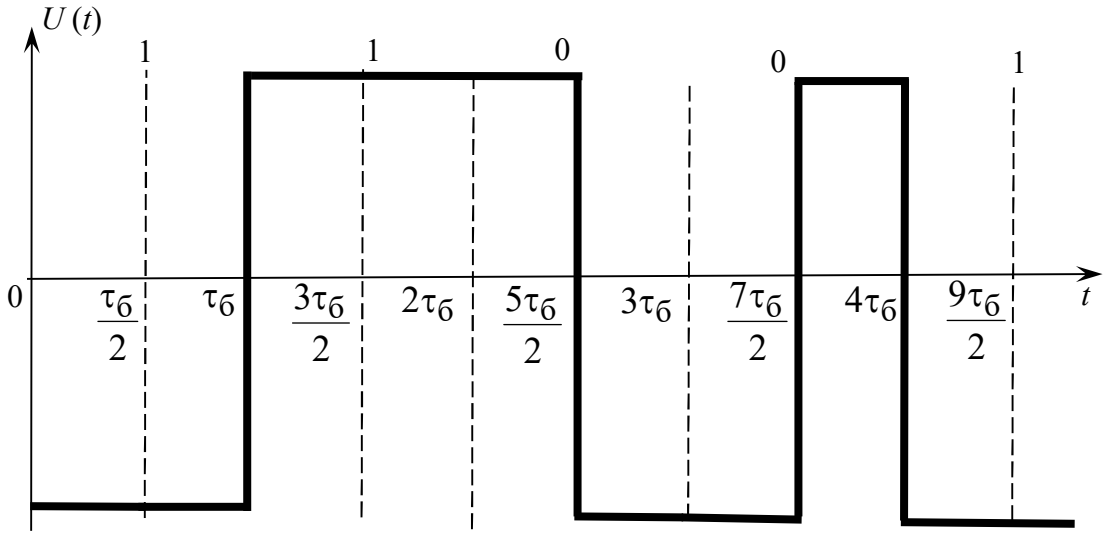


Рис. 1.24 Принцип формування двійкового сигналу за кодом з інверсією кодових посилань

Тобто, головним принципом формування імпульсних кодів є те, що кодування нулів та одиниць здійснюється через зміну полярності сигналу у відповідний час. Для імпульсних кодів рівень сигналу не має суттєвого значення, проте важливими є час та напрямок зміни полярності сигналу. Перевагою імпульсних кодів є наявність внутрішньої синхронізації та відсутність постійної складової у спектрі сигналу. Головним недоліком

сигналів із імпульсним кодуванням є розширення їх частотного спектру, зазвичай він у два рази перевищує бітову швидкість передавання інформації.

Порівняльний аналіз спектрів сигналів з потенціальним та імпульсним способом кодування буде наведений у підрозділі 1.2.9 цієї частини посібника.

1.2.5 Потенціальні коди

1.2.5.1 Уніполярні коди

Всі потенціальні коди не містять внутрішніх засобів синхронізації сигналу. Найпростішим таким кодом є уніполярний код (англійський термін unipolar code), який цілком відповідає потенціальному коду, часова діаграма якого показана на рис. 1.4, а. Тобто, одиниці відповідає високий рівень, а нулю – низький рівень сигналу. Для більш довгої послідовності символів сигнал з уніполярним кодом показаний на рис. 1.25. Як було відмічено у підрозділі 1.2.1, такий спосіб кодування має низку недоліків, серед яких слід відмітити наявність постійної складової у спектрі, відсутність внутрішньої синхронізації сигналу та складність розпізнавання низькорівневого сигналу нуля за умови наявності завад. Проте, частотний діапазон уніполярного сигналу є досить вузьким і, відповідно до наведеної у підрозділі 4.3 першої частини посібника теореми Найквіста, частотна смуга цього коду співпадає із бітовою швидкістю передавання інформації. Такий сигнал називається уніполярним кодом без повернення до нуля (англійський термін Non Return to Zero, NRZ) [30, 33].

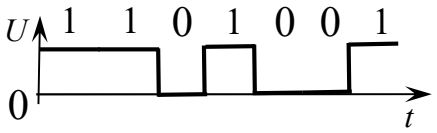


Рис. 1.25 Сигнал з уніполярним кодом без повернення до нуля

Для внесення синхронізації в уніполярний код під час передавання

одиниць використовують імпульсний сигнал, а не постійне значення. Зрозуміло, що такий двійковий сигнал можна розглядати як комбінацію потенціального та імпульсного кодування. Проте проблема синхронізації сигналу у разі передавання послідовності нулів у цьому випадку залишається. Такий сигнал називається уніполярним кодом із поверненням до нуля. Часова діаграма такого сигналу наведена на рис. 1.26.

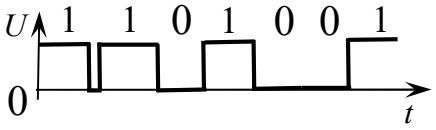


Рис. 1.26 Сигнал з уніполярним кодом із поверненням до нуля

Іншим уніполярним кодом, який широко використовується у системах оптичного зв'язку, є потенціальний код із інверсією за умови одиниці. Принцип роботи цього коду досить простий. У разі передавання нуля рівень сигналу не змінюється, а у разі передавання одиниці – змінюється. Часова діаграма сигналу із таким способом кодування наведена на рис. 1.27.

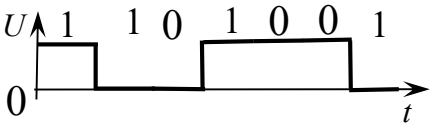


Рис. 1.27 Сигнал з кодом із інверсією за умови одиниці

Перевагою коду із інверсією за умови одиниці над біполярними кодами в оптичних системах зв'язку є те, що тут бажано використовувати не 3, а 2 рівні кодування сигналу, яким відповідає наявність світла та його відсутність.

1.2.5.2 Біполярні коди

Біполярні потенціальні коди відрізняються від уніполярних тим, що в них значення нуля кодується не низьким рівнем напруги, а негативним потенціалом. Відповідна форма біполярного сигналу була показана на рис. 1.4, а. Серед біполярних кодів найбільш простим також є код без

повернення до нуля [30, 33]. Єдиною відмінністю цього коду від уніполярного є те, що в ньому рівню нуля відповідає не нульове, а від'ємне значення напруги. Відповідна форма сигналу показана на рис. 1.28.

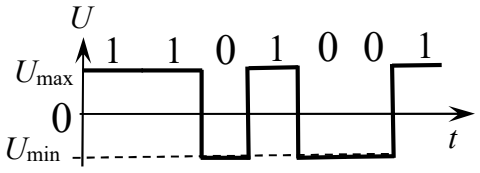


Рис. 1.28 Сигнал з біполярним кодом без повернення до нуля

Як і для уніполярного сигналу, можливим є використання у біполярному сигналі без повернення до нуля не постійних значень, а імпульсів. У цьому разі, на відміну від уніполярного сигналу, імпульсами передаються як значення 1, так і значення 0, що дозволяє забезпечити повну синхронізацію. Такі сигнали називаються біполярними кодами із поверненням до нуля. Відповідна форма сигналу наведена на рис. 1.29.

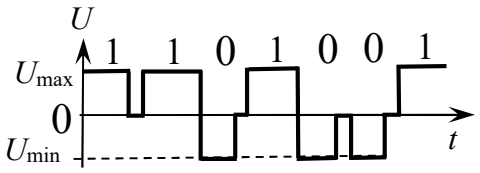


Рис. 1.29 Сигнал з біполярним кодом із поверненням до нуля

Тобто, сигнал із біполярним кодом із поверненням до нуля, об'єднує в собі переваги як потенціальних, так й імпульсних кодів. Він має внутрішню синхронізацію та достатньо вузький частотний спектр, проте уникнути наявності постійної складової в такому сигналі не вдається. Постійна складова для цього сигналу визначається довжиною послідовностей нулів та одиниць. Якщо інформація, яка передається, має велику кількість одиниць, такий сигнал буде мати позитивну постійну складову, а якщо переважає

кількість нулів – відповідно негативну.

Цього недоліку певною мірою вдається уникнути у коді без повернення до нуля із інверсією за умови одиниці (англійський термін Non Return to Zero Inversion, NRZI) [15 – 23, 30, 33]. Його також називають кодом із інверсією маркерів із чергуванням (англійський термін Alternate Mark Inversion, AMI). Цей код також класифікується як біполярний, проте, на відміну від біполярного коду без повернення до нуля, у ньому нулі кодуються нульовим рівнем сигналу, а максимальний та мінімальний рівень сигналу використовуються для кодування послідовностей одиниць, які ідуть підряд. Полярність напруги, якою кодується одиниця, постійно змінюється. Тобто, якщо для кодування поточної одиниці було використане позитивне значення потенціалу, для кодування наступного символу буде використане негативне значення. Після появи нулів значення сигналу одиниці також змінюється. Відповідна часова залежність для коду без повернення до нуля з інверсією за умови одиниці наведена на рис. 1.30.

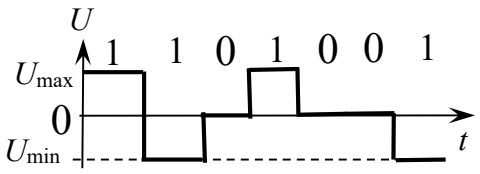


Рис. 1.30 Сигнал з кодом без повернення до нуля з інверсією за умови одиниці

Біполярний код без повернення до нуля із інверсією за умови одиниці, який також називають кодом AMI, дозволяє уникнути постійної складової сигналу і має внутрішню синхронізацію за умови одиниць, які йдуть підряд, проте за умови нулів, які йдуть підряд, ці властивості коду втрачаються. Тому були розроблені більш досконалі способи кодування сигналів, які дозволяють покращити характеристики коду AMI. Насамперед, це способи кодування, у яких довгі послідовності нулів замінюються на інші електричні сигнали, зазвичай з використанням заборонених символів, щоб їх не можна

було хибно прийняти за іншу, неспотворену кодову комбінацію. До таких способів електричного кодування сигналів належать коди B8ZS та HDB3, які розглядатимуться у наступному підрозділі. Іншим способом уникнення довгих комбінацій нулів, що йдуть підряд, є математична обробка інформації. Алгоритми обробки двійкових числових послідовностей будуються так, що у вихідному коді кількість нулів, незалежно від їх кількості у початковій комбінації, стає обмеженою. Такі алгоритми у комп'ютерній алгебрі називаються алгоритмами скрамблювання та будуть розглянуті у підрозділі 1.3.6 цієї частини посібника.

1.2.6 Коди B8ZS та HDB3

Коди B8ZS та HDB3 були розроблені як модифікація біполярного коду без повернення до нуля із інверсією за умови одиниці для покращення його характеристик з точки зору необхідності придушення постійної складової сигналу [15 – 23, 30, 33].

Код B8ZS (абревіатура англійського словосполучення Bipolar with 8-Zeros Substitution, біполярний код із заміною восьми нулів) будується наступним чином. Припустимо, що вхідна числова комбінація містить довгу послідовність із восьми нулів, які йдуть підряд. У цьому разі у вихідному сигналі після трьох нулів замість решти п'яти формується така комбінація із п'яти сигналів: V – 1^* – 0 – V – 1^*, де V (абревіатура англійського слова Violation – заборонено) позначає сигнал одиниці, заборонений для даного такту, 1^* – сигнал одиниці, дозволений для даного такту. Якщо на приймальній стороні реєструється така кодова комбінація, приймальний пристрій після трьох нулів спостерігає дві помилки у п'яти переданих символах. Зрозуміло, що наявність такої великої кількості спотворень сигналу у каналі зв'язку є дуже малоймовірною. Тому, якщо приймальний пристрій фіксує відмічену вище послідовність кодових символів після трьох нулів, вона автоматично заміняється на п'ять нулів. Часова діаграма біполярного коду без повернення до нуля із інверсією за умови одиниці для

кодової послідовності 11000000001100000 наведена на рис. 1.31, а, а часова діаграма для коду B8ZS для цієї ж послідовності нулів та одиниць – на рис. 1.31, б.

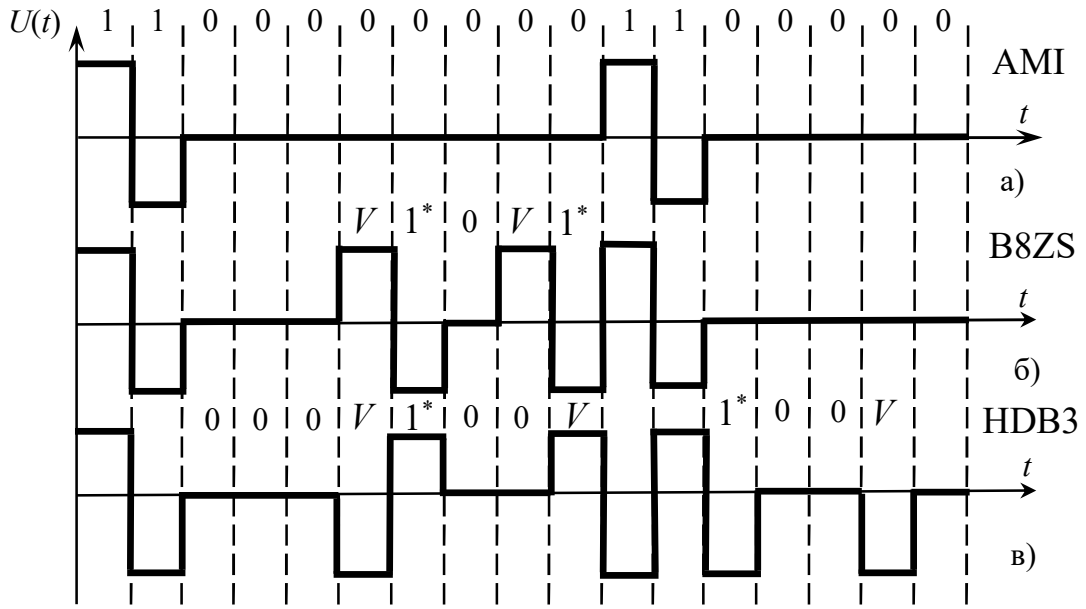


Рис. 1.31 Порівняння часових діаграм сигналів із кодами AMI (а), B8ZS (б) та HDB3 (в) для вхідної числової кодової послідовності 11000000001100000

Код HDB3 (абревіатура англійського словосполучення High Density Bipolar 3-Zeros, біполярний код високої щільності із трьома нулями), на відміну від коду B8ZS, виправляє будь-яки чотири нулі, які йдуть підряд у вхідній послідовності, проте алгоритм формування коду HDB3 є більш складним. У коді HDB3 кожні чотири нулі замінюються послідовностями символів, які містять символ *V*, проте для більшої ефективності придушення постійної складової сигналу полярність сигналу *V* чергується під час послідовних змін. Крім того, для заміни чотирьох нулів, які йдуть підряд, використовуються різні кодові комбінації, залежно від попередніх символів вхідної послідовності. Якщо перед заміною послідовності нулів вхідний код містив непарну кількість одиниць, для заміни використовується послідовність символів 000*V*, а якщо кількість одиниць у коді була парною – послідовність символів 1^*00V. Часова діаграма для коду HDB3 для кодової послідовності 11000000001100000 наведена на рис. 1.31, в.

1.2.7 Потенціальний багаторівневий код 2B1Q

Серед багаторівневих систем кодування із потенціальним кодом найбільше застосування знайшов потенціальний код 2B1Q, який має чотири рівні кодування сигналу [15 – 23, 30, 33]. За рахунок цього за один такт кодується не один біт, як в двійкових кодах, а два біти, що дозволяє, згідно із формулою Хартлі (4.31), наведеній у першій частині посібника, вдвічі підвисити швидкість передавання інформації. Проте, згідно із оцінками, наведеними у підрозділі 4.6.3 першої частини посібника, за умови наявності шумів підвищення швидкості передавання інформації буде меншим.

У коді 2B1Q парі бітів 00 відповідає негативне значення потенціалу –2,5 В, парі бітів 01 – негативне значення потенціалу –0,833 В, парі бітів 11 – позитивне значення потенціалу +0,833 В, а парі бітів 10 – позитивне значення потенціалу +2,5 В. У разі використання такого способу кодування необхідно уникати довгих послідовностей однакових пар бітів, оскільки інакше сигнал перетворюється на постійну складову. Часова діаграма сигналу, закодованого потенціальним кодом 2B1Q, наведена на рис. 1.32.

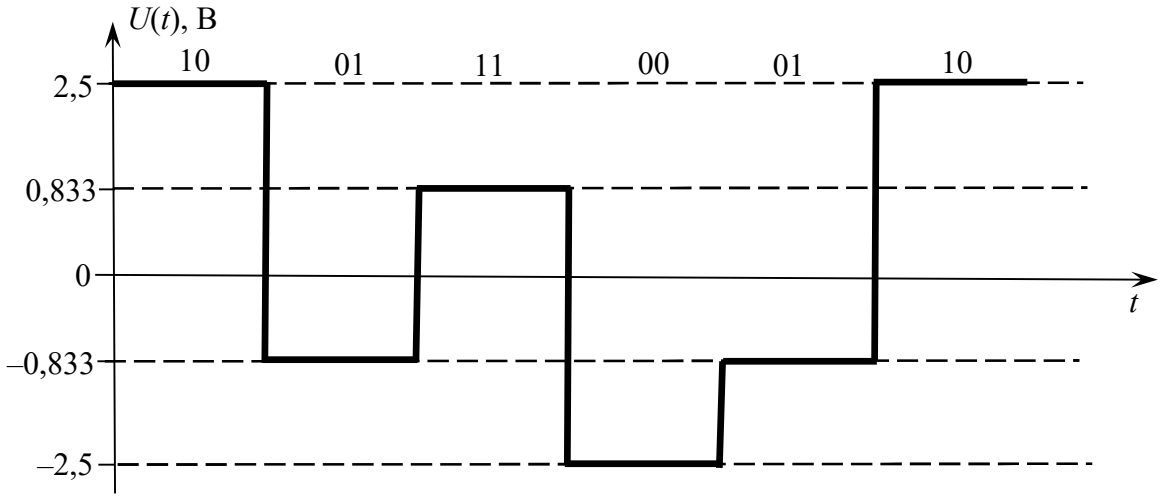


Рис. 1.32 Спосіб формування сигналу для чотирирівневого потенціального коду 2B1Q

1.2.8 Код MLT-3

Іншим стандартом кодування, в якому використовується три рівні сигналу, а саме: позитивний потенціал (+*U*), негативний потенціал (–*U*) та

нульовий потенціал (0), є стандарт багаторівневої передачі MLT-3 (абревіатура англійського словосполучення **M**ulti **L**evel **T**ransmission – 3). Цей метод кодування, який запропонований фірмою Cisco Systems [41, 42], оснований на тому, що за умови передавання нуля рівень сигналу зберігається, а у разі передавання одиниці здійснюється перехід на сусідній рівень, наприклад, з рівня негативного потенціалу на рівень нуля, або з рівня нуля на рівень позитивного потенціалу. Тобто, як в сигналі з інверсією за умови одиниці (NRZI), у сигналі MLT-3 рівень сигналу не змінюється за умови передавання нуля. Крім цього, у разі передавання довгих послідовностей одиниць, для завершення циклу передавання використовується чотири переходи [41, 42]. Використання стандарту MLT-3 дозволяє вчетверо зменшити частотний діапазон сигналу відносно тактової частоти. Тобто, сигнал MLT-3 має найменший частотний діапазон із всіх можливих способів кодування. Це робить стандарт MLT-3 найбільш зручним для передавання інформації у разі використання мідних проводів як середовища передачі. У зв'язку з цим цей метод кодування використовується у мережах FDDI, основаних на мідних проводах, та у мережах стандарту Fast Ethernet 100 BASE-TX [15 – 23]. Приклад формування сигналу стандарту MLT-3 наведений на рис. 1.33.

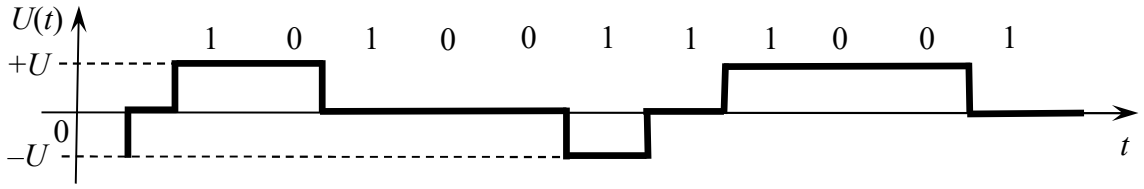


Рис. 1.33 Приклад формування коду MLT-3

1.2.9 Спектри сигналів із різним способом кодування та їх порівняльний аналіз

Слід відзначити, що розглянуті потенціальні та імпульсні коди мають досить широкий частотний спектр, що, відповідно, потребує використання каналів зв'язку із широкою смугою пропускання. Частотні спектри сигналів,

які відповідають розглянутим вище способам їх кодування у цифрових системах зв'язку, наведені на рис. 1.34. Вимірювання спектрів сигналів проводились для довільних двійкових послідовностей з рівними ймовірностями комбінацій одиниць і нулів у вхідних даних. Під час побудови графічних залежностей, наведених на рис. 1.34, проводились усереднення за усіма можливими вхідними комбінаціями одиниць та нулів [17 – 23]. Для всіх наведених графічних залежностей частота сигналу $F_{\text{с}}$ відповідає бітовій швидкості передавання сигналу, відповідно до теореми Найквіста, наведеній у підрозділі 4.3 першої частини посібника.

Як видно із рис. 1.34, потенціальний код NRZ має досить вузький частотний спектр, проте, як відмічалось у попередньому підрозділі, суттєвий недолік цього способу кодування полягає в тому, що він має постійну складову. Найбільш широкий спектр мають манчестерські та біполярні імпульсні коди, оскільки їх використання потребує підвищення тактової частоти сигналу в 3 рази. Найбільш вузький частотний спектр має сигнал із кодом B8ZS, максимум спектру цього сигналу відповідає частоті $\frac{F_{\text{с}}}{2}$. Також ефективним для використання у проводових системах зв'язку є код MLT-3, частотний діапазон якого становить $\frac{F_{\text{с}}}{4}$.

У зв'язку з цим у сучасних стандартах зв'язку використовуються не біполярні та імпульсні коди, а коди із скрамблюванням та надлишкові коди. Способи формування таких кодів будуть розглянуті у підрозділі 1.3. Наприклад, якщо у стандартах комп'ютерних мереж Ethernet та Token Ring, які були розроблені та використовувались у вісімдесятих роках минулого століття, для формування сигналів використовувались манчестерські коди, у сучасних стандартах FDDI, Fast Ethernet, Gigabit Ethernet та інших використовують натуральні коди, способи формування яких будуть описані у підрозділі 1.3, а також коди B8ZS та HDB3. Проте, як видно з рис. 1.34, спектр сигналу для коду B8ZS зсунутий в область високих частот на

величину $\frac{F_c}{4}$. Спектри натуральних кодів будуть наведені у підрозділі 1.3.

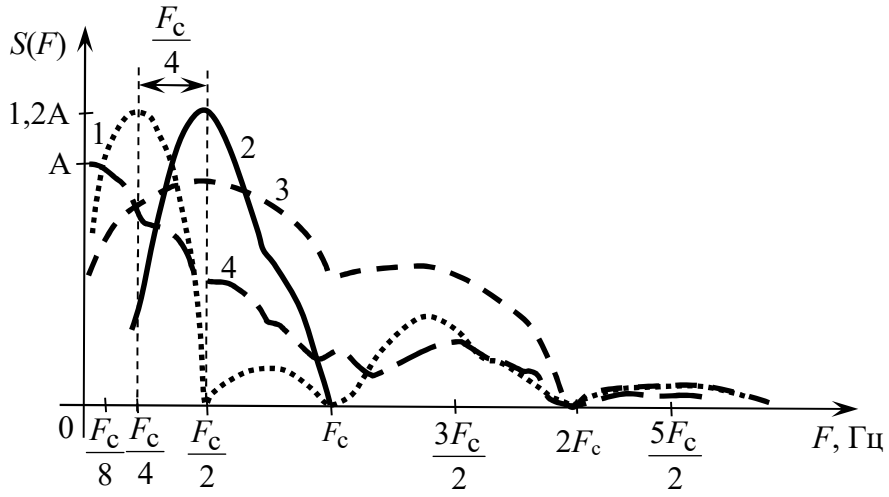


Рис. 1.34 Спектри сигналів для різних способів кодування. 1 – код 2B1Q, 2 – код B8ZS, 3 – манчестерські та біполярні імпульсні коди, 4 – код NRZ

Слід відзначити, що спектри сигналів, наведені на рис. 1.34, отримані експериментально для реальних систем зв'язку у разі передавання псевдовипадкових послідовностей нулів та одиниць. Для конкретних кодових послідовностей із відомим значенням кожного біта спектр сигналу можна оцінити теоретично з використанням комп'ютерних засобів моделювання. Відповідні програми, написані мовою програмування системи MatLab, наведена у додатку В. Оцінки спектрів у цих програмах основані на логічному описанні розглянутих у цьому підрозділі способів формування сигналів та на визначенні за часовим описанням сигналу його частотного діапазону. Для розрахунку спектрів сигналів використовувалось співвідношення для інтеграла Фур'є у дійсній формі, тобто, співвідношення (3.29), наведені у підрозділі 3.2.2 першої частини посібника:

$$A(\omega)=\frac{1}{\pi}\int_0^{\infty} f(t)\cos(\omega t)dt,\ B(\omega)=\frac{1}{\pi}\int_0^{\infty} f(t)\sin(\omega t)dt,$$

$$A_{\Sigma}(\omega)=\sqrt{A^2(\omega)+B^2(\omega)} \qquad (1.15)$$

Слід зазначити, що для розглянутих потенціальних та імпульсних кодів двійкових сигналів вирази (1.15) у значній мірі спрощується, оскільки інтеграли $A(\omega)$ та $B(\omega)$, які описують синусну та косинусну складові спектру цифрового сигналу, для постійного значення амплітуди на окремих ділянках часу можна обчислити аналітично. Наприклад, якщо значення амплітуди сигналу на інтервалі часу τ_i, від моменту часу t_i до t_{i+1}, складає U_i, у загальному випадку для всього часу дії цифрового сигналу t_n, вважаючи цей час скінченним, можна записати наступні аналітичні вирази:

$$t_0 = 0;\ A(\omega) = \frac{\sum_{i=1}^{n} U_i \cdot (\sin(\omega t_i) - \sin(\omega t_{i-1}))}{\omega}; \qquad (1.16)$$

$$B(\omega) = \frac{\sum_{i=1}^{n} U_i \cdot (\cos(\omega t_{i-1}) - \cos(\omega t_i))}{\omega}.$$

У програмах, наведених у додатку В, аналіз часових залежностей сигналів для заданої двійкової кодової послідовності та форми подання проводиться з використанням функції **`digspectrum`**. Параметрами виклику цієї функції є послідовність вхідних бітів для формування часової залежності сигналу та параметр форми подання сигналу n_s. Можливими значеннями параметра n_s є наступні.

1. n_s = 1: уніполярний код із імпульсами синхронізації від'ємної полярності. Значенню 0 відповідає нульова амплітуда сигналу, значенню 1 – одинична, а амплітуда синхронізуючих імпульсів складає –1. У наведених нижче тестових задачах всі розрахунки були зроблені для шпаруватості сигналу 20% від часу передавання одного біту $t_б$ та тривалості синхронізуючих імпульсів $\tau_{імп} = 0{,}1t_б$.

2. n_s = 2: уніполярний код без повернення до нуля.

3. n_s = 3: уніполярний код без повернення до нуля з інверсією за умови одиниці, або код AMI.

4. n_s = 4: манчестерський код.

5. n_s = 4: код 2B/1Q.

Всі наведені далі результати розрахунків отримані для бітової швидкості передавання даних $t_б = 10^{-6}$ сек. Такий обмежений діапазон розглянутих часових параметрів сигналу був обумовлений тим, що головним завданням був аналіз узагальнених спектрів сигналів та умовного частотного діапазону відносно заданої бітової швидкості передавання даних джерелом

повідомлення, а ці параметри сигналів безпосередньо не залежать від швидкості передавання бітів $v_б = \frac{1}{t_б}$. У разі зміни цього параметру частотний діапазон сигналу, безумовно, змінюється, але хід залежностей амплітуди сигналу від частоти цілком зберігається. Слід відзначити, що експериментально отримані графічні залежності для спектрів сигналів різної форми подання, наведені на рис. 1.34, також побудовані для відносної граничної частоти F_c. Результатом роботи цієї функції є видання на екран графічної залежності $A(t)$ для заданої кодової послідовності та визначеного способу кодування. Для побудови графічних залежностей амплітуди сигналу від часу $U(t)$ брався відносно малий час квантування сигналу $\Delta t = \frac{t_б}{50}$.

Функціональне призначення іншої програми **SpectrFourier**, наведеної у додатку В, полягає в обчисленні спектральної характеристики заданого сигналу з використанням співвідношень (1.16). Параметрами виклику цієї функції є упорядковані масиви значень відліків часу та значень амплітуди сигналу, які цим відлікам відповідають. Важливою умовою коректності роботи програми є співпадання розмірності цих масивів. Для описання залежності амплітуди сигналу від часу відповідно до співвідношень (1.16) проводився аналіз цих вхідних послідовностей та бралися лише ті значення відліків часу, яким відповідала зміна напруги, а проміжні відліки відкидалися. У цьому разі залежність амплітуди цифрового сигналу від часу описується з використанням спрощеної матриці із двома рядками, в першому з яких стоять табульовані значення часу, а в другому – значення амплітуди сигналу, які відповідають цим часовим інтервалам. Такі залежності можна також записати у вигляді арифметико-логічного виразу (1.9) наступним чином:

$$U(t) = \sum_{i=1}^{n} (t_{i-1} < t \le t_i) \cdot U_i \,, \qquad (1.17)$$

де t_i – відлікові значення інтервалів часу, які відповідають зміні напруги цифрового сигналу, U_i – значення напруги на цих інтервалах.

На рис. 1.35 наведені отримані з використанням програми **digspectrum** для заданої кодової послідовності 110010101 часові залежності сигналів за умови застосування різних способів кодування, а також частотні спектри, які відповідають цим сигналам.

Отримані розрахункові результати у знаній мірі підтверджують розглянуті вище теоретичні міркування. Дійсно, найширший спектр мають сигнали манчестерського коду та коду з імпульсною синхронізацією, а найбільш вузький спектр відповідає багаторівневому коду 2B1Q. Аналогічний результат дає порівняння спектрів сигналів кодової послідовності 100000001 для коду із синхронізацією, коду AMI, манчестерського коду та коду 2B1Q, суміщених на одній графічній площині. Відповідні графічні залежності наведені на рис. 1.36. Більші максимальні значення амплітуди сигналу для манчестерського коду та для коду 2B1Q пов’язані з тим, що для цих видів кодів значенню 0 не відповідає нульова амплітуда, в обох випадках біт нуля кодується іншим способом. Видно також, що максимальне значення амплітуди для манчестерського коду відповідає циклічній частоті 3 F_c, а для коду 2B1Q – частоті F_c.

Цікавим є також аналіз спектрів сигналів коду AMI за умови появи великої кількості одиниць, які йдуть підряд. На рис. 1.37 наведені такі спектри для двох, трьох, п’яти, шести та десяти одиниць, які йдуть підряд. Насамперед зрозуміло, що спектр сигналу коду AMI не має постійної складової у разі наявності у коді парної кількості одиниць. Крім цього видно, що у разі збільшення кількості одиниць спектр широкосмужного сигналу стискується до гребінчастого, а амплітуда сигналу на дискретних відліках $3\cdot\omega_c$, $9\cdot\omega_c$ та $15\cdot\omega_c$ суттєво збільшується. Тобто, у граничному випадку, за умови $n \to \infty$, неперервний спектр широкосмужного сигналу перетворюється до дискретного.

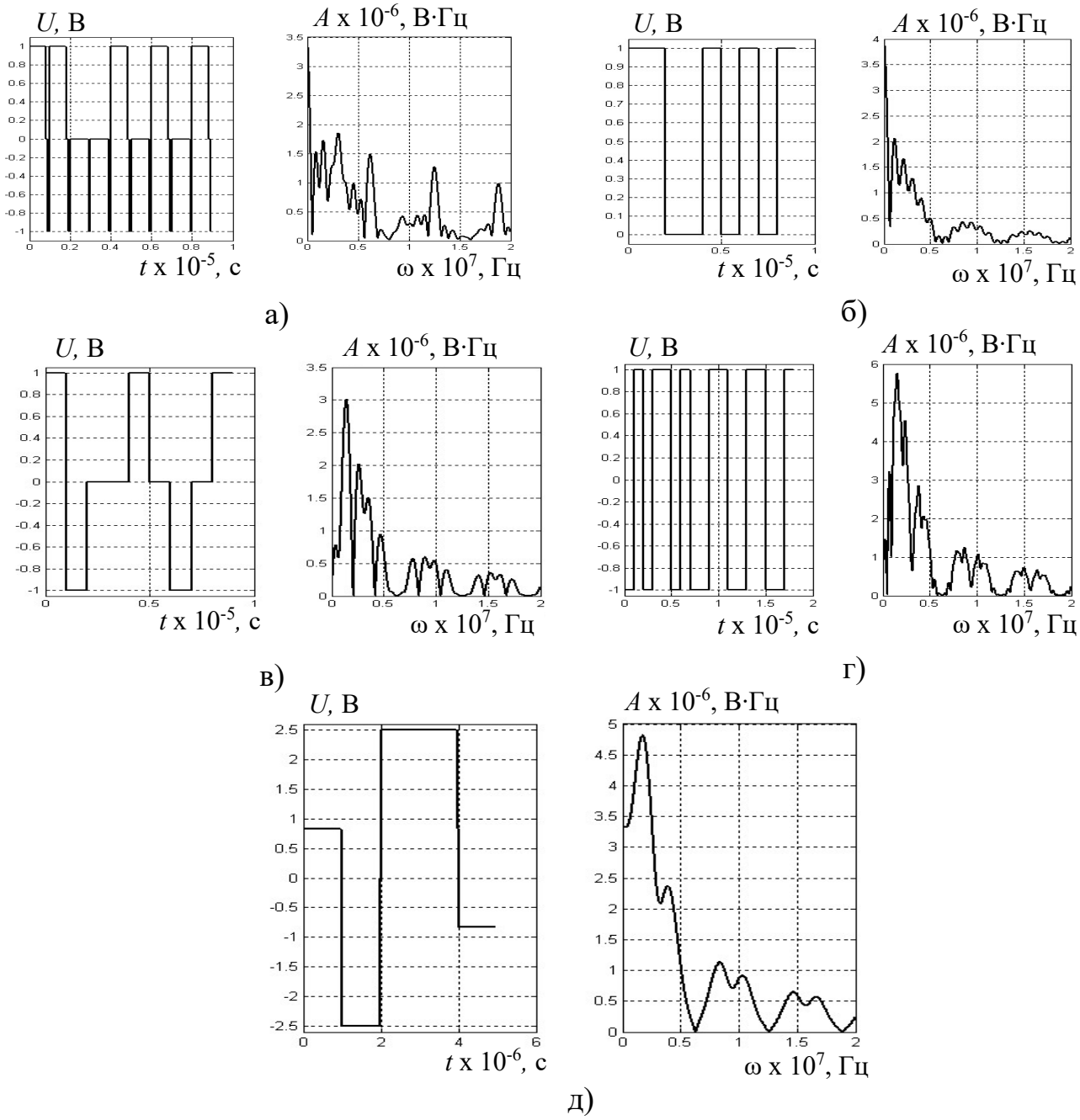


Рис. 1.35 Часові залежності та спектри сигналів для кодової послідовності 110010101 за умови різних способів кодування: а – уніполярний код із імпульсами синхронізації від'ємної полярності, б – уніполярний код без повернення до нуля, в – уніполярний код без повернення до нуля з інверсією за умови одиниці (код AMI), г – манчестерський код, д – код 2B1Q

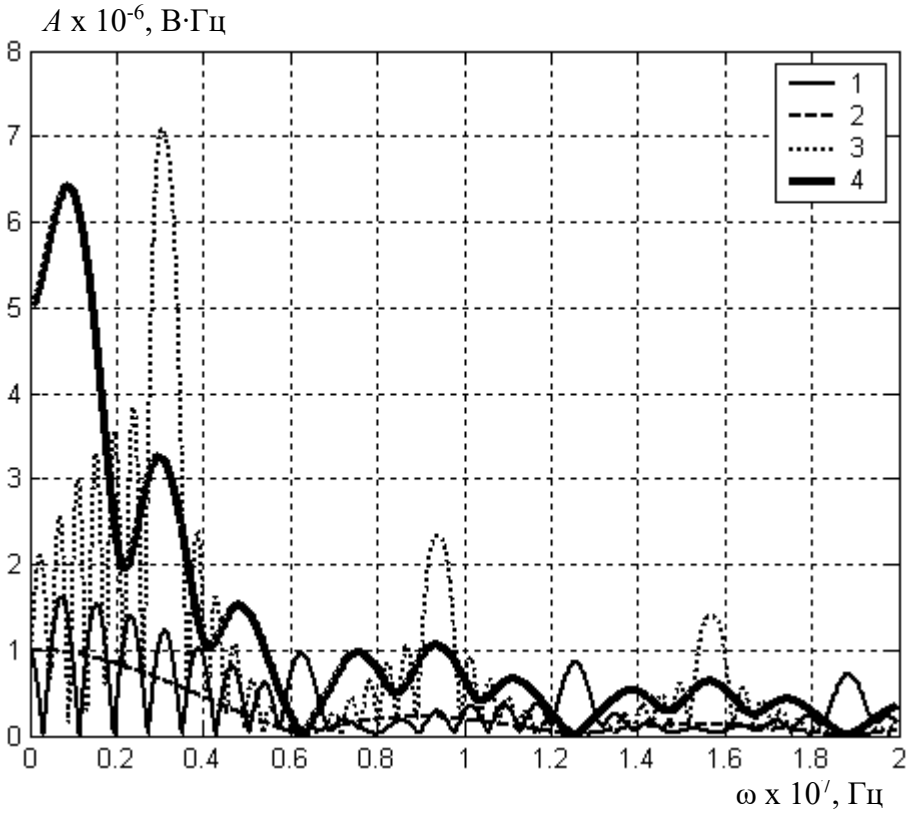


Рис. 1.36 Спектри сигналів для кодової послідовності100000001 за умови використання різних способів кодування: 1 – уніполярний код із імпульсами синхронізації від'ємної полярності, 2 – уніполярний код без повернення до нуля з інверсією за умови одиниці (код AMI), 3 – манчестерський код, 4 – код 2B/1Q

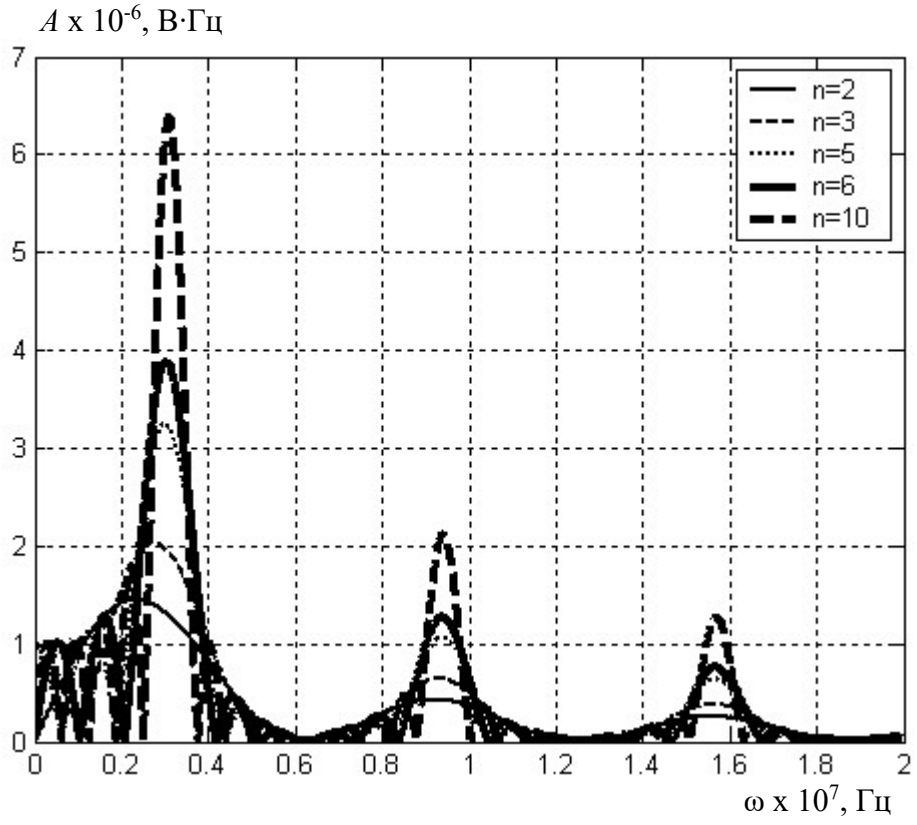


Рис. 1.37 Аналіз спектрів сигналу коду AMI за умови наявності різної кількості одиниць, які йдуть підряд: *n* – кількість одиниць

Надалі у цьому розділі посібника програма **digspectrum**, наведена у додатку В, буде використана для аналізу спектрів сигналів коду AMI за умови застосування різних способів логічного кодування двійкових послідовностей, які розглядатимуться.

1.3 Натуральні двійкові коди

1.3.1 Поняття про натуральні двійкові коди та їхні параметри

Натуральні двійкові коди широко використовуються для подання інформації у цифрових електронних системах, переваги яких розглядалися у підрозділі 6.1 першої частини посібника. Як було відмічено у підрозділі 1.1.2 цієї частини посібника, серед натуральних цифрових кодів розрізняють двійкові та багатопозиційні. Головним параметром натуральних кодів є розрядність коду, а кількість можливих кодових комбінацій розраховується за формулою (1.3).

Серед натуральних кодів розрізняють позиційні та непозиційні [44 – 46]. Відмінною рисою позиційних кодів є те, що кожний розряд має свій ваговий коефіцієнт, пропорціональний степені системи числення, у який записане відповідне число. Наприклад, у десятковій системі числення вагові коефіцієнти розрядів числа складають $10^0 = 1$, $10^1 = 10$, $10^2 = 100$, $10^3 = 1000$... У двійковій системі числення відповідні коефіцієнти розрядів числа становлять: $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$... У двійковій арифметиці та у комп’ютерній техніці зазвичай використовують двійкову, вісімкову та шістнадцяткову системи числення [25, 44 – 46, 51]. Значення числа, яке записано у позиційному коді, можна розрахувати із наступного співвідношення [39, 40, 44 – 46]:

$$A = k_1 + k_2 \cdot s + k_3 \cdot s^2 + k_4 \cdot s^3 + \cdots + k_n \cdot s^{n-1}, \qquad (1.18)$$

де n – кількість розрядів числа, s – основа системи числення, k – значення розряду позиційного коду, а індекси визначають номер відповідного розряду.

Загалом теорія натуральних двійкових кодів тісно пов’язана із теорією чисел, яка розглядалася у розділі 1 другої частини посібника [48].

Серед непозиційних кодів найбільш поширеними у комп'ютерній техніці є рефлексні коди, зокрема код Грея, який розглядатиметься у підрозділі 1.3.4. До непозиційних натуральних кодів можна також віднести коди 4В/5В, 8В/6Т та коди двійкових чисел, отримані з використанням алгоритмів скрамблювання. Всі ці типи кодів розглядатимуться далі у цьому підрозділі.

Згідно із структурною схемою, наведеною у першій частині посібника на рис. 2.41 [1], у обчислювальних електронних системах та у системах зв'язку кодування сигналу здійснюється до його модуляції, для цього використовуються способи подання двійкових сигналів у каналах зв'язку, які були розглянуті у підрозділі 1.2 цієї частини посібника. Проте слід відзначити, що класифікація кодів, наведена на рис. 1.1, є досить умовною, і зазвичай чіткої границі між натуральними і завадостійкими кодами не існує. Наприклад, у системах зв'язку часто використовують коди 4В/5В та 8В/6Т, які, з одного боку, формуються за звичайними засобами кодування без розрахунку контрольних сум, а з іншого боку, є надлишковими. Способи формування цих кодів будуть розглянуті у підрозділі 1.3.5. Слід відзначити, що оскільки ці коди є надлишковими, під час проведення інформаційних оцінок в системах, де вони використовуються, необхідно розраховувати загальну кількість розрядів, яка завжди буде більшою, ніж кількість інформаційних розрядів. Для проведення відповідних розрахунків слід користуватися формулою (1.1) [2 – 6, 8, 24]. Способи оцінювання кількості контрольних розрядів у завадостійких кодах розглядалися у п'ятому розділі другої частини посібника [73] та будуть додатково розглянуті у підрозділі 2.2.

Цифрове кодування сигналів в електронних системах та в системах зв'язку має такі переваги [1, 24, 28, 29].

1. Поліпшення характеристик цифрової послідовності через зменшення довжини неперервних послідовностей одиниць та нулів, що зменшує ймовірність похибок через нестабільність частоти тактового генератора у синхронних цифрових системах. Цей недолік не впливає на роботу цифрових систем у разі використання кодів із самосинхронізацією, зокрема

манчестерських кодів.

2. Придушення постійної складової у спектрі цифрового сигналу. Відповідні способи кодування цифрових сигналів розглядалися у підрозділі 1.2.

3. Поліпшення умов синхронізації за рахунок використання кодів із самосинхронізацією. Проте загальним недоліком сигналів із самосинхронізацією є те, що вони, зазвичай, мають більшу частотну смугу (рис. 1.34, 1.35).

4. Звуження частотної смуги сигналу за однакової швидкості передавання інформації з метою підвищення ефективності використання каналу зв'язку. Тут можуть також бути використання засоби ущільнення сигналів, які розглядалися у підрозділі 7 першої частини посібника [1] та будуть досконало розглянуті у цій частині у розділі 4.

5. Використання у приймальному обладнанні електронних засобів виправлення помилок на фізичному рівні еталонної моделі OSI, яка розглядалася у підрозділі 2.3.2 першої частини посібника [1].

Форми подання двійкових електричних та оптичних сигналів в електронних системах на фізичному рівні розглядалися у підрозділі 1.2.2 цієї частини посібника. У цьому підрозділі розглядатимуться способи логічного кодування сигналів.

1.3.2 Код ASCII

Код ASCII (абревіатура англійського словосполучення **A**merican **S**tandard **C**ode for **I**nformation **I**nchenging, американський стандартний код для обміну інформацією) широко використовується у комп'ютерній техніці та у комп'ютерних системах для подання цифрової та текстової інформації у разі її обробки та виведення на зовнішні пристрої [13, 14, 39, 40]. Слід відзначити, що у 60-80-х роках XX століття існували різні стандарти подання цифрової інформації в ЕОМ, і організація зв'язку між обчислювальними машинами різних стандартів потребувала застосування складних пристроїв та програмних засобів для перекодування інформації [69]. Це у значній мірі

стримувало розвиток комп'ютерної індустрії та глобальних комп'ютерних мереж [15 – 20]. Проте тепер, починаючи з вісімдесятих років минулого сторіччя, можна сказати, що код ASCII фактично є стандартом подання інформації в обчислювальній техніці [64, 65]. Відмінною рисою коду ASCII є те, що, як універсальний засіб для подання цифрової та текстової інформації у двійковій формі, він об'єднує в собі риси як двійкових, так і комбінаторних кодів. Це зайвий раз підтверджує те, що класифікація кодів, яка була наведена у підрозділі 1.1.2, є досить умовною.

До кодів ASCII входять наступні групи символів [39, 40].

1. Символи керування форматом тексту.

2. Символи керування введенням тексту.

3. Коди цифр.

4. Коди літер англійської та національних абеток.

5. Коди спеціальних символів.

У комп'ютерній техніці код ASCII використовується для введення інформації та для виведення результатів розрахунків. Відповідно, машинний код комп'ютерних програм будується наступним чином [39, 40].

1. Введення числових та текстових даних у коді ASCII.

2. Переведення числових даних із формату ASCII до двійкового формату.

3. Проведення обчислень та обробки текстової інформації.

4. Переведення отриманих результатів із двійкового формату до коду ASCII.

5. Виведення даних на зовнішні пристрої.

Для здійснення операцій введення та виведення даних використовується або безпосереднє звернення до портів зовнішніх пристроїв, або система переривань [39, 40]. Слід відзначити, що, у разі програмування мовами високого рівня, операції введення та виведення даних організуються через відповідні функції користувача, у яких перекодування даних із формату ASCII до двійкового формату здійснюється автоматично. Проте у разі

програмування мікропроцесорів та мікроконтролерів з використанням мови ASEMBLER таке перекодування необхідно здійснювати вручну під час написання програм з використанням відповідних кодів команд [39, 40]. Коди ASCII використовуються також для передачі текстових повідомлень у комп'ютерних мережах. Таблиця кодів ASCII наведена у додатку Г [39, 40].

1.3.3 Двійкові коди для від'ємних та дробових чисел та особливості їхнього використання у обчислювальній техніці

Для кодування від'ємних чисел у двійковій арифметиці використовують наступний алгоритм.

1. Інверсія бітів додатного двійкового числа.

2. Додавання до отриманого результату одиниці.

Наведемо приклад запису у двійковому форматі від'ємного числа – 65.

Число 65	01000001	
Інверсія бітів	10111110	
Додати 1	10111111	Число –65

Таке кодування від'ємних чисел називається кодом із доповненням до одиниці (англійський термін **C**omplement **C**ode) [39, 40].

Перевагою запису двійкових чисел у коді із доповненням до одиниці є виконання принципу комутативності. Тобто, якщо число $-b$ записано у коді із доповненням до одиниці, для будь-яких чисел a та b виконується тотожність: $a - b = a + (-b)$. Це правило означає, що за умови запису всіх від'ємних чисел у коді із доповненням до одиниці, всі арифметичні операції можна здійснювати автоматично, не звертаючи увагу на знак числа.

Єдина проблема полягає у тому, що у разі подання від'ємних чисел у коді із доповненням до одиниці втрачається однозначність між десятковим числом та його двійковим кодом. Дійсно, двійкове число 10111111 можна трактувати як від'ємне число –65, а можна як додатне число 191. Проте,

згідно із сформульованим вище принципом комутативності, під час проведення арифметичних обчислень цієї проблеми не виникає. Вона має місце у двох випадках: у разі здійснення операції порівняння даних та у разі виведення результатів розрахунків. Тому для коректного виведення результатів розрахунків під час написання програм мовою ASEMBLER програмістам необхідно контролювати знак числа, для цього використовується біт знаку регістру стану процесора [39, 40]. Щодо здійснення операцій порівняння, тут, для уникнення плутанини, використовуються окремі команди процесора для порівняння знакових і беззнакових даних [39, 40].

Тобто, у разі правильного здійснення операцій порівняння та операцій виведення результатів розрахунків, використання коду із доповненням до одиниці є найбільш ефективним засобом для проведення розрахунків із додатними та від'ємними числами [39, 40]. Тому код із додаванням до одиниці використовується у цифрових системах керування та вимірювання для подання інформації про аналогову величину, яка вимірюється, і зазвичай відповідне число може мати як додатне, так і від'ємне значення.

Наведений вище алгоритм формування додаткового коду від'ємного числа G у математичній формі можна записати наступним чином:

$$G_{\text{дод}} = \begin{cases} G, & \text{за умови } G \geq 0; \\ 2^n - |G|, & \text{за умови } G < 0, \end{cases} \tag{1.19}$$

де n – розрядність числа G. Якщо число, яке є інверсним до двійкового числа G, позначити як $G_{\text{інв}}$, тоді для від'ємних цілих чисел має місце тотожність [39, 40, 44]:

$$G_{\text{дод}} = G_{\text{інв}} + 1, \tag{1.20}$$

а для від'ємних дробів із фіксованою комою [39, 40, 44 – 46]:

$$G_{\text{дод}} = G_{\text{інв}} + 2^{-(n-1)}. \tag{1.21}$$

Наведемо приклад подання від'ємного числа у форматі з фіксованою комою.

Приклад 1.1. Записати число –5,375 у форматі із фіксованою комою, вважаючи що розрядна сітка ЕОМ має 4 розряди.

Оскільки розрядна сітка ЕОМ має 4 розряди, використаємо для запису дрібного числа у форматі із фіксованою комою дві комірки пам'яті. Спочатку запишемо у додатковому коді число -5. Згідно із наведеним алгоритмом та із формулою (1.18) маємо:

$$-5_{10} = \left(\overline{0101} + 1\right)_2 = 1011_2.$$

Тепер необхідно сформувати код дрібної частини числа. Розрядна сітка для неї формується наступним чином [39, 40, 44 – 46]: $2^{-1} = \frac{1}{2}$, $2^{-2} = \frac{1}{4}$, $2^{-3} = \frac{1}{8}$, $2^{-4} = \frac{1}{16}$. Таким чином, дробову частину числа {-5,375} формуємо наступним способом:

$$0{,}375 = \frac{3}{8} = \frac{1}{4} + \frac{1}{8} = 2^{-2} + 2^{-3} = 0.0110_2.$$

Тоді від'ємне дробове число -5,375 записується у додатковому коді наступним чином. До першої чотирирозрядної комірки заноситься число 1011, а до другої – відповідно число 0110. Остаточно маємо:

$$-5{,}375_{10} = 1011.0110_2.$$

Для подання чисел із плаваючою комою в комп'ютерній техніці використовують зсунутий код, який формується наступним чином [39, 40, 44 – 46]:

$$G_{\text{зс}} = \begin{cases} 2^{n-1} + |G|, & \text{за умови } G \geq 0; \\ 2^{n-1} - |G|, & \text{за умови } G < 0. \end{cases} \qquad (1.22)$$

За умови використання зсунутих кодів спрощується порівняння чисел із знаком, яке у даному випадку зводиться до порівняння абсолютних величин двох чисел. У комп'ютерній арифметиці існує теорема про те, що за умови $G_1 > G_2$ виконується тотожність $G_{\text{зс}1} > G_{\text{зс}2}$ [39, 40, 44 – 46].

Для подання порядку числа із плаваючою комою використовують також зсунутий код із від'ємним нулем, який є схожим на звичайний

зсунутий код та формується наступним чином [39, 40, 44 – 46]:

$$G_{\text{зс}} = \begin{cases} 2^{n-1} + |G| + 1, & \text{за умови } G \geq 0; \\ 2^{n-1} - |G| + 1, & \text{за умови } G < 0. \end{cases} \tag{1.23}$$

Як відомо, головною проблемою запису дробових чисел у комп'ютерній техніці та їхньої обробки є проблема округлення, яка безпосередньо пов'язана із точністю обчислень. Якщо у десятковій системі числення у скінченній формі можна записати такі дроби, знаменник яких розкладається на степені множників чисел 5 та 2, то у двійковій системі у скінченній формі записуються лише дроби, знаменники яких є множниками числа 2. Наприклад, дріб $\frac{1}{25} = \frac{1}{5^2} = 0{,}04$ у десятковій системі числення є скінченним числом, а у випадку подання у двійковій системі цей дріб буде нескінченним. Тому обмеження кількості розрядів у запису дрібного числа є серйозною проблемою під час організації комп'ютерних розрахунків. Невиправдане збільшення кількості розрядів у випадку організації циклів із великою кількістю повторень зазвичай призводить не до збільшення, а до зменшення точності розрахунків. Відповідні оцінки оптимальної кількості розрядів двійкового числа для різних чисельних методів наводяться у підручниках [11, 12].

1.3.4 Рефлексні коди та код Грея

Перед вивченням цього підрозділу необхідно повторити розділи 2 та 5 другої частини посібника

Розглянемо засоби натурального кодування, які застосовуються для формування двійкових послідовностей в цифрових системах обробки інформації. Насамперед це рефлексні коди (англійський термін reflect code), ця назва пояснюється симетричним розташуванням елементів у деяких розрядах таких кодів. Використання рефлексних кодів для обробки та передавання інформації пов'язане, по-перш за все, із одним істотним

недоліком звичайного двійкового кодування чисел. У простому двійковому коді у разі переходу від одного числа до наступного може здійснюватися одночасна зміна декількох розрядів числа, що під час перетворення аналогової інформації до цифрової форми може приводити до виникнення великої кількості помилок. Наприклад, у разі зміни числа 7 (0111_2) на число 8 (1000_2) змінюються всі чотири розряди двійкового запису числа.

За умови використання рефлексних кодів вдається уникнути великої кількості помилок, обумовлених одночасною зміною розрядів числа. Рефлексні коди будуються таким чином, що у разі переходу від коду одного числа до коду наступного, сусіднього числа, змінюється значення лише один із його розрядів.

Розглянемо код Грея, який є одним із самих поширених рефлексних кодів. Цей код відноситься до класу непозиційних, оскільки, на відміну від двійкового коду, його розряди не мають відповідних вагових коефіцієнтів. Двійкові коди чисел від 0 до 15 та відповідні їм коди Грея наведені у таблиці 1.2.

Таблиця 1.2 – Кодові послідовності для коду Грея

Число	Двійковий код	Код Грея	Число	Двійковий код	Код Грея
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

Із таблиці 1.2 видно, що код Грея відповідає системі кодування сигналів за методом Хармута – Уолша, розглянутій у другій частині посібника у розділі 5 [48].

З іншого боку, для будь якого двійкового числа код Грея можна легко отримати за наступним, простим рекурентним алгоритмом.

1. Починаємо обробку числа із старшого розряду, і цей старший розряд

залишаємо без змін.

2. Для визначення наступних розрядів коду сумуємо поточний та попередній розряди числа за модулем 2. Тобто:

$$g_n = a_n; g_{n-1} = a_n \oplus a_{n-1}; n = N, N-1, \ldots, 1, \quad (1.24)$$

де a_n – розряди двійкового числа, g_n – розряди коду Грея, N – загальна кількість розрядів. Наочна ілюстрація алгоритму переведення двійкового коду числа до коду Грея наведена на рис. 1.35, а.

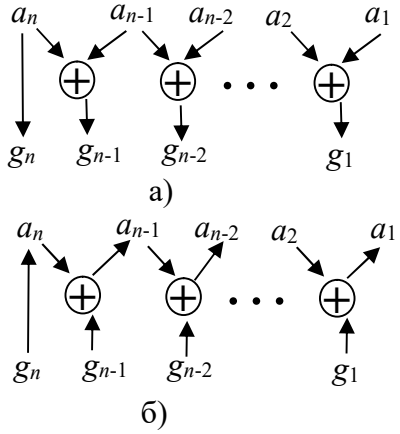


Рис. 1.38 Наочна ілюстрація алгоритмів перетворення двійкового числа до коду Грея (а) та зворотного перетворення (б)

Тобто, згідно із формулою (1.24) та рис. 1.38, а, для формування із двійкового числа його коду Грея необхідно додати за модулем два числа: початкове двійкове число та число, зсунуте вліво на 1 розряд [2 – 6, 8].

Зворотне перетворення коду Грея до двійкового числа здійснюється наступним чином.

1. Починаємо обробку коду із старшого розряду і переносимо старший розряд коду без змін до старшого розряду двійкового числа.

2. Для визначення наступних розрядів сумуємо за модулем 2 попередній, обчислений розряд числа, із поточним розрядом коду, як показано на рис. 1.38, б. Тобто:

$$a_n = g_n; a_{n-1} = a_n \oplus g_{n-1}; n = N, N-1, \ldots, 1. \quad (1.25)$$

Згідно із формулою (1.25) та рис. 1.38, б, для здійснення зворотного перетворення коду Грея до двійкового числа необхідно додати за модулем 2 два числа: закодоване число, та число, яке створене із закодованого через

зсунення праворуч на 1 розряд [2 – 6, 8].

Приклад 1.2. Побудувати код Грея для числа $8 = 1000_2$ та декодувати отриманий результат.

Для отримання коду Грея використаємо послідовно формулу (1.24).

1. $g_4 = a_4 = 1$;

2. $g_3 = a_4 \oplus a_3 = 0 \oplus 1 = 1$;

3. $g_2 = a_3 \oplus a_2 = 0 \oplus 0 = 0$;

4. $g_1 = a_2 \oplus a_1 = 0 \oplus 0 = 0$.

Тобто, для числа 8 отримуємо код Грея 1100, що відповідає значенню, наведеному у таблиці 1.2.

Зворотне перетворення коду Грея до двійкового числа здійснюється за рекурентним співвідношенням (1.25) наступним чином.

1. $a_4 = g_4 = 1$;

2. $a_3 = a_4 \oplus g_3 = 1 \oplus 1 = 0$;

3. $a_2 = a_3 \oplus g_2 = 0 \oplus 0 = 0$;

4. $a_1 = a_2 \oplus g_1 = 0 \oplus 0 = 0$.

Використання кодів Грея значно спрощує обробку числових даних в електронних системах. Тому в більшості випадків рефлексні коди застосовуються у апаратних пристроях, призначених для обробки числових даних, але вони знаходять широке застосування і у системах зв'язку. Використання кодів Грея замість двійкових кодів дозволяє також економити електроенергію, яка витрачається на перемикання регістрів збереження даних та комірок пам'яті обчислювальних систем під час проведення ітераційних обчислень [2, 3, 54].

Вага одиниць у відповідному розряді коду Грея j визначається із співвідношення [54]:

$$S = \sum_{i=0}^{j} 2^i. \qquad (1.26)$$

Формула (1.26) використовується наступним чином. Знак сумування членів ряду (1.26) є додатним для всіх парних одиниць коду Грея, та

від'ємним для всіх непарних одиниць. Під час формування суми (1.26) розряди коду Грея нумеруються зліва направо, а розряди, в яких стоять нулі, в суму не записуються. Результатом використання співвідношення (1.26) є десяткове число, еквівалентне відповідному коду. Наведемо приклад використання співвідношення (1.26).

Приклад 1.3. Знайти десяткове число, якщо його код Грея становить 1101101.

Згідно із співвідношенням (1.26) маємо:

$$[1101101]_g = \sum_{i=0}^{6} 2^i + \sum_{i=0}^{5} 2^i - \sum_{i=0}^{3} 2^i + \sum_{i=0}^{2} 2^i + \sum_{i=0}^{0} 2^i = 2^6 - 2^3 + 2^0 = 64 - 8 + 1 = 57_{10}.$$

Розглянемо інший приклад, який наочно демонструє ефективність використання кодів Грея замість звичайного двійкового коду у разі проведення циклічних обчислень із зміною вмісту регістра-лічильника на 1.

Приклад 1.4. Відомо, що імовірність помилки у одному розряді регістру під час зміни його вмісту з 1 на 0 або з 0 на 1 становить 10^{-3}. Знайти сумарну імовірність помилки у разі зміни значення регістру з числа 7_{10} на число 8_{10} за умови використання коду Грея та звичайного двійкового коду. Розрахувати енергію, яка необхідна для перемикання регістру в першому і другому випадках, якщо один розряд регістру під час перемикання споживає енергію 10 мВт.

За умовою задачі значення регістру змінюється з величини $7_{10} = 0111_2$ на величину $8_{10} = 1000_2$, тобто, у двійковому коді числа змінюються 4 розряди. За умови використання коду Грея у цьому випадку змінюється лише 1 розряд регістра. Тобто, імовірність помилки у разі використання коду Грея відповідає імовірності помилки у одному розряді і становить 10^{-3}. У разі використання звичайного двійкового коду змінюються 4 розряди регістра, тому, з урахуванням імовірності подвійних, потрійних помилок, а також помилки у всіх чотирьох розрядах, маємо:

$$P_\Sigma = C_4^1 \cdot 10^{-3} + C_4^2 \cdot 10^{-6} + C_4^2 \cdot 10^{-9} + C_4^4 \cdot 10^{-12} \approx 0{,}00406.$$

Тобто, імовірність одночасної помилки у трьох або чотирьох розрядах є

невисокою і її можна нехтувати, проте імовірність помилки у одному розряді у разі перемикання чотирьох розрядів регістру є в 4 рази більшою, ніж за умови використання коду Грея, коли перемикається лише 1 розряд регістра.

Щодо енергії, яка споживається, то у разі використання коду Грея перемикається 1 розряд регістра і для цього, за умовою задачі, необхідна енергія 10 мВт, а у разі використання двійкового коду за тих же умов споживається енергія 40 мВт.

Із наведеного прикладу зрозуміло, у разі використання коду Грея імовірність помилки обчислювального пристрою є значно меншою і регістр-лічильник споживає значно меншу кількість енергії.

Одним із цікавих та ілюстративних способів подання коду Грея є використання гіперкубів, які тісно пов'язані із теорією графів та ґраток, що розглядалися у другому та сьомому розділах другої частини посібника [48, 50]. Такий спосіб подання для лінійних завадостійких кодів також був розглянутий у розділі 5 другої частини посібника [48]. Для коду Грея цей спосіб подання полягає у тому, що всі сусідні за ребрами двійкові числа, які лежать в вершинах кубу, відрізняються лише одним розрядом. Приклад гіперкубу для тризначного коду Грея показаний на рис. 1.39.

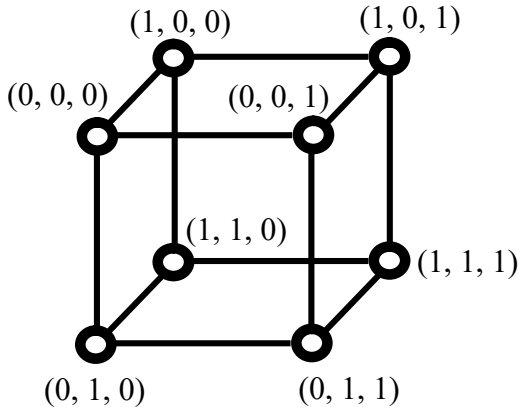


Рис. 1.39 Приклад гіперкубу для трирозрядного коду Грея

У додатку Д наведений код комп'ютерної програми **greycode**, призначеної для формування кодів Грея та для зворотного перетворення коду Грея до двійкового числа. Програма написана з використанням засобів

матричного програмування системи науково-технічних розрахунків MatLab. Для написання програми були використані співвідношення (1.24) та (1.25), записані у вигляді арифметико-логічних виразів (1.9).

Проблема полягає лише у тому, що у розглянутих вище теоретичних міркуваннях та у співвідношеннях (1.24), (1.25), для формування бітових послідовностей коду Грея використовується запис двійкового числа зправа-наліво у порядку зростання розрядів, проте у векторах числа впорядковані навпаки, зліва-направо. Ця проблема пов'язана лише із тим, що в арифметиці європейці користуються арабською формою запису чисел, і на неї автори вже неодноразово звертали увагу у другій частині посібника [48 – 50]. Оскільки програма для обробки чисел, записаних у форматі коду Грея, орієнтована на використання матриць та векторів, у ній застосований порядок обробки чисел зліва-направо, як це прийнято у матричній алгебрі, але початковий вектор коду двійкового числа, для зручності, вводиться у форматі зправо-наліво, оскільки у повсякденній практиці ми користуємося саме таким записом. Тобто, на першій позиції вхідного вектора стоїть старший розряд, а на останній – молодший. У такому ж форматі запису розрядів, зправо-наліво, виводяться і результати обчислень. Для перетворення формату подання вектора бітів числа зручно використовувати множинну індексацію системи MatLab [13, 14]. Найпростіше для такого перетворення застосовувати наступний набір системних команд [13, 14]:

```
n=length(c);
d=c(n:-1:1);
```

З урахуванням такої форми подання двійкових чисел рекурентне співвідношення (1.24) у формі арифметико-логічного виразу (1.9) записується наступним чином:

$$g_i = (i = 1) \cdot a_i + (i > 1) \cdot a_i \oplus a_{i-1},\ i = 1 : n - 1, \qquad (1.27)$$

де a_i –вектора **A** вхідної послідовності бітів довжини n, g_i – елементи вектора **G** вихідної послідовності.

Відповідно, для зворотного перетворення, з метою отримати із коду

Грея код звичайного двійкого числа, згідно із співвідношенням (1.25) та із обраною формою запису чисел, арифметико-логічний має вигляд:

$$a_i = (i = 1) \cdot g_i + (i > 1) \cdot a_{i-1} \oplus g_i,\ i = 1 : n - 1. \qquad (1.28)$$

Для проведення рекурентних розрахунків з використанням співвідношень (1.27), (1.28), у програмі **`greycode`** викликається процедура **`recvectbin`**, головні особливості написання якої розглядалися у підрозділі 1.2.2. Це зайвий раз свідчить про те, що розглянуті методи матричного програмування є досить універсальними та можуть бути пристосовані для коректного запису різних рекурентних функцій обробки двійкових послідовностей, які часто використовуються у теорії кодування цифрових сигналів.

1.3.5 Коди 4B/5B та 8B/6T

Як було відмічено у підрозділі 1.3.1 цієї частини посібника, однією із головних задач кодування сигналів у системах зв'язку є уникнення у послідовностях, які передаються, великої кількості нулів, що слідують один за одним. Для вирішення цієї задачі використовують дві універсальні технології кодування сигналів. Перша з них є найпростішою і полягає у тому, що свідомо збільшується кількість бітів числа, і через це в створеному коді уникають послідовностей, які містять велику кількість нулів. Такий засіб обробки даних дещо нагадує завадостійкі коди, але оскільки код не містить контрольних розрядів, з точки зору теорії кодування такі коди слід розглядати як натуральні. Хоча, з іншого боку, вони мають дозволені та заборонені послідовності та відповідну коректувальну здатність. Дійсно, у апаратних засобах обробки натуральних кодів із збільшеною кількістю розрядів зазвичай передбачається фільтрація заборонених послідовностей, але такі перевірки не мають системний характер, тому коректувальну здатність натурального коду вкрай важко оцінювати та прогнозувати. Тобто, розгляд практичних реалізацій різноманітних кодів ще раз свідчить про те, що класифікація кодів, яка була наведена у підрозділі 1.1, є досить умовною.

Проте така класифікація суттєво спрощує аналіз властивостей кодів, а також дозволяє краще зрозуміти їх призначення та можливості використання.

Сьогодні найбільш поширеним серед натуральних кодів із збільшеною кількістю розрядів є двійковий код 4B/5B, який використовується у проводових комп'ютерних мережах Fast Ethernet та оптичних мережах FDDI [15 – 23]. Основою цього коду є запис чотирьох біт інформації у вигляді 5 біт, тобто у код додається 1 розряд, і за рахунок цього видаляються послідовності, які містять значну кількість нулів, що слідують один за одним. Коди 4B/5B для чисел від 0 до 15 наведені у таблиці 1.3.

Як видно з наведених даних, у коді 4B/5B немає жодної послідовності, яка б містила більше двох нулів підряд. Використання цього коду гарантує, що для будь-якого сполучення кодових символів повідомлення не буде містити більше трьох нулів підряд.

Частотний спектр сигналу для коду 4B/5B наведений на рис. 1.40 [17 – 20, 49]. Вимірювання проводились для різних послідовностей нулів та одиниць вхідної кодової комбінації та усереднювались за ансамблем.

Із рис 1.34 та 1.40 можна зробити висновок, що спектр сигналу для коду 4B/5B за формою близький до спектру коду B8ZS. Різниця між спектрами цих двох сигналів полягає в тому, що спектр коду 4B/5B зсунутий праворуч на частоту $\frac{F_c}{4}$, що пов'язано із більш високою частотою роботи системи за умови однакової швидкості передавання інформації.

У логічному коді 8B/6T для кодування восьми біт вхідної інформації використовують шість сигналів, кожний із яких має три стани. Тобто, на $2^8 = 256$ вхідних послідовностей припадає $3^6 = 729$ вихідних послідовностей, тому надлишковість коду 8B/6T є вищою, ніж коду 4B/5B [15 – 23]. Таблиці перекодування із звичайного двійкового формату до кодів 4B/5B або 8B/6T досить просто реалізуються в електронному обладнанні, і такі кодери та декодери незначно ускладнюють електронні приймально-передавальні системи [24, 38, 58].

Таблиця 1.3 – Двійкові послідовності для коду 4B/5B

Число	Двійковий код	Код 4B/5B	Число	Двійковий код	Код 4B/5B
0	0000	11110	8	1000	10010
1	0001	01001	9	1001	10011
2	0010	10100	10	1010	10110
3	0011	10101	11	1011	10111
4	0100	01010	12	1100	11010
5	0101	01011	13	1101	11011
6	0110	01110	14	1110	11100
7	0111	01111	15	1111	11101

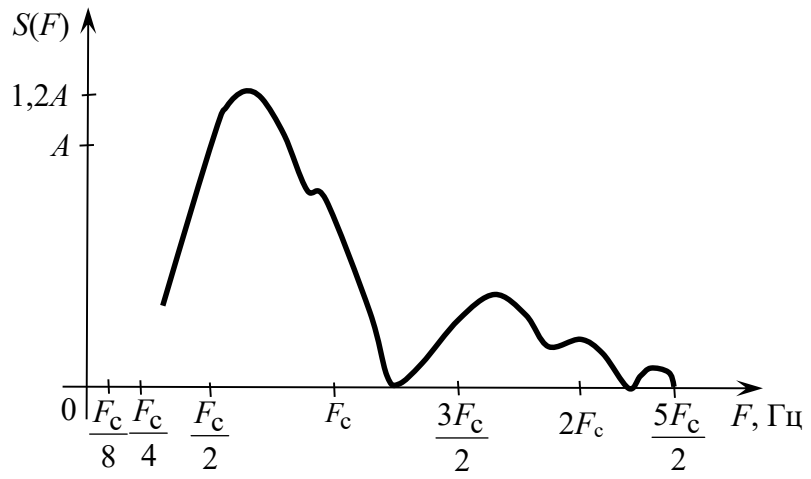


Рис. 1.40 Частотний спектр сигналу, закодованого кодом 4B/5B

Комп'ютерна програма **code4b5b**, призначена для побудови коду 4B/5B та для переведення бітових послідовностей коду 4B/5B до натурального двійкового коду, наведена у додатку Е. Робота цієї програми основана на використанні матриці відповідності для запису чисел у натуральному двійковому коді та у коді 4B/5B, аналогічної таблиці 1.3. П'ятибітові коди чисел від 0 до 15 записуються у відповідних рядках цієї матриці. У програмі **code4b5b**, як і у програмі для формування кодів Грея **greycode**, використаний звичний та зручний формат подання вхідних та

вихідних бітових послідовностей двійкових чисел, вони за номерами розрядів читаються зправа-наліво, оскільки у цій програмі використання методів матричного аналізу та обробки векторів не передбачено. У разі необхідності порозрядної обробки цих бітових послідовностей можна змінити порядок елементів в отриманому вихідному векторі за допомогою простих командних рядків, наведених у підрозділі 1.3.4.

Розрахований спектр сигналу коду 4B/5B для вхідної кодової послідовності 10000001 за умови використання коду AMI наведений на рис. 1.41, там же, для порівняння, показаний відповідний спектр двійкового сигналу без додаткового кодування. Для побудови спектрів сигналів використовувалась програма, наведена у додатку В.

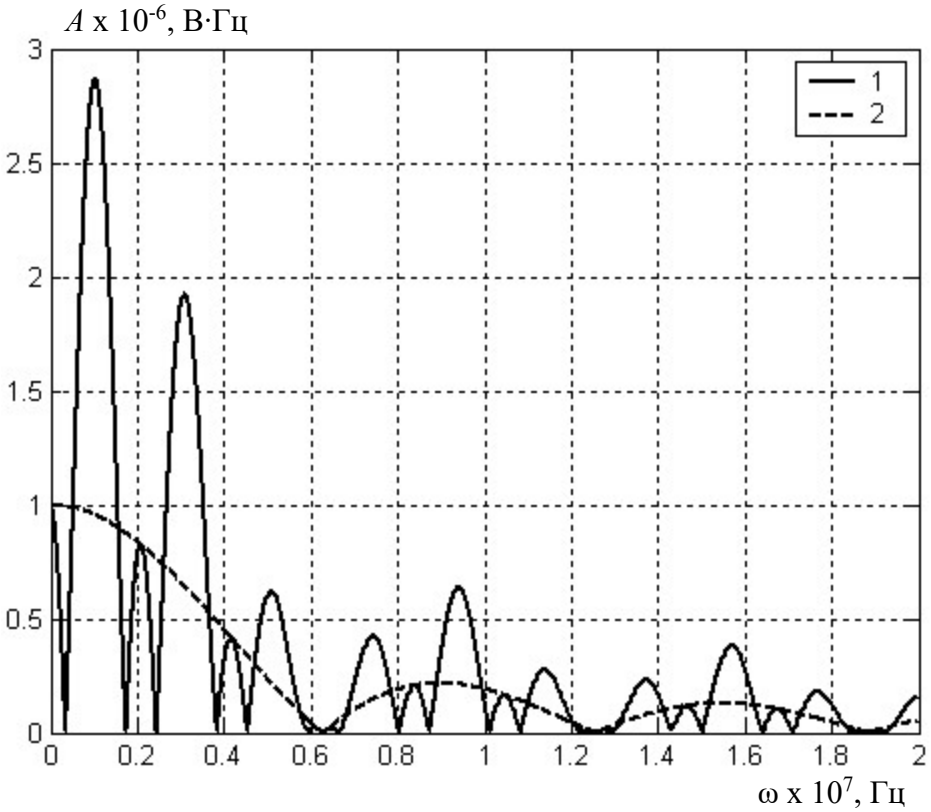


Рис. 1.41 Спектри сигналів для кодової послідовності 10000001, сформованих з використанням коду 4B/5B (1) та без додаткового кодування (2) за умови застосування для формування електричного сигналу коду AMI

Зрозуміло, що сигнал з використанням коду 4B/5B має май же таку ж саму частотну смугу, що й сигнал без додаткового кодування, проте його

постійна складова є значно меншою, а енергія – більшою, що обумовлено наявністю більшої кількості одиниць. Це значно спрощує розпізнавання сигналу з кодом 4B/5B на фоні завад та зменшує імовірність помилкового передавання інформації. Максимальна амплітуда сигналу, закодованого кодом 4B/5B, відповідає частотам ω_c та $3 \cdot \omega_c$.

1.3.6 Принципи та алгоритми скрамблювання

Перед вивченням цього підрозділу необхідно повторити підрозділ 3.4 другої частини посібника

Алгоритми скрамблювання (від англійського слова scramble – куча, або сміття), на відміну від простих кодів із збільшеною кількістю розрядів числа, зокрема кодів 4B/5B та 8B/6T, не вносять у початкову послідовність додаткових розрядів. Проте головним завданням скрамблювання є зменшення у двійковому запису кількості нулів, які слідують один за одним. Цей спосіб кодування дозволяє одночасно вирішити дві важливі проблеми, а саме, проблему синхронізації сигналу і проблему придушення постійної складової [30, 33]. Сутність алгоритмів скрамблювання зазвичай полягає у обчисленні розрядів коду через відомі біти двійкового числа з використанням рекурентних співвідношень. Тому числові коди, отримані з використанням алгоритмів скрамблювання, можна розглядати як ітераційні. З іншого боку, це натуральні коди, оскільки кількість розрядів у початковому числі та у кодовій послідовності є однаковою. Алгоритми скрамблювання також іноді називають алгоритмами перемішування [30, 33]. Узагальнена схема системи зв’язку із кодуванням інформації за принципом скрамблювання наведена на рис. 1.42. Основними елементом цієї схеми є генератори змішаної послідовності, які встановлені у передавальному та приймальному пристроях системи зв’язку.

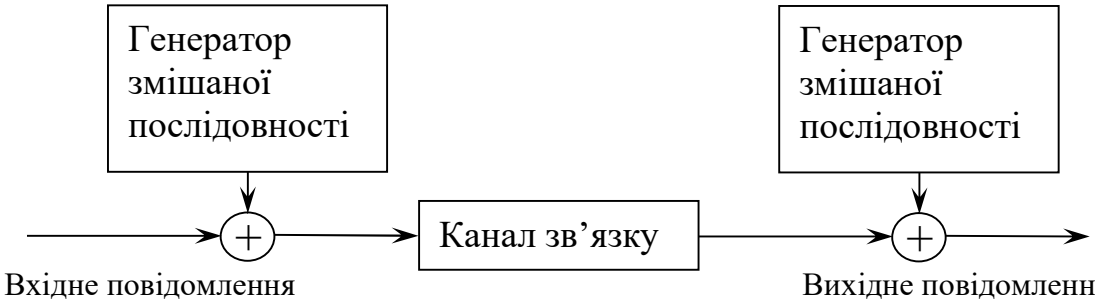


Рис. 1.42 Узагальнена схема системи зв'язку із кодуванням повідомлень за принципом скремблювання

Розглянемо приклад простого рекурентного алгоритму скремблювання [19, 20, 28, 29]. Перші п'ять розрядів коду B_i обчислюються наступним чином:

$$B_1 = A_1;\ B_2 = A_2;\quad B_3 = A_3;\qquad B_4 = A_4 \times B_1;\qquad B_5 = A_5 \times B_2. \tag{1.29}$$

Для подальших розрядів, починаючи з шостого, використовується рекурентне співвідношення:

$$B_i = A_i \times B_{i\text{-}3} \times B_{i\text{-}5};\ i = 6\ldots N, \tag{1.30}$$

де A_i – розряди двійкового числа, B_i – розряди коду, який формується, N – кількість розрядів числа, × – операція об'єднання чисел за модулем 2, або операція „виключного або" для двох або трьох елементів. Для двох елементів ця операція еквівалентна операції XOR, розглянутій у підрозділі 1.2.2. У разі, якщо порівнюється три або більша кількість бітів, результат порівняння дорівнює 0, якщо кількість одиниць є парною та 1 у противному випадку [28, 29, 38, 52 – 54].

Приклад 1.5. З використанням рекурентного алгоритму скремблювання побудувати код дванадцятирозрядного двійкового числа A = 110110000001.

Згідно із співвідношеннями (1.29) для перших п'яти розрядів коду маємо:

$B_1 = A_1 = 1;\qquad B_2 = A_2 = 1;\qquad B_3 = A_3 = 0;\qquad B_4 = A_4 \times B_1 = 1 \times 1 = 0;$

$B_5 = A_5 \times B_2 = 1 \times 1 = 0;$

Далі, за умови відомих значень розрядів $B_1 - B_5$ та $A_1 - A_{12}$, використовуємо рекурентне співвідношення (1.30):

$B_6 = A_6 \times B_3 \times B_1 = 0 \times 0 \times 1 = 1$; $B_7 = A_7 \times B_4 \times B_2 = 0 \times 0 \times 1 = 1$;

$B_8 = A_8 \times B_5 \times B_3 = 0 \times 0 \times 0 = 0$; $B_9 = A_9 \times B_6 \times B_4 = 0 \times 1 \times 0 = 1$;

$B_{10} = A_{10} \times B_7 \times B_5 = 0 \times 1 \times 0 = 1$; $B_{11} = A_{11} \times B_8 \times B_6 = 0 \times 0 \times 1 = 1$;

$B_{12} = A_{12} \times B_9 \times B_7 = 1 \times 1 \times 1 = 1$.

Тобто, остаточний результат використання співвідношень (1.29, 1.30) такий: B = 110001101111. Як бачимо, застосування обраного нами алгоритму скрамблювання дозволило зменшити у кодовій послідовності кількість нулів, які слідують один за одним, від шести до трьох. Взагалі, у теорії чисел доведена теорема про те, що у разі перетворення чисел з використанням рекурентних співвідношень (1.29, 1.30), кількість нулів, які слідують один за одним, не перевищує трьох.

Зворотний процес дескрамблювання виконується із використанням таких рекурентних співвідношень:

$$C_1 = B_1; \quad C_2 = B_2; \quad C_3 = B_3;$$
$$C_4 = (B_4 \times B_1) \times B_1; \quad C_5 = (B_5 \times B_2) \times B_2; \qquad (1.31)$$
$$C_i = (B_i \times B_{i-3} \times B_{i-5}) \times B_{i-3} \times B_{i-5};\ i = 6 \ldots N.$$

Алгоритми скрамблювання знайшли широке втілення у сучасних технологіях перетворення інформації в комп’ютерних мережах. Вони втілені у стандартах проводових мереж Fast Ethernet та Gigabit Ethernet, а також у стандартах безпроводових мереж Wi-Fi та WiMAX. Перевагою методів кодування, основаних на скрамблюванні, є відсутність надлишкових розрядів у коді числа. Проте ці алгоритми є досить ресурсоємними і потребують деякого часу на здійснення процесів кодування та декодування. Можливості знаходження та виправлення помилок у скрамблювальних кодах відсутні, для виконання приймально-передавальною системою цих функцій слід додатково використовувати завадостійкі коди, які будуть описані у розділах 2 та 3 цієї частини посібника. Слід відзначити, що сьогодні алгоритми скрамблювання та дескрамблювання у приймально-передавальному обладнанні реалізовані на рівні електронних схем, тому виконання цих операцій здійснюється досить швидко.

У деякій літературі до кодів, побудованих за алгоритмами скремблювання, відносять також коди B8ZS та HDB3, які розглядалися у підрозділі 1.2.4 [24]. Проте слід зауважити на те, що алгоритми скремблювання передбачають математичну обробку вхідної інформації та проведення рекурентних обчислень, а коди B8ZS та HDB3 є простими натуральними кодами, які формуються із дозволених та заборонених двійкових послідовностей. Тобто, скремблювання слід розглядати як засіб логічного формування двійкових послідовностей, а коди B8ZS та HDB3 формуються апаратним способом на рівні обробки двійкових сигналів. Якщо звернутися до еталонної моделі OSI, коди B8ZS та HDB3 відповідають фізичному рівню, а алгоритми скремблювання – канальному. Тому ці способи кодування сигналів не є взаємопов'язаними і бажано розглядати їх окремо.

З практичної точки зору ефективний генератор перемішаної послідовності, оснований на алгоритмі скремблювання, можна реалізувати на регістрі зсуву із зворотним зв'язком [28 – 30, 33, 52, 53]. У цьому разі регістр зсуву проектується так, щоб бітова послідовність вихідного сигналу була пристосована для передавання через канал зв'язку із мінімальними спотвореннями. Така бітова послідовність має починатися з одиниці. Тоді, якщо правильно підібрати необхідні зворотні зв'язки, регістр зсуву, який має n каскадів, буде генерувати псевдовипадкову послідовність одиниць та нулів із періодом $2^n - 1$ без повторень. Для забезпечення правильної роботи кодувального та декодувального пристроїв за алгоритмом скремблювання необхідно відповідним чином встановити зворотні зв'язки. Вони формуються наступним чином. Якщо існує регістр зсуву розрядності n, необхідно обрати примітивний поліном ступені n, який не можна розбити на множники і який ділиться на $2^k + 1$, де k – кількість інформаційних розрядів числа, яке кодується. Наприклад, для п'ятирозрядного регістра непривідним поліномом може бути поліном $x^5 + x^2 + 1$, якому відповідає двійкове число 10011. Теорія двійкових поліномів була досконало розглянута у другій частині посібника у

розділі 3. Це означає, що сигнали зворотного зв'язку повинні формуватися з другого та з п'ятого розряду регістру, а надходять вони на вхід першого розряду. Відповідна структурна схема генератора послідовностей із перемішуванням бітів, основаного на алгоритмі скремблювання, наведена на рис. 1.43 [28, 30, 52, 53].

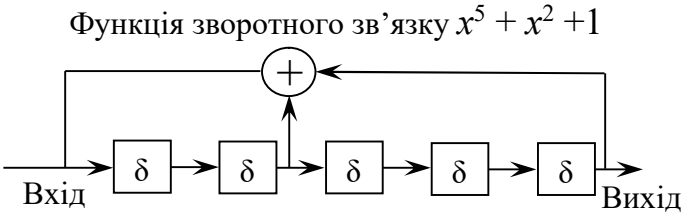


Рис. 1.43 Узагальнена схема генератора псевдовипадкових послідовностей із скремблюванням на основі утворювального поліному $x^5 + x^2 + 1$

У системах зв'язку, зокрема у системі CDMA (абревіатура англійського терміна Code Division Multiple Access, множинний доступ із кодовим розділенням) для реалізації алгоритмів скремблювання довгих кодових послідовностей використовують також утворювальні поліноми більш високих порядків, зокрема [28]:

$$f(x) = x^{18} + x^7 + 1; \qquad f(x) = x^{18} + x^{10} + x^7 + x^5 + 1;$$

$$f(x) = x^{25} + x^3 + 1; \qquad f(x) = x^{25} + x^3 + x^2 + x + 1.$$

Із розглянутих вище теоретичних міркувань зрозуміло, що алгоритми скремблювання базуються на теорії поліноміальних твірних функцій, які були розглянуті у підрозділі 3.8 другої частини посібника. Алгебраїчна теорія поліноміальних твірних функцій також є підґрунтям для вивчення особливостей формування завадостійких циклічних кодів, які розглядатимуться у підрозділі 3.4.

У реальних системах зв'язку із використанням алгоритмів скремблювання регістр зсуву із зворотним зв'язком має бути встановлений як у передавальний, так і у приймальний пристрій [30, 33]. У цьому випадку необхідно забезпечувати ініціалізацію регістра зсуву таким чином, щоб послідовність біт, яка кодується, починалась з 1, а передавальний та

приймальний пристрій починали працювати синхронно. Для забезпечення самосинхронізації приймального та передавального регістрів необхідно використовувати трішки іншу схему їхнього підключення, яка показана на рис. 1.44 [52].

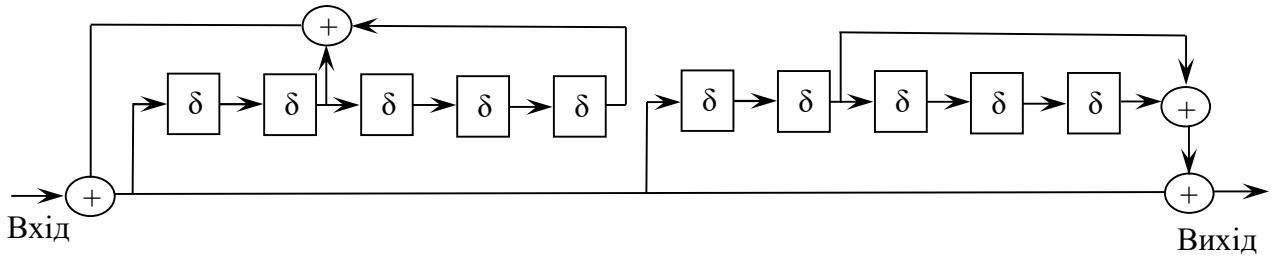


Рис. 1.44 Узагальнена схема генератора псевдовипадкових послідовностей із синхронізацією

Більш досконало сучасні стандарти систем зв'язку, зокрема стандарт CDMA, розглядатимуться у четвертій частині посібника.

У додатку Д наведена комп'ютерна програма scrambling, в якій реалізовані алгоритми скрамблювання та дескрамблювання бітових послідовностей з використанням співвідношень (1.29) – (1.31), написана мовою програмування системи MatLab. Всі логічні дії над двійковими послідовностями були реалізовані окремими функціями користувача з використанням засобів матричного програмування та арифметико-логічних функцій (1.9). Прикладами таких функцій є функції mod3 та mod32, програмні коди яких також наведені у додатку Е.

Зокрема, арифметико-логічна функція для здійснення операції «виключного або» над трьома елементами, записується наступним чином:

$$\begin{aligned}\mathrm{mod}3(x_1,x_2,x_3) &= (x_1=1)\cdot(x_2=1)\cdot(x_3=1)\cdot 0 + \\ &+ (x_1=0)\cdot(x_2=0)\cdot(x_3=0)\cdot 0 + (x_1=1)\cdot(x_2=0)\cdot(x_3=0)\cdot 1 + \\ &+ (x_1=0)\cdot(x_2=1)\cdot(x_3=0)\cdot 1 + (x_1=0)\cdot(x_2=0)\cdot(x_3=1)\cdot 1 + \\ &+ (x_1=1)\cdot(x_2=1)\cdot(x_3=0)\cdot 1 + (x_1=1)\cdot(x_2=0)\cdot(x_3=1)\cdot 1 + \\ &+ (x_1=0)\cdot(x_2=1)\cdot(x_3=1)\cdot 1 .\end{aligned} \qquad (1.32)$$

Тобто, програма scrambling, призначена для реалізації алгоритмів скрамблювання та дескрамблювання двійкових бітових послідовностей, написана з використанням засобів матричного та функціонального програмування системи науково-технічних розрахунків MatLab [13, 14].

Враховуючи співвідношення (1.32), рекурентні співвідношення (1.29), (1.30) для описаного вище алгоритму скрамблювання можна записати через арифметико-логічний вираз (1.9) наступним чином:

$$\begin{aligned}\mathrm{scr}(x_1,\ldots,x_n) &= (i=1)\cdot x_1 + (i=2)\cdot x_2 + (i=3)\cdot x_3 + \\ &+ (i=4)\cdot \mathrm{mod}\,2(x_i, y_{i-3}) + (i=5)\cdot \mathrm{mod}\,2(x_i, y_{i-3}) + \\ &+ (i>5)\cdot \mathrm{mod}\,3(x_i, y_{i-3}, y_{i-5});\ n>5,\ \ i=1{:}n,\end{aligned} \tag{1.33}$$

де x_k – елементи вхідного вектора, y_k – елементи вихідного вектора.

Відповідно, співвідношення для алгоритму дескрамблювання (1.31) можна переписати у наступному вигляді:

$$\begin{aligned}\mathrm{descr}(y_1,\ldots,y_n) &= (i=1)\cdot y_1 + (i=2)\cdot y_2 + (i=3)\cdot y_3 + \\ &+ (i=4)\cdot \mathrm{mod}\,3(y_i, y_{i-3}, y_{i-3}) + \\ &+ (i=5)\cdot \mathrm{mod}\,3(y_i, y_{i-3}, y_{i-3}) + \\ &+ (i>5)\cdot \mathrm{mod}\,32(y_i, y_{i-3}, y_{i-5});\ n>5,\ \ i=1{:}n.\end{aligned} \tag{1.34}$$

У програмі scrambling реалізований як алгоритм скрамблювання, заданий арифметико-логічним співвідношенням (1.33), так і алгоритм дескрамблювання, заданий співвідношенням (1.34). Результати тестування цієї програми також наведені у додатку Е. Цікавим те, що принцип модульного програмування системи MatLab дозволяє використовувати універсальну процедуру recvectbin для реалізації різних рекурентних алгоритмів, пов’язаних із обробками двійкових послідовностей. Змінюється лише форма запису рядків із арифметико-логічними виразами відповідно із заданими рекурентними алгоритмами. Будучи написаною для реалізації алгоритмів різницевого кодування, тепер ця функція також використана для реалізації алгоритмів скрамблювання. Алгоритмічні та функціональні особливості написання функції recvectbin, які враховують принципи

логічного та модульного програмування системи науково-технічних розрахунків MatLab [13, 14], були досконало описані у підрозділі 1.2.2.

Спектри сигналів для розглянутої кодової послідовності 110110000001 з використанням алгоритму скрамблювання та без його використання за умови подання сигналу у коді AMI наведені на рис. 1.45. Для побудови спектрів була використана програма, наведена у додатку В.

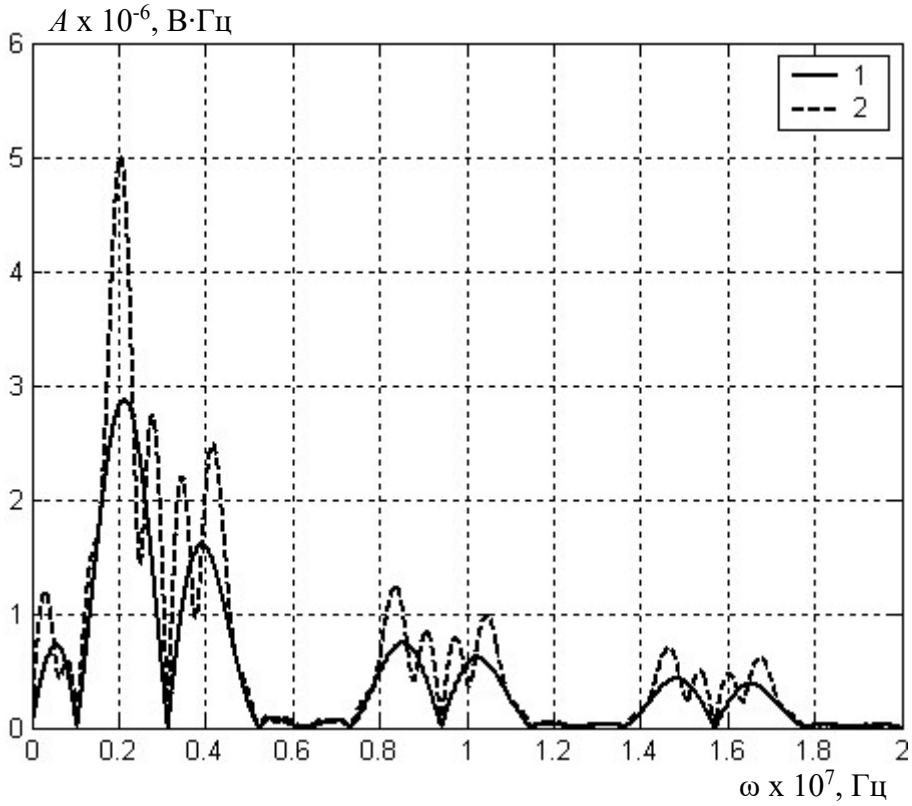


Рис. 1.45 Спектр сигналу AMI для кодової послідовності 110110000001 з використанням (1) та без використання алгоритму скрамблювання

На рис. 1.45 видно, що більша кількість одиниць у кодовій послідовності після використання алгоритму скрамблювання дозволяє збільшити амплітуду сигналу на робочих частотах на уникнути помилок розпізнавання сигналу на приймальній стороні. Частотний діапазон сигналу можна обмежити частотою $1{,}2 \cdot \omega_c$, проте, як видно з рис. 1.45, приблизно 90% енергії сигналу зосереджено в діапазоні $0{,}3 \cdot \omega_c$, а на нульовій частоті енергія сигналу дорівнює нулю.

Існує ще одна особливість використання алгоритмів скремблювання у безпроводових системах зв'язку. Тут середовище передавання інформації є єдиним для всіх передавальних пристроїв. Тому змішана кодова послідовність у цьому випадку реалізується із різними утворювальними поліномами для кожного терміналу, і це дозволяє уникнути помилок, які можуть виникати через конфлікт передавальних пристроїв [28 – 30, 33]. Такий спосіб кодування інформації відповідає кодовому розділенню сигналів, який був описаний у розділі 7 першої частини посібника [1]. Скремблювання у системах зв'язку використовують також для шифрування інформації, оскільки здійснити зворотний процес дескремблювання можливо лише за умови, що на приймальній стороні відомий утворювальний поліном. Алгоритми шифрування інформації базуються на теорії чисел, яка розглядалася у другій частині загального посібника у першому розділі [48], і будуть описані у розділі 4 цієї частини посібника. Тобто, не зважаючи на те, що алгоритми скремблювання відповідають засобам натурального кодування, вони забезпечують виявлення помилок, що притаманно завадостійким кодам. Це також зайвий раз підкреслює умовність класифікації кодів, наведеної у підрозділі 1.1.

1.4 Натуральні ітеративні блокові коди

1.4.1 *M*-послідовності та їхні властивості

Перед вивченням цього підрозділу необхідно повторити розділ 2 першої частини посібника

У сучасній теорії кодування, особливо для формування цифрових кодів бездротових мереж зв'язку, широко використовуються послідовності максимальної довжини, або *m*-послідовності. Ці послідовності основані на теорії скінченних полів Галуа та поліноміальних операцій над ними, які розглядалися у другому та третьому розділах другої частини посібника [48]. Головною перевагою ітеративних кодів, побудованих на основі *m*-послідовностей, є низьке значення взаємних кореляційних функцій та високе

значення автокореляційних функцій. Способи обчислення кореляційних функцій сигналів розглядалися у третьому розділі першої частини посібника [1]. Цікавою властивістю *m*-послідовностей є те, що, хоча за своєю сутністю вони є детермінованими, їхні часові реалізації більше відповідають випадковим, ніж детермінованим процесам [26, 28]. Надамо узагальнене визначення *m*-послідовності [28, 66].

Визначення 1.2. *M*-послідовністю, або послідовністю максимальної довжини (англійський термін – **M**aximum **L**ength **S**equence, **MLS**) називається псевдовипадкова двійкова послідовність, яка породжена регістром зсуву із лінійним зворотним зв'язком та має максимальний період.

Головним способом формування *m*-послідовностей є наступне рекурентне співвідношення [28, 29, 66]:

$$d_i = -f_{n-1}d_{i-1} - f_{n-2}d_{i-2} - f_0 d_{i-n}, \qquad (1.35)$$

де $f_0, f_1, \ldots, f_{n-1}$ – фіксовані константи, які належать до поля Галуа $GF(p)$, n – степінь рекурсії.

Розглянемо головні властивості *m*-послідовностей [28, 60, 66].

Властивість 1.1. Властивість урівноваження. На одному періоді *m*-послідовності із кількістю елементів p будь-який не рівний нулю елемент зустрічається p^{n-1} разів, а нульовий елемент – $p^{n-1} - 1$ разів.

Властивість 1.2. Будь-які дві *m*-послідовності, які генеруються рекурентним співвідношенням (1.35) із однаковими коефіцієнтами f_n, відрізняються одна від одної лише циклічним зсувом.

Властивість 1.3. Властивість зсуву та віднімання. Якщо із *m*-послідовності d_i, заданої співвідношенням (1.35), відняти поелементно за модулем p послідовність d'_i, сформовану як циклічний зсув послідовності d_i, тоді елементи послідовності d'_i також визначаються співвідношенням (1.35).

Наведемо приклади формування *m*-послідовності [28].

Приклад 1.6. Побудувати *m*-послідовність із степінню рекурсії $n = 3$ над полем Галуа $GF(2)$ з коефіцієнтами рекурсії $f_2 = 0, f_1 = 1, f_0 = 1$. Початкові елементи послідовності: $d_0 = 1; d_1 = 0; d_2 = 0$.

Використовуючи співвідношення (1.35), маємо:

$d_3 = d_1 + d_0 = 1; d_4 = d_2 + d_1 = 0; d_5 = d_3 + d_2 = 1; d_6 = d_4 + d_3 = 1;$

$d_7 = d_5 + d_4 = 1; d_8 = d_6 + d_5 = 0.$

Розглядаючи цю рекурентну послідовність далі, отримуємо періодичний числовий ряд із періодом $k = 7$:

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

Приклад 1.7. Побудувати *m*-послідовність із степінню рекурсії $n = 3$ над полем Галуа $GF(3)$ з коефіцієнтами рекурсії $f_2 = 0, f_1 = 2, f_0 = 1$. Початкові елементи послідовності: $d_0 = 1; d_1 = 0; d_2 = 0$.

У даному випадку співвідношення (1.35) приймає вигляд:

$$d_i = d_{i-2} + 2d_{i-3}, i \geq 3.$$

Тоді відповідна послідовність має період $k = 13$ та записується у вигляді:

1, 0, 0, 2, 0, 2, 1, 2, 2, 1, 0, 2, 2, 2, 0, 0, 1, 0, 1, 2, 1, 1, 2, 0, 1, 1...

Узагальнена структурна схема генератора лінійної *m*-послідовності, яка відповідає співвідношенню (1.35), наведена на рис. 1.41.

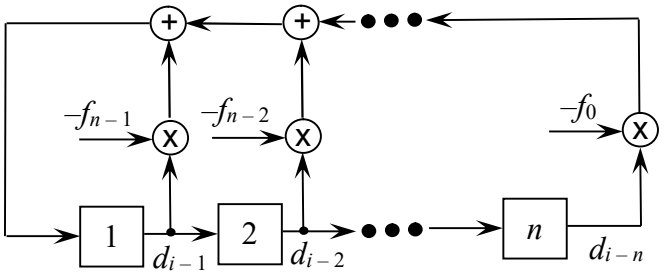


Рис. 1.46 Узагальнена структурна схема генератора рекурентної *m*-послідовності за співвідношенням (1.35)

Приклад 1.8. Побудувати схему генератора рекурентної послідовності для прикладу 1.6.

Згідно із рис. 1.46 схема генератора рекурентної m-послідовності для прикладу 1.6 буде мати вигляд, показаний на рис. 1.47.

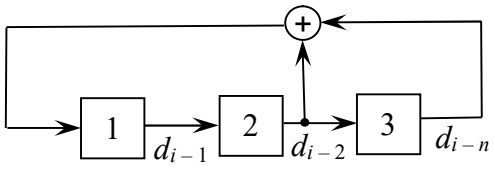


Рис. 1.47 Схема генератора рекурентної m-послідовності для прикладу 1.6

Розглянемо важливу теорему теорії кодування сигналів, яка встановлює зв'язок між співвідношенням (1.35) та періодичними m-послідовностями [28].

Теорема 1.1. Лінійне співвідношення (1.35) приводить до генерування m-послідовності у тому і лише у тому випадку, коли коефіцієнти f_i відповідають примітивному непривідному поліному.

Теорія примітивних привідних поліномів була розглянута у третьому розділі другої частини посібника [48]. Зрозуміло, що наведена теорема 1.1 тісно пов'язана із алгебраїчними операціями над групами Галуа та із циклотомічними класами, які розглядалися у другому розділі другої частини посібника [48].

1.4.2 Автокореляційні функції для m-послідовностей

Перед вивченням цього підрозділу необхідно повторити розділи 3 та 5 першої частини посібника та підрозділ 6.5 другої частини посібника

Розглянуті у попередньому підрозділі теоретичні положення дозволяють перейти до визначення автокореляційної функції m-послідовності [28, 66]. Такий аналіз значень двійкових сигналів, заданих через m-послідовності, має вельми важливе значення для формування у системах зв'язку сигнальних конструкцій із мінімальними похибками виявлення та розпізнавання сигналу. Відповідний математичний апарат розглядався у п'ятому розділі першої частини посібника [1] та у підрозділі

6.5 другої частини посібника [49].

Із властивості 1.1 безпосередньо випливає, що довжина *m*-послідовності $\{d_i\}$ із степінню рекурсії n становить $L = 2^n - 1$. Тоді таку послідовність можна записати у вигляді [28, 66]:

$$a_i = (-1)^{d_i} = \begin{cases} +1, & d_i = 0; \\ -1, & d_i = 1. \end{cases}$$

Отримана послідовність $\{a_i\}$ складається із символів ± 1. Крім цього, вона має період $N = L = 2^n - 1$ і є гомоморфним відображенням послідовності $\{d_i\}$ [28]. Автокореляційну функцію для послідовності $\{a_i\}$, згідно із теоретичними відомостями, наведеними у третьому розділі першої частині посібника, відповідно до співвідношення (3.151) [1], можна записати наступним чином [28, 60, 66]:

$$R_p(m) = \sum_{i=0}^{N-1} a_i a_{i-m} = \sum_{i=0}^{N-1} (-1)^{d_i + d_{i-m}} . \qquad (1.36)$$

Записане співвідношення (1.36) можна спростити, використовуючи розглянуту вище властивість 1.3, під час виконання цих алгебраїчних перетворень сумування коефіцієнтів $d_i + d_{i-m}$ необхідно здійснювати за модулем 2. На основі властивості 1.3 у теорії кодування доведена теорема про те, що елементи нової послідовності $\{d'_i\} = \{d_i + d_{i-m}\}$ із періодом L можуть приймати два значення: $d'_i = \{0,1\}$ за умови $(m \neq 0 \bmod N)$ або $d'_i = \{0,0\}$ у разі $(m = 0 \bmod N)$ [28, 66]. Тоді співвідношення (1.36) можна переписати у вигляді:

$$R_p(m) = L_0 - L_1 = \begin{cases} N, & m = 0 \bmod N; \\ -1, & m \neq 0 \bmod N, \end{cases} \qquad (1.37)$$

де L_0 – кількість нулів у послідовності $\{d_i\}$, L_1 – кількість одиниць у ній.

Розглянемо приклад обчислення автокореляційної функції *m*-послідовності з використанням співвідношення (1.37).

Приклад 1.9. Обчислити автокореляційну функцію для *m*-послідовності, заданої у прикладі 1.6, та побудувати графік цієї функції.

Згідно із співвідношенням (1.35), функція $\{a_i\}$ для *m*-послідовності

$$\{d_i\} = 1, 0, 0, 1, 0, 1, 1,...,$$

яка була розглянута у прикладі 1.6, записується у вигляді:

$$\{a_i\} = -1,+1+1,-1,+1,-1,-1,\ldots.$$

Значення $\{a_i\}$ для різних значень m, а також значення $R_p(m)$, які їм відповідають, зведені у таблиці 1.4.

Таблиця 1.4 – Значення автокореляційної функції *m*-послідовності для прикладу 1.9

m	a_0	a_1	a_2	a_3	a_4	a_5	a_6	$R_p(m)$
0	–	+	+	–	+	–	–	+7
1	–	–	+	+	–	+	–	–1
2	–	–	–	+	+	–	+	–1
3	–	–	–	–	+	+	–	–1

Блок-схема узгодженого фільтра для заданої *m*-послідовності наведена на рис. 1.48, а часові залежності сигналів та їхня кореляційна функція до *i* після блоку згладжувального фільтру показані на рис. 1.49 [28, 66]. Наведені часові діаграми відповідають точкам 1 – 9, які показані на рис. 1.48. Кореляційний фільтр, який наведений на рис. 1.48, побудований на основі теорії оптимальних фільтрів, яка була розглянута у підрозділі 6.5 другої частини посібника.

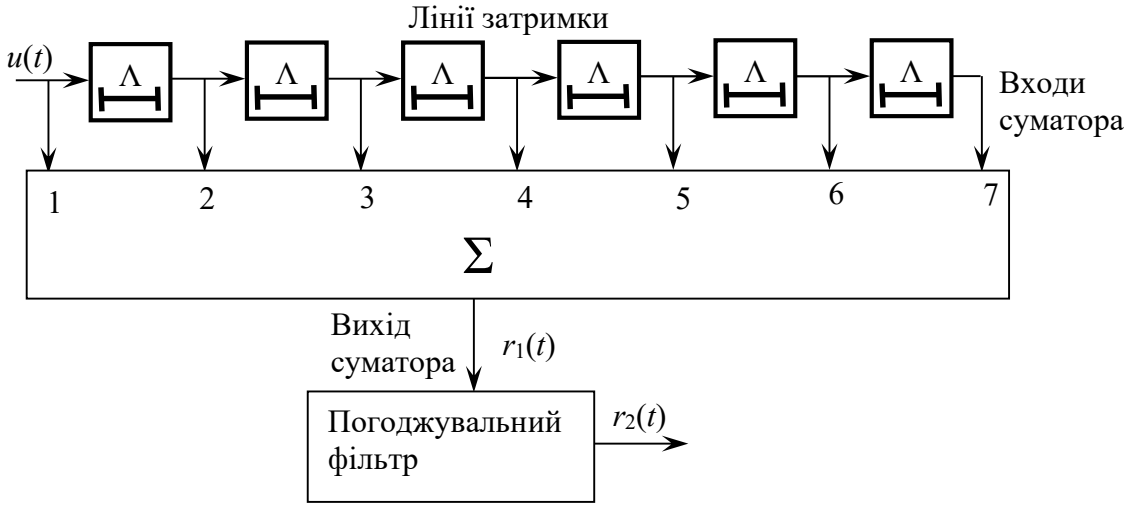


Рис. 1.48 Кореляційний фільтр для *m*-послідовності, яка розглядається у прикладі 1.9

Отримана кореляційна функція $r_2(t)$ є цікавою з точки зору забезпечення мінімальної похибки розпізнавання сигналів із *m*-послідовності. Дійсно, величина взаємної кореляції сигналів є досить високою і відповідає періоду послідовності $k = 7$, у той час як коефіцієнт кореляції для двох будь-яких різних сигналів становить –1, тобто ці сигнали є протилежними. Властивості *m*-послідовностей, які були розглянуті, базуються на теорії груп, розглянутій у розділі 2 другої частини посібника [48], та використовуються для побудови кодів Голда, які будуть розглянуті у наступному підрозділі.

Єдиним суттєвим недоліком розглянутого способу формування *m*-послідовностей є, те, що вони існують лише для обмеженої підмножини натуральних чисел $N = 2^n - 1 = 3, 7, 15, 31, 63, 127, 255\ldots$Тому іноді в інженерній практиці використовують також інші способи формування періодичних бінарних послідовностей, які базуються на теорії груп та кінцевих полів Галуа [48, 55 – 57], наприклад послідовності Лежандра [28, 66]. Послідовність Лежандра завжди може бути сформованою із елементів послідовності непарної довжини p, елементи цієї послідовності обчислюються наступним чином [28, 66]:

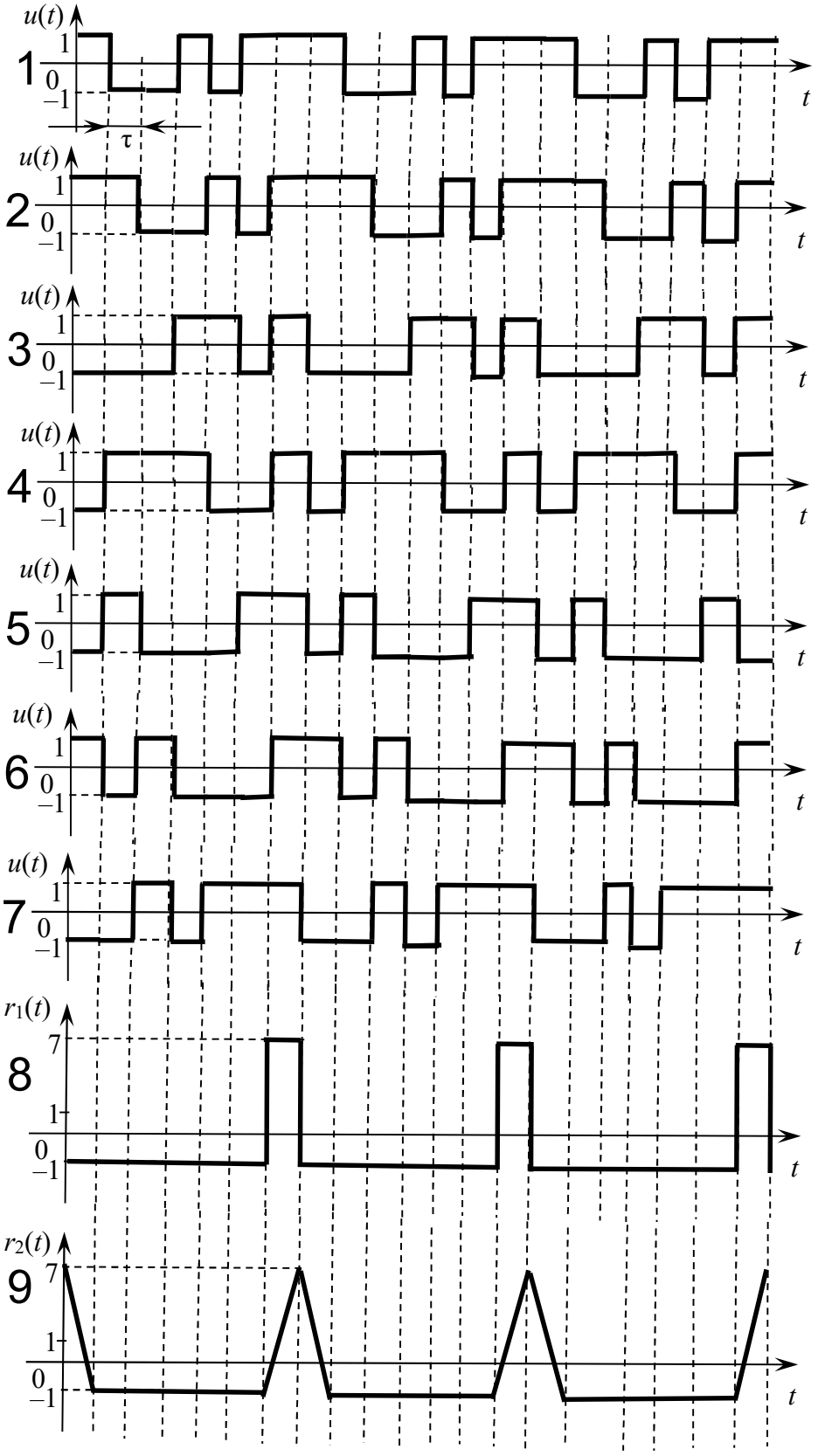


Рис. 1.49 Наочна ілюстрація способу обчислення кореляційної функції *m*-послідовності для прикладу 1.9

$$a_i = \begin{cases} +1, & i = 0 \bmod N; \\ \psi(i), & i \neq 0 \bmod N, \end{cases} \tag{1.38}$$

де $\psi(x)$ – характер ненульового елемента із поля Галуа $GF(p)$, який обчислюється через співвідношення [28, 66]:

$$\psi(x) = \begin{cases} 1, & \log_\alpha(x) = 0 \bmod N; \\ -1, & \log_\alpha(x) \neq 0 \bmod N; \end{cases} = (-1)^{\log_\alpha(x)}, \tag{1.39}$$

α – примітивний елемент поля Галуа $GF(p)$ [48].

Кореляційна функція для елементів послідовності Лежандра записується у вигляді [28, 66]:

$$R_p(m) = \begin{cases} N, & m = 0 \bmod N; \\ -1, & m \neq 0 \bmod N, \end{cases} N = 3 \bmod 4, \tag{1.40}$$

тобто, вона цілком аналогічна функції (1.37) для m-послідовності. Перевага послідовностей Лежандра полягає дише в тому, що потужність множини натуральних чисел, для яких вони існують, є значно більшою. Тому надалі будемо розглядати спосіб формування кодів Голда лише для m-послідовностей.

1.4.3 Границя Велча у широкосмужних системах зв'язку

Для безпроводових систем із великою кількістю користувачів, де приймальні пристрої всіх користувачів розділені територіально та налаштовані на приймання одного сигналу базової станції [1], можна обчислити можливу кількість користувачів K, виходячи із наступних міркувань. Насамперед будемо вважати, що розглядається широкосмужна система із фазовою модуляцією. Згідно із теорією статистичної радіотехніки, розглянутої у підрозділі 6.5 другої частини посібника [49], для цього випадку комплексну кореляційну функцію оптимального приймача можна записати у вигляді [28, 60, 66]:

$$\dot{z}_k = \int_{(i+1)T+\tau_k}^{iT+\tau_k} \dot{Y}(t)\dot{S}_k^*(t-\tau_k)\exp(-j\varphi_k)dt, \qquad (1.41)$$

де $\dot{Y}(t)$ – комплексна обвідна для спостереження зашумленого сигналу на вході приймача, $\dot{S}_k^*(t-\tau_k)$ – комплексна обвідна сигналу широкомовної системи, який спостерігається користувачем, T – час спостереження сигналу, τ_k – затримка сигналу, φ_k – зсув фази сигналу. Вважаючи, що $\tau_k = 0$ та $\varphi_k = 0$, перепишемо співвідношення (1.41) у спрощеному вигляді [28, 60, 66]:

$$\dot{z}_k = \int_0^T \dot{Y}(t)\dot{S}_k^*(t)dt. \qquad (1.42)$$

У теорії широкополосних систем зв'язку показано, що якщо сигнали всіх абонентів мережі є синхронізованими та кількість абонентів K не перевищує кількості сигналів N, тобто, за умови $K < N$, оптимальною є ортогональна система сигналів і завади від приймачів інших користувачів відсутні. Проте більш цікавою є інша ситуація, коли $K > N$ і взаємні завади між приймачами користувачів існують. Надамо відповідне визначення [28].

Визначення 1.3. Взаємні завади між приймачами користувачів у безпроводових системах зв'язку називаються завадами множинного доступу, або ЗМД (англійський термін – multiple access noise).

За умови наявності завад множинного доступу модель вхідного сигналу на вході приймача $\dot{Y}(t)$ можна записати у вигляді [28, 60, 66]:

$$\dot{Y}(t) = \dot{S}(t, \mathbf{b}') + \dot{N}(t) = \sum_{l=1}^{k} b_l' \dot{S}(t) + \dot{N}(t), \qquad (1.43)$$

де $\mathbf{b}'$ – вектор, який характеризує наперед невідому користувачу абетку символів повідомлень, $\dot{N}(t)$ – комплексна обвідна шуму. Слід відзначити, що модель сигналу (1.43) цілком відповідає моделі системи зв'язку, яка була наведена на рис. 2.41 першої частини посібника [1]. Взагалі прийнятий символ повідомлення b_l' може відрізнятися від переданого символу b_l, що у

загальному випадку свідчить про хибність приймання повідомлення. З урахуванням (1.43), перепишемо співвідношення (1.42) у наступному вигляді [28, 60, 66]:

$$\dot{z}(b_k) = 2b'_k E + 2E \sum_{\substack{l=1,\\ l\neq k}}^{K} b'_l \dot{\rho}_{l,k} + \int_0^T \dot{N}(t)\dot{S}^*_k(t)dt, \qquad (1.44)$$

де $\dot{\rho}_{l,k}$ – коефіцієнт кореляції комплексних обвідних сигналу для символів l та k, E – енергія сигналу на один символ повідомлення, що передається. Згідно із співвідношенням (3.184), яке було наведено у підрозділі 3.5 першої частини посібника [1], енергія сигналу розраховується наступним чином [28, 60, 66]:

$$E = \frac{1}{2}\int_0^T \left|\dot{S}_k(t)\right|^2 dt. \qquad (1.45)$$

Зрозуміло, що друга складова у співвідношенні (1.44) описує завади множинного доступу. З іншого боку, кожна із складових $b'_l \dot{\rho}_{l,k}$ являє собою внесок передавачів у сумарну заваду, і ці складові є випадковими, оскільки залежать від випадкових символів b'. Проте, у разі фазової модуляції сигналу, завжди правильною є тотожність $|b'|=1$. З урахуванням цих спрощень, обумовлених фізичними особливостями роботи систем зв'язку із фазовою модуляцією, співвідношення (1.44) можна переписати у спрощеній формі [28, 60, 66]:

$$P_{Ik} = 4E^2 \sum_{\substack{l=1,\\ l\neq k}}^{K} \left|\dot{\rho}_{l,k}\right|^2, \qquad (1.46)$$

де P_{Ik} – потужність завади множинного доступу для приймача із номером k. Відповідно, узагальнене значення завади множинного доступу для всієї системи P_I можна визначити як суму виразів (1.46) за усіма значеннями k [28, 60, 66]:

$$P_I = 4E^2 \sum_{k=1}^{K} \sum_{\substack{l=1, \\ l \neq k}}^{K} \left| \dot{\rho}_{l,k} \right|^2 . \tag{1.47}$$

Основною задачею проектування ефективних широкосмужних безпроводових систем зв'язку є мінімізація завади множинного доступу [28]. Відповідний параметр системи зв'язку називається повним квадратом кореляції. Надамо відповідне визначення.

Визначення 1.4. Повним квадратом кореляції (англійський термін – **T**otal **S**quared **C**orrelation, **TSC**) називається мінімальне значення завади множинного доступу, яке пропорційне сумі квадратів коефіцієнтів взаємної кореляції сигналів за всіма приймальними пристроями користувачів.

Згідно із співвідношенням (1.47), для повного квадрату кореляції, який у літературі зазвичай позначається символом *TSC*, можна записати [28, 60, 66]:

$$TSC = \left(\sum_{k=1}^{K} \sum_{l=1}^{K} \left| \dot{\rho}_{l,k} \right|^2 \right)_{\min} . \tag{1.48}$$

Перехід від співвідношення (1.47) до (1.48) є правильним з урахуванням того, що для сигнальних конструкцій із фазовою модуляцією модуль коефіцієнту автокореляції для всіх сигналів дорівнює 1, тобто [28, 60, 66]:

$$\forall \left(k \in N, k \le K \right) \left| \dot{\rho}_{k,k} \right| = 1 . \tag{1.49}$$

Зрозуміло, що співвідношення (1.49) записано з використанням теорії предикатів, яка розглядалася у підрозділі 7.1 другої частини посібника [50].

Цікавим є те, що для величини *TSC* у широкосмужних безпроводових системах зв'язку існує нижня границя. Вважаючи, що всі вектори кодових послідовностей

$$\mathbf{a}_k = \left(a_{k,0}, a_{k,1}, \ldots, a_{k,N-1} \right). \tag{1.50}$$

є нормованими, коефіцієнт взаємної кореляції сигналів складає [28, 60, 66]:

$$\dot{\rho}_{l,k} = \frac{\left(a_k a_l \right)}{N} = \frac{1}{N} \sum_{i=0}^{N-1} a_{k,i} a_{l,i}^* . \tag{1.51}$$

З урахуванням (1.49) – (1.51), можна переписати співвідношення (1.48) наступним чином [47]:

$$TSC = \frac{1}{N^2}\sum_{k=1}^{K}\sum_{l=1}^{K}\sum_{i=1}^{N-1}\sum_{j=1}^{N-1} a_{k,i}a_{l,i}^{*}a_{k,i}^{*}a_{l,i} = \frac{1}{N^2}\sum_{i=1}^{N-1}\sum_{j=1}^{N-1}\left|\sum_{k=1}^{K} a_{k,i}a_{k,j}^{*}\right|^2. \qquad (1.52)$$

Оскільки всі значення модулів сигналів за змінними i та j є додатними величинами, можна використати відому нерівність Коші-Буняковського [25, 51] і записати [28, 60, 66]:

$$TSC \geq \frac{1}{N^2}\sum_{i=1}^{N-1}\left(\sum_{k=1}^{K}\left|a_{k,i}\right|^2\right)^2. \qquad (1.53)$$

Отримане співвідношення (1.53), з урахуванням того, що для сигналів із фазовою модуляцією $\left|a_{k,i}\right| = 1$, можна переписати у спрощеному вигляді [28, 66]:

$$TSC \geq \frac{K^2}{N}, K > N,$$

або, остаточно [28, 66]:

$$TSC \geq \begin{cases} K, & K \leq N, \\ \dfrac{K^2}{N}, & K > N. \end{cases} \qquad (1.54)$$

Співвідношення (1.37) називається границею Велча для широкосмужних систем зв’язку [28]. Із наведених міркувань зрозуміло, що цей параметр характеризує найкращий можливий приймач за мінімумом завад множинного доступу. Однак оптимальність ансамблю сигнальних конструкцій, які відповідають границі Велча, не обмежується мінімумом взаємних шумів між приймачами. У теорії систем зв’язку показано, що сигнальні конструкції, які відповідають співвідношенню (1.54), також забезпечують максимальну пропускну здатність каналу зв’язку з гаусовським шумом [28, 66].

Оскільки у виразі (1.53) для *TSC* розглядається максимальна кількість різних сигналів у системі, можна сказати, що ця величина відповідає кількості сполучень із K по 2. Ця величина, згідно із співвідношенням

(1.212), наведеним у підрозділі 1.3 другої частини посібника [49], складає:

$$N_c = \overline{C}_K^2 = \frac{K!}{(K-2)!} = K(K-1). \tag{1.55}$$

Відповідно до (1.38), середній квадрат кореляції на одну пару сигналів складає:

$$\overline{\rho^2} = \frac{TSC - K}{K(K-1)}. \tag{1.56}$$

Тоді, враховуючи співвідношення (1.54), (1.56), для нижньої границі середнього квадрату кореляції можна записати:

$$\overline{\rho^2} \geq \begin{cases} 0, & K \leq N, \\ \dfrac{K-N}{N(K-1)}, & K > N. \end{cases} \tag{1.57}$$

Наведений вище спосіб обчислення нижньої границі повного квадрату кореляції (1.54) та середнього квадрату кореляції (1.57) надає можливість не лише проводити інженерні оцінки можливої кількості користувачів у безпроводовій мережі, але й формувати сигнальні конструкції, для яких найменше значення повного квадрату кореляції є близьким до граничної величини, яка визначається формулою (1.54). Зрозуміло, що співвідношення (1.53) є необхідною та достатньою умовою виконання нерівності (1.54). Із цього безпосередньо слідує, що, згідно із (1.57), (1.53), на границі Велча лежать такі послідовності, для яких виконуються умови:

$$\sum_{k=1}^{K} a_{k,i} a_{k,j}^{*} = 0, \; i \neq j. \tag{1.58}$$

Запишемо коди сигналів у вигляді породжувальної матриці, спосіб формування яких був розглянутий у розділі 5 другої частини посібника [48]:

$$\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_K] = \begin{bmatrix} a_{1,0} & a_{2,0} & \ldots & a_{K,0} \\ a_{1,1} & a_{2,1} & \ldots & a_{K,1} \\ \ldots & \ldots & \ldots & \ldots \\ a_{1,N-1} & a_{2,N-1} & \ldots & a_{K,N-1} \end{bmatrix}. \tag{1.59}$$

Тоді для матриці **A**, заданої співвідношенням (1.58), виконання умови (1.58) свідчить про ортогональність її рядків. Тобто, для формування

сигнальної конструкції, яка відповідає границі Велча (1.54), необхідно та достатньо сформувати ортогональну матрицю із розмірністю *K*x*N*. За умови *K*>*N* ця задача не є складною, її можна розв'язати одним із методів, які розглядалися в другій частині посібника, наприклад, методом невизначених коефіцієнтів або через обчислення власних чисел матриці [73, 74]. На практиці для побудови матриць сигнальних конструкцій також широко використовують матриці Адамара, які розглядалися у підрозділах 4.5 та 5.5 другої частини посібника [48].

У широкосмужних безпроводових системах зв'язку із кодовими конструкціями, для яких значення *TSC* є близьким до границі Велча, співвідношення потужності сигналу до потужності завади q_I^2 можна оцінити наступним чином [28, 29]:

$$q_I^2 = \frac{K}{TSC - K} = \frac{N}{K - N}. \tag{1.60}$$

Розглянемо приклад формування сигнальної конструкції, кодові послідовності якої наближаються до границі Велча (1.54) [28].

Приклад 1.10. Побудувати бінарний ансамбль довжиною N = 14, якщо кількість користувачів складає K = 16, за умови, що кодові послідовності мають лежати на границі Велча.

Для розв'язування цієї задачі скористаємося матрицею Адамара $\mathbf{H}_4$, яку запишемо у наступному вигляді:

$$\mathbf{H}_4 = \begin{bmatrix} + & + & + & + \\ - & - & + & + \\ + & - & + & - \\ + & - & - & + \end{bmatrix}.$$

Скориставшись відомим способом розмноження матриці Адамара, який був описаний у підрозділі 5.5.5 другої частини посібника, на основі $\mathbf{H}_4$ сформуємо матрицю $\mathbf{H}_{16}$:

$$\mathbf{H}_{16} = \begin{bmatrix} \mathbf{H}_4 & \mathbf{H}_4 & \mathbf{H}_4 & \mathbf{H}_4 \\ -\mathbf{H}_4 & -\mathbf{H}_4 & \mathbf{H}_4 & \mathbf{H}_4 \\ \mathbf{H}_4 & -\mathbf{H}_4 & \mathbf{H}_4 & -\mathbf{H}_4 \\ \mathbf{H}_4 & -\mathbf{H}_4 & -\mathbf{H}_4 & \mathbf{H}_4 \end{bmatrix} =$$

$$= \frac{1}{4} \begin{bmatrix} + & + & + & + & + & + & + & + & + & + & + & + & + & + & + & + \\ - & - & + & + & - & - & + & + & - & - & + & + & - & - & + & + \\ + & - & + & - & + & - & + & - & + & - & + & - & + & - & + & - \\ + & - & - & + & + & - & - & + & + & - & - & + & + & - & - & + \\ - & - & - & - & - & - & - & - & + & + & + & + & + & + & + & + \\ + & + & - & - & + & + & - & - & + & + & - & - & + & + & - & - \\ - & + & - & + & - & + & - & + & - & + & - & + & - & + & - & + \\ - & + & + & - & - & + & + & - & - & + & + & - & - & + & + & - \\ + & + & + & + & - & - & - & - & + & + & + & + & - & - & - & - \\ - & - & + & + & + & + & - & - & - & - & + & + & + & + & - & - \\ + & - & + & - & - & + & - & + & + & - & + & - & - & + & - & + \\ + & - & - & + & - & + & + & - & + & - & - & + & - & + & + & - \\ + & + & + & + & - & - & - & - & - & - & - & - & + & + & + & + \\ - & - & + & + & + & + & - & - & + & + & - & - & - & - & + & + \\ + & - & + & - & - & + & - & + & - & + & - & + & + & - & + & - \\ + & - & - & + & - & + & + & - & - & + & + & - & + & - & - & + \end{bmatrix}. \qquad (1.61)$$

Для того, щоб створити кодову конструкцію за заданою умовою, достатньо довільно обрати будь-які 14 із сформованих шістнадцяти рядків матриці Адамара (1.61). Слід відзначити, що, як було показано у підрозділі 5.5.5 другої частини посібника, спосіб кодування сигналів з використанням матриць Адамара також відповідає формуванню системи ортогональних функцій Уолша.

Приклад 1.11. Для двійкового коду, побудованого у прикладі 1.10, оцінити повний квадрат кореляції та співвідношення потужності сигналу до потужності шуму.

Згідно із співвідношенням (1.54) повний квадрат кореляції складає:

$$TSC = \frac{K^2}{N} = \frac{256}{14} \approx 18{,}3 .$$

Відповідно, для величини q_I^2, згідно із співвідношенням (1.60 маємо:

$$q_I^2 = \frac{N}{K-N} = \frac{14}{16-14} = 7 .$$

1.4.4 Коди Голда та спосіб їхнього формування

Логічним наслідком розглянутої вище теорії *m*-послідовностей та границі Велча є коди Голда [28, 66]. Коди Голда, згідно із класифікацією, наведеною на рис 1.2, можна розглядати як ітеративні групові коди. Сьогодні ці коди широко використовуються у широкомовних безпроводових системах передавання інформації, оскільки вони дозволяють отримувати мінімальну похибку розпізнавання сигналів за умови достатньо високого рівня шумів, що забезпечується через мінімальну степінь взаємної кореляції сигналів та високий коефіцієнт автокореляції [28, 66].

Розглянемо відповідні визначення та властивості двійкових *m*-послідовностей, на яких ґрунтується формування кодів Голда [28, 66].

Визначення 1.5. Децимацією *m*-послідовності називається вибір кожного елементу послідовності із номером *d* та записування всіх вибраних елементів послідовності один за одним.

Визначення 1.6. Параметр *d*, який характеризує номер елемента, в що повторюється процесі децимації, називається періодом децимації.

Властивість 1.4. Якщо для *m*-послідовності $\{u_i\}$ з періодом $L = 2^n - 1$ проводиться децимація з періодом *d*, і відомо, що числа *d* та *L* є взаємнопростими, тоді отримана послідовність $\{v_i\}$ також завжди буде періодичною із періодом *L*.

Властивість 1.5. Припустимо, що степінь рекурсії *n* бінарної *m*-послідовності є непарним числом, а період децимації *d* і степінь рекурсії *n* є взаємнопростими числами. У разі виконання цих умов завжди виконуються наступні твердження.

1. Період децимації *d* і період послідовності *L* є взаємнопростими числами.

2. Результат децимації $\{v_i\}$ є *m*-послідовність із періодом *L*.

3. Взаємна кореляційна функція послідовностей $\{u_i\}$ та $\{v_i\}$ може приймати лише три можливих значення, які визначаються співвідношенням:

$$R_{p,uv}(m) \in \left\{\sqrt{2(L+1)}-1,-\sqrt{2(L+1)}-1,-1\right\} = \left\{2^{\frac{n+1}{2}}-1,-2^{\frac{n+1}{2}}-1,-1\right\}. \quad (1.62)$$

Властивість 1.6. Припустимо, що степінь рекурсії n бінарної m-послідовності є парним числом, яке не кратне чотирьом, а період децимації d також є парним числом і взаємнопростим із $n/2$. У разі виконання цих умов завжди виконуються наступні твердження.

1. Період децимації d і період послідовності L є взаємнопростими числами.

2. Результат децимації $\{v_i\}$ є m-послідовність із періодом L.

3. Взаємна кореляційна функція послідовностей $\{u_i\}$ та $\{v_i\}$ може приймати лише три можливих значення, які визначаються співвідношенням:

$$R_{p,uv}(m) \in \left\{2\sqrt{(L+1)}-1,-2\sqrt{(L+1)}-1,-1\right\} = \left\{2^{\frac{n+2}{2}}-1,-2^{\frac{n+2}{2}}-1,-1\right\}. \quad (1.63)$$

Будемо формувати код Голда наступним чином. Припустимо, що відомі послідовності $\{u_i\}$ та $\{v_i\}$. Тоді, згідно із властивостями двійкових m-послідовностей 1.4 – 1.6, елементи сигнальної конструкції $a_{k,i}$ можна визначити наступним чином:

$$a_{k,i} = u_i v_{i-k};\ k = 1,2,\ldots,L;\ i = \ldots -1,0,1,\ \ldots;\ a_{L+1,i} = u_i;\ a_{L+2,i} = v_i. \quad (1.64)$$

Узагальнена структурна схема кодувального пристрою, який формує код Голда, побудована на основі регістрів зсуву, суматора за модулем 2, лінії затримки та пристроїв відображення на множину $\{\pm 1\}$, наведена на рис. 1.45 [28].

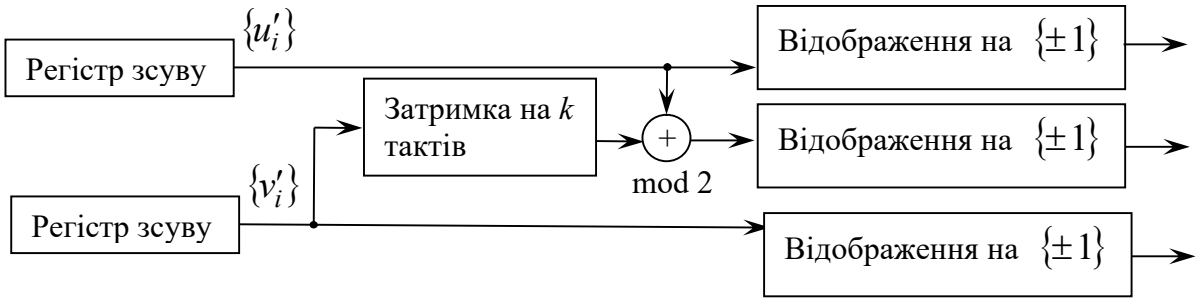


Рис. 1.50 Структурна схема кодувального пристрою для формування коду Голда

Слід відзначити, що спосіб формування кодів Голда є дуже схожим на алгоритм диференціального кодування, який задається співвідношеннями (1.13) та (1.14) та був описаний у підрозділі 1.2.2.

Взаємну кореляційну функцію для коду Голда можна оцінити за співвідношенням [28]:

$$R_{p,uv}(m) = \sum_{i=0}^{L-1} u_i u_{i-1} v_{i-k} \,. \tag{1.65}$$

Максимальне значення кореляційної функції (1.48) відповідає оцінкам (1.62), (1.63). Враховуючи нормування за змінною L, остаточно маємо [28]:

$$\rho^2_{\max} = \begin{cases} \dfrac{\left(\sqrt{2(L+1)}+1\right)^2}{L^2}, & n \neq 0 \bmod 2; \\ \dfrac{\left(2\sqrt{(L+1)}+1\right)^2}{L^2}, & n = 2 \bmod 4. \end{cases} = \begin{cases} \dfrac{2}{L}, & n \neq 0 \bmod 2; \\ \dfrac{4}{L}, & n = 2 \bmod 4. \end{cases} \tag{1.66}$$

Розглянемо приклад формування коду Голда [28].

Приклад 1.12. Побудувати послідовності коду Голда із довжиною $L = 7$.

У даному випадку код Голда можна будувати на основі m-послідовності, розглянутої у прикладі 1.9, тобто будемо вважати, що $\{u'_i\} = \{1,0,0,1,0,1,1\}$, а період децимації d складає 3. В результаті отримуємо послідовність $\{v'_i\} = \{1,1,1,0,1,0,0\}$. Скористаємось співвідношеннями (1.64). Сумування за модулем 2 послідовностей $\{u'_i\}$ та $\{v'_i\}$ дає результат $\{0,1,1,1,1,1,1\}$, після відображення якого на абетку $\{\pm 1\}$ отримуємо код першого символу: $\{+,-,-,-,-,-,-\}$. Аналогічно, після зсуву символів послідовності $\{v'_i\}$ на одну позицію праворуч та сумування їх за модулем 2 із символами послідовності $\{u'_i\}$, отримуємо результат $\{1,1,1,0,0,0,1\}$, або, у коді $\{\pm 1\}$, $\{-,-,-,+,+,+,-\}$. Таким же чином отримуються інші п'ять послідовностей коду Голда.

Коди Голда відрізняються досить високою завадостійкістю, і тому вони

часто використовуються для кодування сигналів у сучасних системах безпроводового зв'язку, зокрема у мобільній телефонії та безпроводових комп'ютерних мережах, у космічному радіозв'язку тощо. Відповідні стандарти систем зв'язку розглядатимуться у четвертій частині посібника.

Приклад 1.13. Для коду Голда, який був сформований у прикладі 1.12, обчислити максимальне значення функції кореляції.

Згідно із співвідношенням (1.66) маємо:

$$\rho_{\max}^2 = \frac{2}{L} = \frac{2}{7} = 0{,}286\,.$$

Недоліком кодів Голда є те, що відповідні кодові послідовності існують лише у разі довжини базової послідовності $L = 2^n - 1$, тобто, для обмеженої множини натуральних чисел. Тому, поряд із кодами Голда, у сучасних системах зв'язку також використовують інші натуральні блокові коди, зокрема множини Касамі та ансамблі Камалітдінова. Способи формування таких послідовностей цілком подібні на спосіб формування кодів Голда та описані у підручнику [28].

1.5 Ефективні коди

1.5.1 Поняття про ефективні коди та їхні параметри

Перед вивченням цього підрозділу необхідно повторити розділи 1 та 4 першої частини посібника та розділ 6 другої частини посібника

У тих випадках, коли відсутні статистичні залежності між символами кодової комбінації, у системах зв'язку широко використовуються способи ефективного кодування (англійський термін effective coding) [2 – 5, 8, 33]. Головною задачею ефективного кодування є забезпечення мінімального завантаження каналу зв'язку. Сутність принципу ефективного кодування полягає в тому, що із символів абетки, імовірність появи яких є високою, формуються короткі кодові групи, а із символів, які зустрічаються рідко – довгі групи. За рахунок цього у разі передавання довгих повідомлень у багатьох випадках вдається у значній мірі збільшити швидкість їх

передавання, і, відповідно, зменшити час використання каналу зв’язку. Серед ефективних кодів найбільш відомими та розповсюдженими є коди Шеннона – Фано та коди Хаффмена [2 – 5, 8, 33].

Згідно із наведеним правилом формування ефективного коду, цей спосіб кодування потребує найбільш економного подання дискретних елементів коду x_i. Тобто, формування ефективного коду – це складна оптимізаційна задача, розв’язування якої потребує мінімізації кількості розрядів для найбільш ймовірних символів повідомлення, тобто, для символів, які найчастіше зустрічаються. Таким чином, розрядність кодової комбінації, яка відповідає символу x_i, зворотно пропорційна імовірності появи цього символу $p(x_i)$.

Згідно із наведеними міркуваннями зрозуміло, що, на відміну від натуральних двійкових кодів, розглянутих у підрозділі 1.3, розрядність символів ефективного коду m є змінною величиною. Саме з цієї причини можна ввести такий параметр ефективного коду, як середня розрядність символу m_i на повідомлення із кількістю символів N, і ця величина буде меншою, ніж для натурального кодування. З точки зору синхронізації систем зв’язку із ефективним кодуванням суттєвим є те, що цей спосіб кодування не потребує розділювальних знаків між символами, незважаючи на те, що довжина їх є різною. Це безпосереднє пов’язано із способом побудови абетки ефективних кодів. Він полягає в тому, що початкова послідовність знаків, яка відповідає будь якому символу, не може бути початком іншого символу. Тобто, згідно із класифікацією граматик, наведеною у підрозділі 7.3.1, ефективні коди є префіксними. Відповідні способи побудови кодів Шеннона – Фано та кодів Хаффмена будуть розглянуті у підрозділах 1.5.4 та 1.5.5.

Існує декілька способів формування ефективних кодів. Основна різниця між ними в тому, що у деяких випадках враховується лише імовірність появи одного символу $p(x_i)$, а в інших випадках розглядається також умовні імовірності появи кодових послідовностей та кореляція між

окремими символами [9, 25, 27, 49].

Згідно із наведеними вище міркуваннями розглянемо простий приклад побудови ефективного коду [2, 5, 8].

Приклад 1.14. Побудувати ефективний код для абетки з чотирьох символів x_1, x_2, x_3, x_4, імовірності появи яких становлять: $p(x_1) = 0{,}5$; $p(x_2) = 0{,}25$; $p(x_3) = p(x_4) = 0{,}125$.

Будемо розв'язувати поставлену задачу ефективного кодування наступним чином. Оскільки, згідно з умовою задачі, найбільш імовірною є поява символу x_1, саме цьому символу має відповідати найкоротша кодова послідовність із одного елемента. Для двійкового ефективного коду таким елементом може бути 0 або 1. Відповідно до цього припустимо, що елементу x_1 відповідає кодова послідовність $k_1 = 1$. Тоді для забезпечення виконання правила про те, що початок кодової послідовності для всіх символів має бути унікальним, коди символів x_2, x_3 та x_4 повинні починатися з нуля і кількість елементів в цих кодах має бути більшою, ніж один. За цих умов легко побудувати код символу x_2, він буде $k_2 = 01$. Інших коректних комбінацій із двох елементів двійкового коду не існує. Тому, згідно із правилами формування ефективного коду, коди символів x_3 та x_4, імовірності появи яких є найменшими, будуть складатися із трьох елементів та починатися з двох нулів. Наприклад, можна прийняти, що $k_3 = 000$, а $k_4 = 001$. Тоді остаточно маємо: $k_1 = 1$, $k_2 = 01$, $k_3 = 000$, $k_4 = 001$.

Наведений приклад наочно ілюструє описані вище принципи побудови ефективного коду. Теоретичні підґрунтя, на основі яких формуються ефективні коди, розглядатимуться у наступному підрозділі.

1.5.2 Принципи кодування для каналів зв'язку без завад та теорема Шеннона

Можливості ефективного кодування та всі способи формування ефективних кодів базуються на теоремі Шеннона про кодування для каналів без завад, яка у загальному вигляді формулюється наступним чином.

Теорема 1.2. За умови будь-якої продуктивності джерела повідомлень, яка є меншою за пропускну здатність каналу зв'язку, існує спосіб кодування, який дозволяє передавати через такий канал всі повідомлення, які надходять від джерела, без спотворень. У випадку, коли продуктивність джерела повідомлень перевищує пропускну здатність каналу зв'язку, не існує такого способу кодування, який забезпечував би передавання повідомлень без спотворень без необмежного їхнього накопичування.

Тобто, з математичної точки зору необхідна та достатня умова можливості формування ефективного коду може бути записана наступним чином [2 – 5, 8]:

$$\bar{I}(Z) = C_{\text{д}} - \varepsilon, \tag{1.67}$$

де $\bar{I}(Z)$ – продуктивність джерела повідомлень, $C_{\text{д}}$ – пропускна здатність каналу зв'язку, ε – додатна нескінченно мала величина ($\varepsilon > 0$). З іншого боку, за умови

$$\bar{I}(Z) > C_{\text{д}} \tag{1.68}$$

не існує такого способу кодування, який дозволив би передавати інформацію без спотворень.

Існує повний точний доказ теореми Шеннона 1.2, проте від дуже складний [2, 5, 7]. Тому обмежимося наочними логічними міркуваннями, які дають можливість впевнитися у тому, що теорема 1.2 дійсно є правильною.

Теоретичним підґрунтям для доказу теореми 1.2 є ідея про можливість підвищення швидкості передавання інформації за умови, що символи абетки кодуються не окремими знаками, а їх послідовностями такої довжини, для якої є правильною теорема про їх асимптотичну рівноймовірність. В цьому випадку розглядають типові послідовності символів, які, за умови нескінченно великих повідомлень, тобто, коли кількість символів в повідомлені N прямує до нескінченності ($N \to \infty$), є рівноймовірними. Якщо кількість знаків у послідовності, яка розглядається, дорівнює N, а ентропія джерела повідомлень становить $H(Z)$, тоді кількість типових послідовностей

символів абетки, імовірність появи яких є однаковою, становить

$$n_{\text{т}} \approx 2^{H(Z)\cdot N}. \tag{1.69}$$

Враховуючи, що для багатознакових послідовностей

$$N = \frac{T}{\tau}, \tag{1.70}$$

де T – тривалість послідовності, яка кодується, τ – тривалість одного знака, а також зв'язок між ентропією та інформацією, який для багатознакових послідовностей записується у вигляді [2 – 5, 8]:

$$\bar{I}(Z) = \frac{H(Z)}{\tau}, \tag{1.71}$$

із співвідношень (1.68 – 1.71) маємо [2 – 5, 8]:

$$n_{\text{т}} \approx 2^{\bar{I}(Z)\cdot T}. \tag{1.72}$$

Тепер кожній типовій послідовності слід поставити у відповідність послідовність символів однакової тривалості *T*, оскільки за введеними припущеннями імовірність появи типових послідовностей також є однаковою. Тоді однаковою буде і кількість елементів у типових послідовностях. У разі, якщо швидкість передавання елементу становить V_T, кількість символів у кодовій комбінації складає TV_T, що дозволяє створити відповідну кількість кодових комбінацій, яка за умови кількості символів абетки *m*, розраховується наступним чином [2 – 5, 8]:

$$n_k = m^{IV_{\text{т}}} = 2^{\left(TV_{\text{т}} \log_2(m)\right)}. \tag{1.73}$$

Враховуючи, що

$$C_{\text{д}} = V_{\text{т}} \log_2(m), \tag{1.74}$$

із (1.71), (1.72) можна записати:

$$n_k = 2^{TC_{\text{д}}}, \tag{1.75}$$

або, із співвідношень (1.69) та (1.75), остаточно маємо [2 – 5, 8]:

$$n_k = 2^{T\left(\bar{I}(Z)+\varepsilon\right)}. \tag{1.76}$$

Аналіз отриманих співвідношень (1.72) та (1.76) показує, що за будь-яких умов виконується тотожність

$$n_k > n_{\text{т}}. \qquad (1.77)$$

Отримане співвідношення (1.77) дозволяє зробити висновок про те, що у разі виконання умови (1.67), тобто, у разі $\bar{I}(Z) < C_{\text{д}}$, кількість кодових комбінацій, яку може пропустити канал зв'язку, є достатньою для кодування всіх типових послідовностей. Більш того, існують лишки кодових послідовностей, які не використані для кодування типових послідовностей символів. Що стосується нетипових послідовностей, то ефективні коди будуються так, що їм відповідають кодові комбінації із більшою кількістю елементів. Під час побудови кодів нетипових послідовностей на початку та в кінці коду ставлять кодові комбінації, які не були використані у типових послідовностях, що дозволяє на приймальній стороні легко відрізняти закодовані символи абетки, навіть за умови відсутності засобів синхронізації, які були описані у підрозділі 4.4 першої частини посібника [1]. Слід мати на увазі, що оскільки доведення теореми 1.2 базується на асимптотичному визначенні імовірності появи типових послідовностей, за умови, що кількість символів в повідомлені N та тривалість передавання типових послідовностей T прямують до нескінченності ($N\to\infty$, $T\to\infty$), імовірність появи нетипових послідовностей символів наближається до нуля. Тобто, у реальних системах зв'язку нетиповими послідовностями зазвичай можна нехтувати. Слід відзначити, що через виділення типових та нетипових послідовностей символів та через подальший аналіз способів кодування саме типових послідовностей із рівними ймовірностями забезпечується рівномірне та незалежне надходження символів на вхід каналу зв'язку, що, згідно із інформаційними оцінками, наведеними у розділах 1 та 4 першої частини посібника [1], дозволяє цілком уникнути надлишковості повідомлень, які мають бути переданими.

Вірність другої частини теореми Шеннона 1.2, яка говорить проте, що у разі виконання умови (1.68) передавання інформації через канал зв'язку без спотворень є неможливим, витікає із того, що пропускна здатність каналу визначається як максимально можлива швидкість передавання інформації

для множини джерел повідомлень даного класу. Тому якщо пропускна здатність каналу є меншою за продуктивність джерела повідомлень, неминуче виникає постійне накопичення інформації на приймальній стороні.

На практиці часто використовують інше формулювання теореми Шеннона 1.2, а саме [2 – 5, 8]: повідомлення джерела із ентропією $H(Z)$ завжди можна закодувати послідовностями символів абетки із кількістю символів m так, що середня кількість символів на знак повідомлення буде наближатися до величини

$$l_{\text{ср}} \geq \frac{H(Z)}{\log_2 m}, \tag{1.78}$$

але не буде меншою за неї.

Теорема 1.2 не лише встановлює обмеження на швидкість передавання інформації через канал зв'язку, але і вказує на існування ефективних кодів, які не лише забезпечують рівномірне та незалежне надходження символів на вхід каналу зв'язку, але й також відповідають максимальній кількості інформації, яка переноситься цими символами та становить $\log_2 m$. На практиці максимальної швидкості передавання інформації вдається досягти у разі кодування повідомлень великими блоками, а гранична величина $\log_2 m$ досягається у разі безмежного збільшення довжини блоків кодування. Наприклад, для двійкового кодування кількість символів на знак повідомлення може бути зменшена до ентропії джерела повідомлень. Дійсно, згідно із (1.78) у разі $m = 1$ маємо [2 – 5, 8]:

$$l_{\text{ср}} \geq H(Z). \tag{1.79}$$

Слід відзначити, що довжина блоків, за допомогою яких передається інформація в каналі зв'язку, зазвичай має суперечливо-компромісний характер. З одного боку, у разі передавання великих повідомлень з використанням довгих блоків, можна значно підвисити ефективність використання каналів зв'язку, а з іншого боку, у цьому випадку ускладнюється передавання коротких повідомлень, які теж часто мають суттєве значення. Наприклад, через передавання центральною станцією

коротких повідомлень здійснюється керування роботою систем зв'язку. Тому в комп'ютерних мережах Ethernet зазвичай для підвищення коефіцієнту використання каналу зв'язку та продуктивності роботи мережі використовують складні інтелектуальні алгоритми, які дозволяють змінювати розмір пакетів, які передаються, залежно від загального розміру повідомлення. Ці алгоритми основані на теорії черг та на аналізі степені завантаженості мережі, математичний апарат теорії черг був розглянутий у підрозділі 6.3 другої частини посібника [49], а основи стандарту Ethernet розглядатимуться у четвертій частині посібника.

Хоча теорема 1.2 явним чином не вказує на спосіб ефективного кодування, проте метод формування ефективних кодів дійсно стає зрозумілим. Під час побудови коду кожного символу абетки необхідно добиватися, щоб сформований код ніс максимальну інформацію. Для цього необхідно, щоб виконувались такі важливі умови.

1. Під час передавання стандартних повідомлень імовірності появи символів 1 та 0 мають бути приблизно рівними.

2. Перші елементи всіх символів мають бути різними.

3. Вибір елементів коду не залежить від значень попередніх символів.

Умови 2 та 3 у деякій мірі є суперечливими, проте їх виконання можна забезпечити. Перші символи формуються відповідно із попередніми кодовими комбінаціями і мають відрізнятися від них, а решта символів вибирається випадковим способом.

Іншим способом аналізу швидкості передавання інформації за умови використання ефективних кодів, який також можна розглядати як доведення теореми 1.2, є метод невизначених множників Лагранжа [43]. Розглянемо задачу ефективного кодування у загальному вигляді. Будемо вважати, що кодувальний пристрій має передати m кодових послідовностей t_1, … t_m, імовірності появи яких становлять p_1, … p_m. Якщо для передавання повідомлень із тривалістю T необхідно N комбінацій, а комбінації, яка передається, відповідає кількість символів n_i, у загальному випадку маємо:

$$N = \sum_{i=1}^{m} n_i; \sum_{i=1}^{m} p_i = 1; T = N \sum_{i=1}^{m} p_i t_i. \tag{1.80}$$

З іншого боку, згідно із теоремою Шеннона, розглянутою у підрозділі 4.13 першої частини посібника [1]:

$$I = -N \sum_{i=1}^{m} p_i \log_2(p_i). \tag{1.81}$$

Відповідно до теорії невизначених множників Лагранжа, перепишемо співвідношення (1.81) з урахуванням (1.80) через такий функціонал [43]:

$$F = -N \sum_{i=1}^{m} p_i \log_2(p_i) + \lambda_1 \left(1 - \sum_{i=1}^{m} p_i \right) + \lambda_2 \left(T - N \sum_{i=1}^{m} p_i t_i \right). \tag{1.82}$$

Теорія невизначених множників Лагранжа полягає у тому, що для визначення оптимального способу кодування символів та максимальної швидкості передавання інформації необхідно знайти часткові похідні від функціоналу (1.82) $\frac{\partial F}{\partial N}$ та $\frac{\partial F}{\partial p_i}$, та, вважаючи їх рівними нулю, розв'язати отриману систему алгебраїчних рівнянь. Відповідно маємо [43]:

$$\frac{\partial F}{\partial N} = -\sum_{i=1}^{m} p_i \log_2(p_i) - \lambda_2 \sum_{i=1}^{m} p_i t_i = 0; \tag{1.83}$$

$$\frac{\partial F}{\partial p_i} = -N \left(\log_2(p_i) + \frac{1}{\ln 2} \right) - \lambda_1 + \lambda_2 N t_i = 0. \tag{1.84}$$

Розв'язуючи рівняння (1.66), отримуємо [5, 43]:

$$\log_2(p_i) = -\frac{\lambda_1}{N} - \lambda_2 t_i - \frac{1}{\ln 2}. \tag{1.85}$$

Тоді, підставляючи отриманий вираз (1.85) до рівняння (1.83), маємо наступний результат [43]:

$$\sum_{i=1}^{m} p_i \left(-\frac{\lambda_1}{N} - \frac{1}{\ln 2} \right) = 0,$$

або, враховуючи умову нормування ймовірностей із системи рівнянь (1.80):

$$\lambda_1 = -\frac{N}{\ln 2}. \tag{1.86}$$

Для знаходження ймовірностей появи символів p_i, визначене значення

λ_1 підставляємо до співвідношення (1.85). Остаточно маємо [5, 43]:

$$\log_2(p_i) = -\lambda_2 t_i,$$

або:

$$p_i = 2^{-\lambda_2 t_i}. \qquad (1.87)$$

Співвідношення (1.70) характеризує один із головних принципів ефективного кодування. Згідно із цим співвідношенням кодові комбінації із малою імовірністю мають більшу тривалість, а комбінації із високою імовірністю – відповідно, меншу.

Тепер, підставляючи отриманий вираз (1.87) до співвідношення (1.81), можна визначити максимальну кількість інформації, яка передається [43]:

$$I_{\max} = -N \sum_{i=1}^{m} \lambda_2 t_i p_i = \lambda_2 N \sum_{i=1}^{m} t_i p_i,$$

або, враховуючи третє співвідношення системи (1.80) [5]:

$$I_{\max} = \lambda_2 T. \qquad (1.88)$$

Тобто, множник Лагранжа λ_2 відповідає швидкості передавання інформації [5]:

$$\lambda_2 = \frac{I_{\max}}{T} = C. \qquad (1.89)$$

Остаточно, з урахуванням співвідношень (1.85) та (1.89), швидкість передавання інформації та час передавання символу складають [5]:

$$\lambda_2 = C; \qquad t_i = \frac{\log_2(p_i)}{C}. \qquad (1.90)$$

Для випадку відсутності статистичної залежності між знаками коду конструктивні методи побудови ефективних кодів запропонували відомі американські вчені Шеннон, Фано та Хаффмен. Використання кодів Шеннона – Фано та кодів Хаффмена дозволяє значно зменшити надлишковість повідомлень та підвисити ефективність використання каналів зв'язку. Коди Шеннона – Фано будуть розглянуті у підрозділі 1.5.4 цієї частини посібника, а коди Хаффмена – у підрозділі 1.5.5. Далі, у наступному підрозділі, будуть розглянуті головні переваги та недоліки ефективних кодів.

1.5.3 Переваги та недоліки ефективних кодів

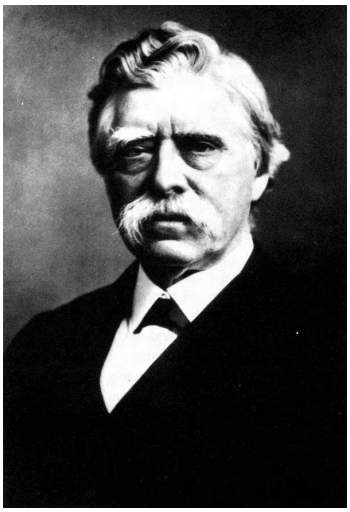
Як було відмічено у попередньому підрозділі, головна перевага ефективних кодів полягає в тому, що вони дозволяють економно та найбільш повно використовувати пропускну здатність каналу зв’язку без завад і забезпечити проходження крізь нього максимальної кількості інформації за одиницю часу.

Проте способи ефективного кодування мають і суттєві недоліки.

По-перше, дещо ускладнюється кодувальна та декодувальна апаратура, що обумовлено різною довжиною кодових комбінацій.

По-друге, канал зв’язку у системах з ефективними кодами ефективно використовується лише у разі рівномірного надходження символів. Тому для забезпечення цієї умови на виході кодувального пристрою передавача встановлюється запам’ятовуючий пристрій. Його роль полягає у тому, що він запам’ятовує символи, які передаються, та надсилає їх до каналу зв’язку із постійною швидкістю. Аналогічні пристрої встановлюються і на приймальній стороні. Тому повідомлення, закодовані ефективними кодами, зазвичай передаються із великою затримкою. Виникнення таких затримок пов’язано із передаванням нетипових послідовностей, імовірність яких є малою, проте вони мають велику довжину.

Роберт Маріо Фано
(1917 – 2016)



Девід Хаффмен
(1925 – 1999)

Третій недолік пов’язаний із тим, що теорія ефективного кодування розроблена для каналу зв’язку без завад. За умови наявності завад значно зменшується достовірність приймання інформації, оскільки ефективні коди не є завадостійкими. Будь-яка помилка ефективного коду може привести до перетворення кодової комбінації в іншу дозволену комбінацію, яка має іншу довжину, що, в свою чергу, приводить до спотворення наступних символів. В результаті невірно декодується не один символ, а ланцюг символів, такий ефект називається «треком помилок». Тому розроблені більш складні способи побудови ефективних кодів, які зводять кількість ланцюгових помилок до мінімуму. Але у сучасних високопродуктивних системах зв’язку частіше застосовуються завадостійкі коди, які будуть розглянуті у розділах 2 та 3 цієї частини посібника.

Четвертий недолік полягає в тому, що всі наведені теоретичні міркування є правильними лише для незалежних подій. Тому в алгоритмах побудови ефективних кодів умовні імовірності появи символів абетки зазвичай не враховуються. Це у деякій мірі обмежує швидкість передавання інформації у системах зв’язку, де використовуються ефективні коди.

1.5.4 Код Шеннона – Фано

Код Шеннона – Фано будується для кодових послідовностей, у яких не існує статистичної залежності між символами абетки. Головний принцип побудови коду цього полягає в тому, що кожний елемент символу, який кодується, обирається таким чином, щоб кількість інформації, яка в ньому міститься, була найбільшою. У кожний момент часу імовірності прийняття кожного елементу, нуля або одиниці, мають бути однаковими. Кількість елементів у різних кодових комбінаціях є різною, тому, згідно із класифікацією, яка наведена у підрозділі 1.1, код Шеннона – Фано є нерівномірним ефективним кодом. У коді Шеннона – Фано реалізовані головні принципи ефективного кодування, які були описані у підрозділі 1.5.1. Тобто, повідомленням, які мають високу імовірність, відповідають короткі

кодові комбінації, а повідомленням із низькою імовірністю – довгі комбінації, і короткі комбінації у будь-якому випадку не є початком довгих. Це дозволяє використовувати коди Шеннона – Фано у асинхронних системах зв'язку без введення додаткових роздільних знаків між окремими символами.

Алгоритм побудови коду Шеннона – Фано є наступним [2 – 5, 8, 33].

1. Символи абетки упорядковано записуються до таблиці за зменшенням їх ймовірностей, загальна кількість символів в абетці становить n.

2. Упорядковані символи абетки розділяються на дві групи таким чином, щоб імовірність їх появи для кожної з груп, по можливості, були близькими одна до одній. В ідеальному випадку ці імовірності можуть бути рівними, проте реалізувати розбиття на групи із рівними ймовірностями вдається не завжди.

3. Формується перший елемент коду. Для всіх символів першої групи цей елемент становить 1, для всіх символів другої групи – 0.

4. Кожна із отриманих груп розбивається на дві підгрупи із приблизно рівними ймовірностями.

5. За алгоритмом, описаним у пункті 3, формується наступний символ коду.

6. Ітераційне повторення операцій, описаних у пунктах алгоритму 4 та 5, до тих пір, доки у кожній групі не залишиться 1 символ.

Ентропія коду Шеннона – Фано визначається за формулою Шеннона (1.8), наведеній у підрозділі 1.2 першої частини посібника [2 – 5, 8, 33]:

$$H(X) = -\sum_{i=1}^{n} p(x_i)\log_2(p(x_i)). \tag{1.91}$$

Максимальна ентропія коду Шеннона – Фано визначається за формулою Хартлі, наведеній у підрозділі 1.2 першої частини посібника [1]:

$$H_{\max}(X) = \log_2(n). \tag{1.92}$$

Тоді надлишковість коду Шеннона – Фано можна визначити із співвідношення [2 – 5, 8, 33]:

$$k = \frac{H_{\max}(X) - H(X)}{H(X)}. \tag{1.93}$$

Середній час передавання повідомлень для коду Шеннона – Фано розраховується із співвідношення [2 – 5, 8, 33]:

$$\overline{T} = \sum_{i=1}^{N} p(x_i)\tau_i, \tag{1.94}$$

де τ_i – тривалість передавання символу. Тоді усереднену швидкість передавання інформації можна розрахувати як [2 – 5, 8, 33]:

$$\overline{I}(X) = \frac{H(X)}{\overline{T}}. \tag{1.95}$$

Слід відзначити, що під час створення реальних кодів за алгоритмом Шеннона – Фано, коли імовірності появи символів абетки у значній мірі розбігаються, у разі кодування одного символу зазвичай не вдається досягти низьких значень коефіцієнта надлишковості *k*. У цьому випадку можна підвисити ефективність коду Шеннона – Фано, якщо будувати кодові послідовності не для одного символу, а для групи із двох і більшої кількості символів. Розширюючи таким чином базову абетку зазвичай вдається досягти більш рівномірного розподілу ймовірностей появи її символів, і відповідно, побудувати більш ефективний код.

Розглянемо описаний вище спосіб побудови коду Шеннона – Фано на конкретному прикладі.

Приклад 1.15. Через канал зв’язку необхідно передавати повідомлення, які складаються із трьох символів із наступними ймовірностями їх появи: $p(x_1) = 0{,}7$; $p(x_2) = 0{,}2$; $p(x_3) = 0{,}1$. Побудувати для заданої абетки натуральний код та код Шеннона – Фано та знайти для обох випадків швидкість передавання інформації, якщо пропускна здатність каналу зв’язку дозволяє передавати 1 елемент повідомлення за 1 мкс.

Натуральний код сформуємо наступним чином. Зрозуміло, що для кодування трьох символів достатньо двох позицій двійкового натурального коду. Нехай $x_1 = 00$; $x_2 = 01$; $x_3 = 10$. Тоді час передавання повідомлень через

канал зв’язку становить:

$$T = 2\tau = 2 \cdot 10^{-6} \text{ сек.}$$

Загальна ентропія повідомлень за умови заданих ймовірностей символів складає:

$$H(X) = -\sum_{i=1}^{3} p(x_i)\log_2\left(p(x_i)\right) = -0{,}7\log_2 0{,}7 - 0{,}2\log_2 0{,}2 - 0{,}1\log_2 0{,}1 = 1{,}16 \frac{\text{біт}}{\text{символ}}.$$

Тоді швидкість передавання інформації:

$$I(X) = \frac{H(X)}{T} = \frac{1{,}16}{2 \cdot 10^{-6}} = 5{,}8 \cdot 10^5 \frac{\text{біт}}{\text{с}} = 0{,}58 \frac{\text{Мбіт}}{\text{с}}.$$

Через розбиття символів абетки на групи побудуємо код Шеннона – Фано. Спосіб виконання цієї операції зрозумілий із таблиці 1.5.

Як видно із таблиці 1.5, на кожній ітерації у відповідний розряд ставиться 0, якщо символ належить до першої групи, або 1, якщо символ належить до другої групи, проте для символу x_1 виконується лише одна ітерація, тому код цього символу є однорозрядним.

Таблиця 1.5 – Принцип побудови коду Шеннона – Фано для прикладу 1.15

Символ	Імовірність	Розбиття на групи		Код	Час передавання символу
		1 ітерація	2 ітерація		
x_1	0,7	I	–	0	τ
x_2	0,2	II	I	10	2τ
x_3	0,1	II	II	11	2τ

Зрозуміло, що підвищення швидкості передавання інформації за умови використання коду Шеннона – Фано забезпечується тим, що, на відміну від натурального коду, символу x_1, імовірність появи якого є максимальною, відповідає не 2 біти, а 1 біт. У цьому випадку середня тривалість кодової комбінації, згідно із співвідношенням (1.77), становить:

$$\overline{T} = \sum_{i=1}^{3} p(x_i)\tau_i = 0{,}7 \cdot 10^{-6}\,\text{сек} + 0{,}2 \cdot 2 \cdot 10^{-6}\,\text{сек} + 0{,}1 \cdot 2 \cdot 10^{-6}\,\text{сек} = 1{,}3\,\text{мкс}.$$

Тоді усереднена швидкість передавання інформації, згідно із співвідношенням (1.95), складає:

$$\overline{I}(X) = \frac{H(X)}{\overline{T}} = \frac{1{,}16}{1{,}3\,\text{мкс}} = 8{,}9 \cdot 10^5\,\frac{\text{біт}}{\text{с}} = 0{,}89\,\frac{\text{Мбіт}}{\text{с}}.$$

Зрозуміло, що для коду Шеннона – Фано швидкість передавання інформації є значно більшою, ніж для натурального коду, і вона наближається до пропускної здатності каналу зв'язку, яка становить $1\,\frac{\text{Мбіт}}{\text{с}}$. Проте, оскільки, як видно із таблиці 1.5, і на першій, і на другій ітерації імовірності символів у групах I та II трішки відрізняються, швидкість передавання інформації є дещо меншою, ніж пропускна здатність каналу зв'язку. Для підвищення швидкості передавання інформації можна кодувати не один символ, а послідовності із двох символів, що дає змогу створювати групи із більш рівномірним розподілом ймовірностей появи символів абетки.

Можна впевнитись у цьому, розглянувши наступний приклад.

Приклад 1.16. Для абетки, заданої у прикладі 1.15, побудувати код Шеннона – Фано для послідовностей із двох символів. Для визначеного коду знайти швидкість передавання інформації, якщо пропускна здатність каналу зв'язку дозволяє передавати 1 елемент повідомлення за 1 мкс.

Як видно із таблиці 1.6, принцип формування коду Шеннона – Фано для послідовностей із двох символів є таким же самим, як і для одного символу. Під час розрахунків ймовірностей появи двох символів поява кожного символу вважається незалежною подією, тому умовні імовірності розраховуються із співвідношення [1, 9, 25, 27, 49]:

$$p(x_i, x_j) = p(x_i)p(x_j). \qquad (1.96)$$

У разі, якщо імовірність появи одного символу залежить від появи іншого, слід розглядати умовні імовірності $p(x_j \mid x_i)$. У цьому випадку умовна імовірність появи послідовності з двох символів розраховується

наступним чином [1, 9, 25, 27, 49]:

$$p(x_i, x_j) = p(x_i)p(x_j \mid x_i). \qquad (1.97)$$

Слід мати на увазі, що за таких умов $p(x_j \mid x_i) \neq p(x_i \mid x_j)$. Наприклад, у разі передавання українського тексту імовірність появи послідовності літер *аб* відрізняється від імовірності появи послідовності літер *ба*. Якщо вважати, що поява символу *а* та символу *б* є незалежними подіями, тоді імовірності появи груп символів *аб* та *ба* співпадають.

Таблиця 1.6 – Принцип побудови коду Шеннона – Фано для прикладу 1.16

Послідовність символів	Імовірність	Розбиття на групи за ітераціями						Код	Час передавання послідовності символів
		1	2	3	4	5	6		
x_1x_1	0,49	I	–	–	–	–	–	0	τ
x_1x_2	0,14	II	I	I	–	–	–	100	3τ
x_2x_1	0,14	II	I	II	–	–	–	101	3τ
x_1x_3	0,07	II	II	I	I	–	–	1100	4τ
x_3x_1	0,07	II	II	I	II	–	–	1101	4τ
x_2x_2	0,04	II	II	II	I	–	–	1110	4τ
x_2x_3	0,02	II	II	II	II	I	–	11110	5τ
x_3x_2	0,02	II	II	II	II	II	I	111110	6τ
x_3x_3	0,01	II	II	II	II	II	II	111111	6τ

Щодо способу формування кодових комбінацій, він залишається незмінним. Якщо набір символів на даній ітерації належить до першої групи – у відповідний розряд ставиться нуль, а якщо до другої – одиниця. Наприклад, якщо комбінації символів x_2x_2, імовірність якої становить 0,04, відповідають номера груп II II II I, код Шеннона – Фано для цієї комбінації становить 1110 (таблиця 1.6). Якщо формування коду послідовності символів закінчено до шостої ітерації, тоді він має меншу кількість розрядів, яка

відповідає кількості проведених ітерацій. Наприклад, формування коду послідовності x_1x_1, імовірність появи якої становить 0,49, завершено вже на першій ітерації, тому код для цієї комбінації символів є однорозрядним і становить 0.

На перший погляд здається, що сформований код для послідовностей із двох символів є менш ефективним, ніж код Шеннона – Фано для одного символу абетки, який був отриманий у прикладі 1.15. По-перше, у разі кодування послідовності із двох символів однорозрядний код з'являється із імовірністю 0,49, а у разі кодування одного символу – із імовірністю 0,7. По-друге, у разі кодування двох символів максимальна розрядність коду становить 6, а не 2, як у разі кодування одного символу. Тому оцінимо ефективність отриманого коду Шеннона – Фано для послідовностей із двох символів та швидкість передавання інформації, використовуючи співвідношення (1.94) та (1.95).

$$\overline{T} = \sum_{i=1}^{9} \frac{p(x_i)\tau_i}{2} = \frac{0{,}49 \cdot 10^{-6}\,\text{сек} + 2 \cdot 0{,}14 \cdot 3 \cdot 10^{-6}\,\text{сек} + 2 \cdot 0{,}07 \cdot 4 \cdot 10^{-6}\,\text{сек}}{2} +$$

$$+ \frac{0{,}04 \cdot 4 \cdot 10^{-6}\,\text{сек} + 0{,}02 \cdot 5 \cdot 10^{-6}\,\text{сек} + 0{,}02 \cdot 6 \cdot 10^{-6}\,\text{сек}}{2} +$$

$$+ \frac{0{,}01 \cdot 6 \cdot 10^{-6}\,\text{сек}}{2} = 1{,}165\,\text{мкс}.$$

Зважаючи на те, що, згідно із розрахунками, проведеними у прикладі 1.15, для заданої імовірності символів абетки ентропія джерела повідомлень становить $H(X) = 1{,}16$, швидкість передавання інформації для коду Шеннона – Фано у разі кодування послідовності із двох символів становить:

$$\overline{I}(X) = \frac{H(X)}{\overline{T}} = \frac{1{,}16}{1{,}165\,\text{мкс}} = 9{,}95 \cdot 10^5\,\frac{\text{біт}}{\text{с}} = 0{,}995\,\frac{\text{Мбіт}}{\text{с}} \approx 1\,\frac{\text{Мбіт}}{\text{с}}.$$

Проведені розрахунки показують, що у разі кодування послідовності із двох символів ефективність коду Шеннона – Фано дійсно є вищою, а швидкість передавання інформації наближається до граничної величини, яка, згідно із теоремою Шеннона 1.2, визначається співвідношенням (1.67) і

обумовлена обмеженою пропускною здатністю каналу зв'язку. Проведений аналіз показує, що у даному разі підвищення швидкості передавання інформації обумовлено тим, що найбільш імовірна послідовність із двох символів x_1x_1 кодується коротким однорозрядним кодом, тобто, два символи абетки передаються за один такт.

1.5.5 Код Хаффмена

Перед вивченням цього підрозділу необхідно підрозділ 7.2 другої частини посібника

Принцип побудови ефективного коду Хаффмена загалом дуже схожий на принцип побудови коду Шеннона – Фано. Проте, на відміну від коду Шеннона – Фано, система кодування коду Хаффмена забезпечує однозначність його побудови із найменшою середньою кількістю елементів на символ абетки за умови заданого розподілу ймовірностей символів. Тобто, алгоритм побудови коду Шеннона – Фано є більш гнучким, а алгоритм побудови коду Хаффмена є більш точним і формалізованим.

Побудова коду Хаффмена здійснюється наступним чином.

1. Як і для коду Шеннона – Фано, послідовно виписуються символи абетки за принципом зменшення ймовірностей їх появи.

2. Із двох останніх символів, які мають найменші імовірності, створюють новий символ, імовірність появи якого визначається як сума ймовірностей появи символів, із яких він складається.

3. Процес, описаний у пункті 2 алгоритму, ітераційно продовжується до тих пір, доки не буде отриманий один допоміжний символ, імовірність появи якого становить 1. Дійсно, поява допоміжного символу, який створений із повного набору символів абетки, завжди є достовірною подією.

4. Необхідно побудувати дерево кодової комбінації, яке відповідає заданій абетці. Із верхньої точки дерева A, яка відповідає імовірності $p = 1$, виходять дві гілки. Гілці, який відповідає більша імовірність, призначається код 1, а гілці, який відповідає менша імовірність – код 0. Таке розгалуження

дерева продовжується, доки не буде сформований останній символ.

5. Через переміщення за гілками кодового дерева, від його вершини до відповідного вузла, записуються коди символів.

Розглянемо приклад побудови коду Хаффмена за описаним алгоритмом.

Приклад 1.17. Побудувати код Хаффмена для абетки символів із такими ймовірностями їхньої появи: $z_1 = 0{,}4$; $z_2 = 0{,}2$; $z_3 = 0{,}2$; $z_4 = 0{,}1$; $z_5 = 0{,}05$; $z_6 = 0{,}05$.

Розглянемо упорядковані символи абетки, які зведені до таблиці, яка наведена на рис. 1.51.

№	Символи повідом-лення	Імовірності появи символів	Допоміжні стовпчики					Код
			1	2	3	4	5	
1.	z_1	0,4	0,4	0,4	0,4	0,6	1	0
2.	z_2	0,2	0,2	0,2	0,4	0,4		10
3.	z_3	0,2	0,2	0,2	0,2			111
4.	z_4	0,1	0,1	0,2				1101
5.	z_5	0,05	0,1					11101
6.	z_6	0,05						11000

Рис. 1.51 Ілюстрація принципу формування коду Хаффмена для прикладу 1.17

Згідно із описаним алгоритмом, почнемо об'єднувати символи із найменшими ймовірностями. Це символи z_6 та z_5, сумарна імовірність появи яких становить 0,1. Імовірності появи символів після здійснення першої операції їх об'єднання наведені у допоміжному стовпчику 1 таблиці (рис. 1.51). Тепер, аналізуючи цей стовпчик, об'єднаємо створений нами груповий символ z_6z_5 із символом z_4. Імовірності обох символів, які об'єднуються, складають 0,1, тобто імовірність символу, який створюється, буде 0,2. Відповідний результат наведений у допоміжному стовпчику 2. Третя ітерація є більш складною. Імовірність появи об'єднаного символу $z_6z_5z_4$ становить 0,2, і його необхідно об'єднати із символом z_3. Проте тоді імовірність появи нового об'єднаного символу $z_6z_5z_4z_3$ становить 0,4 і перевищує імовірність появи символу z_2. Тому у третьому допоміжному

стовпчику верхній символ – це z_1, за ним, у другому рядку, іде символ $z_6z_5z_4z_3$, а далі – нижній символ z_2. Таким же чином, як третій, формується і четвертий допоміжний стовпчик. Необхідно об'єднаний символ $z_6z_5z_4z_3$, імовірність якого становить 0,4, об'єднати із символом z_2, оскільки тепер саме імовірність цього символу, 0,2, є найменшою. Таму у четвертому допоміжному стовпчику у першому рядку стоїть об'єднаний символ $z_6z_5z_4z_3z_2$, імовірність появи якого є найбільшою і становить 0,6, а у другому рядку – символ z_1, імовірність появи якого становить 0,4. У п'ятому допоміжному стовпчику стоїть остаточно сформований об'єднаний символ $z_6z_5z_4z_3z_2z_1$. Оскільки в ньому об'єднані всі символи абетки, імовірність появи цього символу становить 1.

Спосіб формування елементів символів абетки з використанням алгоритму Хаффмена оснований на використанні теорії графів та дерев, яка була описана у підрозділі 7.2 другої частини посібника [50]. Згідно з пунктами алгоритму 4 та 5 необхідно побудувати ієрархічне дерево та вказати імовірності для кожної гілки. Після цього на всіх вузлах розгалуження гілкам із більшою імовірністю ставиться у відповідність значення 1, а гілкам із меншою імовірністю –значення 0. Ці величини розглядаються як вага відповідної гілки для обраного шляху. Якщо імовірності співпадають, вибір значення 0 або 1 може бути довільним.

Для прикладу, який розглядається, дерево для побудови коду Хаффмена показано на рис. 1.52.

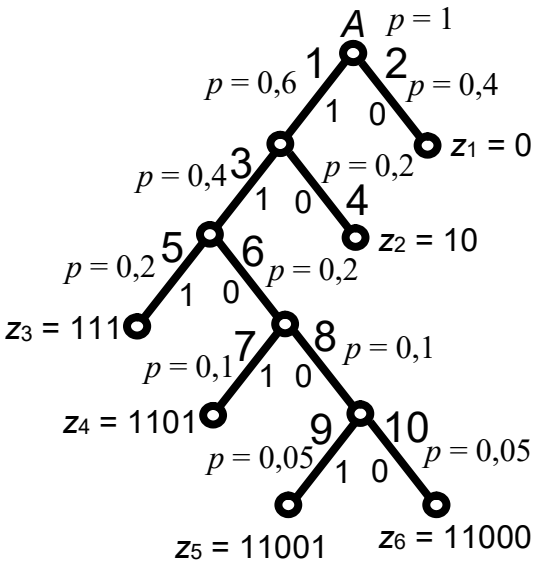


Рис. 1.52 Ієрархічне дерево для побудови коду Хаффмена для прикладу 1.17

Ієрархічне дерево, яке наведено на рис. 1.52, має таку структуру. Вершині дерева відповідає вузол *A*, який описує імовірності появи всіх символів абетки, тому імовірність цієї події дорівнює 1. Від вершини *A* ідуть дві гілки. Гілка 1 описує сумарну імовірність появи символів z_2, z_3, z_4, z_5 та z_6, яка становить 0,6, а гілка 2 – імовірність появи символу z_1, яка складає 0,4. Тому гілці 1 відповідає вага 1, а гілці 2 – відповідно, вага 0. Таким чином, код Хаффмена для символу z_1 становить $z_1 = 0$, а вузол, до якого іде гілка 1, не є кінцевим і від нього розгалужуються дві інші гілки. Гілка 3 відповідає сумарній імовірності появи символів z_3, z_4, z_5 та z_6, яка становить 0,4, а гілка 4 – імовірності появи символу z_2, яка складає 0,2. За таких умов гілці 3 призначаємо вагу 1, а гілці 4 – відповідно, вагу 0. Вузол, до якого іде гілка 4, відповідає символу z_2. Код цього символу формується через шлях, який іде від вершини дерева *A* через гілки 1 та 4. Оскільки гілці 1 відповідає вага 1, а гілці 4 – вага 0, код Хаффмена для символу z_2 становить $z_2 = 10$. Вузол, до якого іде гілка 3, також є проміжним, від нього розгалужуються гілки 5 та 6. Вузол, до якого іде гілка 5, відповідає символу z_3, а вузол, до якого іде гілка 6 – сумарній імовірності символів z_4, z_5 та z_6, яка становить 0,2, як і імовірність символу z_3. Тому, у цьому разі, довільним чином для гілки 5 обираємо вагу 1, а для гілки 6 – вагу 0. Код символу z_3 формується через шлях, який іде від вершини дерева *A* через гілки 1, 3 та 5. Оскільки у даному випадку всім цим трьом гілкам відповідає вага 1, код Хаффмена для символу z_3 становить $z_3 = 111$. А вузол, до якого іде гілка 6, також є проміжним, від нього розгалужуються гілки 7 та 8. Вузол, до якого іде гілка 7, відповідає символу z_4, імовірність якого становить 0,1. Тому код цього символу формується через шлях, який іде від вершини дерева *A* через гілки 1, 3, 6 та 7 і становить $z_4 = 1101$. А вузол, до якого іде гілка 8 – це останній проміжний вузол дерева, від нього розгалужуються гілки 9 та 10. Гілка 9 іде до вузла, який відповідає символу z_5, а гілка 10 – до вузла, який відповідає символу z_6. Імовірності появи цих символів є однаковими, тому довільним чином призначаємо гілці 9 вагу 1, а гілці 10 – вагу 0. Шлях від вершини *A* до вузла,

який відповідає символу z_5, іде через гілки 1, 3, 6, 8 та 9, тому код Хаффмена для символу z_5 становить 11001. Шлях від вершини A до вузла, який відповідає символу z_6, іде через гілки 1, 3, 6, 8 та 10, відповідно, код Хаффмена для символу z_5 становить 11000.

Тобто, остаточно, код Хаффмена для визначених в умові задачі ймовірностей появи символів z_1, z_2, z_3, z_4, z_5 та z_6 становить: $z_1 = 0$, $z_2 = 10$, $z_3 = 111$, $z_4 = 1101$, $z_5 = 11001$, $z_6 = 11000$.

Для обчислення максимальної та мінімальної ентропії коду Хаффмена, а також коефіцієнта його надлишковості, можна, як і для коду Шеннона – Фано, використовувати співвідношення (1.91) – (1.93). Відповідно, для розрахунку середнього часу передавання одного символу можна використовувати співвідношення (1.94), а для розрахунку швидкості передавання інформації – співвідношення (1.95).

Приклад 1.18. Знайти середню тривалість передавання одного символу та швидкість передавання інформації для коду Хаффмена, який був отриманий у прикладі 1.17, якщо пропускна здатність каналу зв'язку дозволяє передавати 1 елемент повідомлення за 1 мкс.

Згідно із формулою Шеннона (1.91) знайдемо загальну ентропію абетки:

$$H(X) = -\sum_{i=1}^{6} p(x_i)\log_2\left(p(x_i)\right) = -0{,}4 \cdot \log_2 0{,}4 - 2 \cdot 0{,}2 \cdot \log_2 0{,}2 -$$

$$- 0{,}1 \cdot \log_2 0{,}1 - 2 \cdot 0{,}05 \log_2 0{,}05 = 2{,}22 \frac{\text{біт}}{\text{символ}}.$$

Середня тривалість кодової комбінації, згідно із співвідношенням (1.94), становить:

$$\overline{T} = \sum_{i=1}^{6} p(x_i)\tau_i = 0{,}4 \cdot 10^{-6}\,\text{сек} + 0{,}2 \cdot 2 \cdot 10^{-6}\,\text{сек} + 0{,}2 \cdot 3 \cdot 10^{-6}\,\text{сек} +$$

$$+ 0{,}1 \cdot 4 \cdot 10^{-6}\,\text{сек} + 2 \cdot 0{,}05 \cdot 5 \cdot 10^{-6}\,\text{сек} = 2{,}3\,\text{мкс}.$$

Тоді, за заданих ймовірностей символів абетки, усереднену швидкість передавання інформації у разі використання сформованого коду Хаффмена можна обчислити із співвідношення (1.95):

$$\bar{I}(X) = \frac{H(X)}{\bar{T}} = \frac{2{,}22}{2{,}3\ \text{мкс}} = 9{,}65 \cdot 10^5\ \frac{\text{біт}}{\text{с}} = 0{,}965\ \frac{\text{Мбіт}}{\text{с}} \approx 1\ \frac{\text{Мбіт}}{\text{с}}.$$

Тобто, використання коду Хаффмена дозволяє досягти середньої швидкості передавання інформації, близької до граничної максимальної величини, яка, згідно із теоремою Шеннона 1.2 та формулою (1.67), обмежена пропускною здатністю каналу зв'язку.

З точки зору оцінки ефективності кодів Хаффмена цікавим є аналіз таких кодів для символів із рівними ймовірностями. Цей випадок є граничним і ефективність кодів Хаффмена за таких умов значно знижується, але саме за цієї причини аналіз таких кодів має важливе теоретичне значення. Тому розглянемо ще два показових приклади.

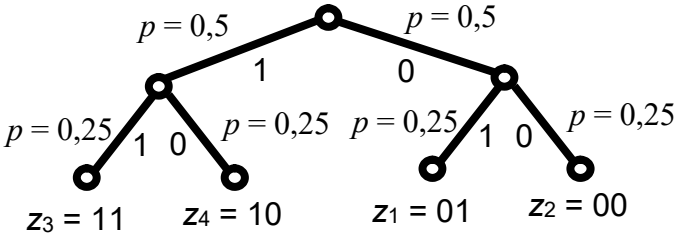
Приклад 1.19. Побудувати код Хаффмена для абетки символів із такими ймовірностями їхньої появи: $p(z_1) = 0{,}25$; $p(z_2) = 0{,}25$; $p(z_3) = 0{,}25$; $p(z_4) = 0{,}25$.

Ієрархічне дерево коду Хаффмена для заданих ймовірностей символів наведене на рис. 1.53, а. Зрозуміло, що оскільки імовірності усіх символів є рівними, сума ймовірностей кожної пари символів буде перевищувати значення імовірності кожного з елементів. Тому дерево коду Хаффмена є розгалуженим, і кількість гілок на кожній із ітерацій зменшується в два рази. Із рис. 1.53, а, зрозуміло, що за таких умов код Хаффмена співпадає із натуральним кодом, тобто: $z_1 = 01$, $z_2 = 00$, $z_3 = 11$, $z_4 = 10$.

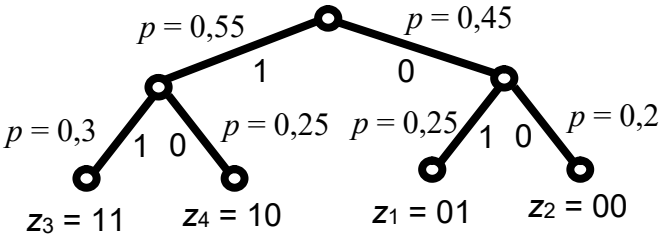
У загальному випадку можна показати, якщо кількість символів абетки становить 2^n та імовірності їхньої появи є однаковими, код Хаффмена для такої абетки співпадає із натуральним кодом.

Слід відзначити, що розглянутий приклад є також важливим і з практичної точки зору. Річ у тому, що аналогічний розв'язок завдання побудови коду Хаффмена буде за умови, коли кількість символів абетки складає 2^n та значення ймовірностей їхньої появи у повідомленнях є близькими одне до одного. Наприклад, ієрархічне дерево коду Хаффмена для значень ймовірностей появи символів $p(z_1) = 0{,}3$; $p(z_2) = 0{,}25$; $p(z_3) = 0{,}25$;

$p(z_4) = 0{,}2$, показане на рис. 1.53, б. Зрозуміло, що структура цього дерева є ідентичною до структури, наведеної на рис. 1.53, а, а коди символів $z_1 - z_4$ співпадають із їхніми натуральними кодами.



а)



б)

Рис. 1.53 Ієрархічні дерева коду Хаффмена для прикладу 1.19

Приклад 1.20. Побудувати код Хаффмена для абетки символів із такими ймовірностями їхньої появи: $p(x_1) = p(x_2) = p(x_3) = p(x_4) = p(x_5) = p(x_6) = \frac{1}{6}$ та оцінити ефективність сформованого коду.

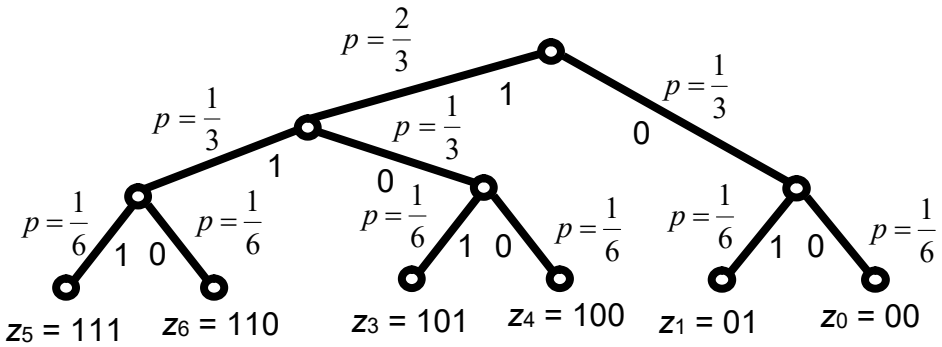


Рис. 1.54 Ієрархічні дерева коду Хаффмена для прикладу 1.20

Ієрархічне дерево коду Хаффмена для заданих значень ймовірностей появи символів наведено на рис. 1.54. Із структури цього дерева зрозуміло,

що у цьому випадку формуються символи різної довжини: $z_1 = 01$, $z_2 = 00$, $z_3 = 101$, $z_4 = 100$, $z_5 = 111$, $z_6 = 110$.

Для даного прикладу оцінимо ефективності коду за умови, що час передавання одного біту складає 1 мкс. Ентропія визначеної абетки складає:

$$H(X) = -\sum_{i=1}^{6} p(x_i)\log_2(p(x_i)) = -\frac{6}{6}\cdot\log_2\frac{1}{6} = -\log_2\frac{1}{6} = 2{,}585\,.$$

Для натурального коду всі шість символів абетки необхідно кодувати трьома бітами, тому час передавання всіх символів абетки є однаковим і складає 3 мкс. Відповідно, швидкість передавання інформації за таких умов становить:

$$\bar{I}(X) = \frac{H(X)}{\bar{T}} = \frac{2{,}585}{3\text{ мкс}} = 8{,}16\cdot 10^5\,\frac{\text{біт}}{\text{с}} = 0{,}816\,\frac{\text{Мбіт}}{\text{с}}.$$

Тобто, за умови використання натурального коду швидкість передавання інформації є дещо меншою, ніж гранична величина за теоремою Шеннона 1.2, яка складає $1\,\frac{\text{Мбіт}}{\text{с}}$. Для сформованого коду Хаффмена середній час передавання символу абетки становить:

$$\bar{T} = \sum_{i=1}^{6} p(x_i)\tau_i = 4\cdot\frac{1}{6}\cdot 3\cdot 10^{-6}\,\text{сек} + 2\cdot\frac{1}{6}\cdot 2\cdot 10^{-6}\,\text{сек} =$$

$$= \left(2+\frac{2}{3}\right)\cdot 10^{-6}\,\text{сек} \approx 2{,}66\cdot 10^{-6}\,\text{сек} = 2{,}66\,\text{мкс}\,.$$

Тобто, у випадку використання ефективного коду Хаффмена швидкість передавання інформації є значно вищою і становить:

$$\bar{I}(X) = \frac{H(X)}{\bar{T}} = \frac{2{,}585}{2{,}66\text{ мкс}} = 9{,}72\cdot 10^5\,\frac{\text{біт}}{\text{с}} = 0{,}972\,\frac{\text{Мбіт}}{\text{с}}.$$

Тобто, для ефективного коду Хаффмена швидкість передавання інформації є близькою до граничної величини, яка визначається теоремою Шеннона 1.2

Існує два різновиди коду Хаффмена: однопрохідний та двопрохідний код [5, 33]. Для двопрохідного коду імовірності появи символів розраховуються безпосередньо для тексту повідомлення, яке кодується, а у разі використання однопрохідного коду використовується узагальнена таблиця

ймовірностей для повідомлень даного типу. Наприклад, у таблиці 1.7 наведені імовірності появи символів української абетки, а у таблиці 1.8 – імовірності появи символів абеток популярних європейських мов, в яких використовуються латинські літери [23, 61].

Таблиця 1.7 – Частоти вживання літер української абетки у відносних одиницях

Літера	Частота	Літера	Частота	Літера	Частота
А	0,072	**Ї**	0,006	**У**	0,04
Б	0,017	**Й**	0,008	**Ф**	0,001
В	0,052	**К**	0,035	**Х**	0,012
Г	0,016	**Л**	0,036	**Ц**	0,006
Д	0,035	**М**	0,031	**Ч**	0,018
Е	0,017	**Н**	0,065	**Ш**	0,012
Є	0,008	**О**	0,094	**Щ**	0,001
Ж	0,009	**П**	0,029	**Ю**	0,004
З	0,023	**Р**	0,047	**Я**	0,029
И	0,061	**С**	0,041	**Ь**	0,029
І	0,057	**Т**	0,055	**'**	0,17

Головною перевагою двопрохідного коду Хаффмена є можливості більшого ущільнення інформації та більш висока ефективність використання ресурсів каналу зв’язку, проте недолік двопрохідного кодування полягає у тому, що разом із повідомленням необхідно передавати і таблицю кодів. Однопрохідний код Хаффмена є не настільки оптимальним з точки зору ущільнення інформації, як двопрохідний, проте він є більш простим для використання, тому у реальний системах зв’язку зазвичай застосовують саме однопрохідний код.

Таблиця 1.8 – Частоти вживання літер латинської абетки для популярних європейських мов

Літера абетки	Частоти появи літер у відносних одиницях для різних мов				
	Англійська	Німецька	Французька	Іспанська	Італійська
A	0,0796	0,0552	0,0768	0,129	0,1112
B	0,016	0,0156	0,008	0,0103	0,0107
C	0,0284	0,0294	0,0332	0,0442	0,0411
D	0,0401	0,0491	0,0360	0,0467	0,0354
E	0,1286	0,1918	0,1776	0,01415	0,01163
F	0,0262	0,0196	0,0106	0,0070	0,0115
G	0,0199	0,036	0,011	0,01	0,0173
H	0,0539	0,0502	0,0064	0,0091	0,0083
I	0,0777	0,0821	0,0723	0,0701	0,01204
J	0,0016	0,0016	0,0019	0,0024	0
K	0,0041	0,0133	0	0	0
L	0,0351	0,0348	0,0589	0,0552	0,0595
M	0,0243	0,0169	0,0272	0,0255	0,0265
N	0,0751	0,102	0,0761	0,062	0,0768
O	0,0662	0,0214	0,0534	0,0884	0,0892
P	0,0181	0,0054	0,0324	0,0326	0,0266
Q	0,0017	0,0001	0,0134	0,0155	0,0048
R	0,0683	0,0701	0,0681	0,0695	0,0656
S	0,0662	0,0707	0,0823	0,0764	0,0481
T	0,0972	0,0586	0,073	0,0436	0,0707
U	0,0248	0,0422	0,0605	0,04	0,0309
V	0,0115	0,0084	0,0127	0,0067	0,0167
W	0,018	0,0138	0	0	0
X	0,0017	0	0,0054	0,0007	0
Y	0,0152	0	0,0021	0,0105	0
Z	0,0005	0,0117	0,0007	0,0031	0,0124

1.5.6 Способи апаратної реалізації кодера та декодера коду Хаффмена

Перед вивченням цього підрозділу необхідно підрозділ 7.3 другої частини посібника

Розглянемо інший приклад формування коду Хаффмена та спосіб апаратної реалізації кодера та декодера цього коду для конкретного набору символів із заданими ймовірностями.

Приклад 1.21. Знайти код Хаффмена для набору із восьми символів $z_1 - z_8$, імовірності яких задані в таблиці, наведеній на рис. 1.55, та побудувати ієрархічне дерево, яке відповідає цьому коду.

У даному випадку на рис. 1.55 одночасно показані імовірності появи кодів символів та алгоритм побудови коду Хаффмена. Як і для прикладу 1.17, у даному випадку також обираються набори символи із найменшими ймовірностями та формуються із них відповідні групи. Наприклад, першою операцією є об'єднання символів z_7 та z_8. Оскільки $p(z_7) = 0{,}02$, а $p(z_8) = 0{,}04$, сумарна імовірність появи цих символів, якщо вони є статистично незалежними, складає $p(z_7) + p(z_8) = 0{,}06$. Аналогічно шукаються імовірності появи інших груп символів, після чого вони впорядковуються за зростанням. Ця процедура цілком відповідає алгоритму формування коду Хаффмена, який був розглянутий у підрозділі 1.5.5. Відповідні кроки цієї процедури наочно показані на рис. 1.55.

№	Символи повідом-лення	Імовірності появи символів	Допоміжні стовпчики						
			1	2	3	4	5	6	7
1.	z_1	0,22	0,22	0,22	0,26	0,32	0,42	0,58	1
2.	z_2	0,2	0,2	0,2	0,22	0,26	0,32	0,42	
3.	z_3	0,16	0,16	0,16	0,2	0,22	0,26		
4.	z_4	0,16	0,16	0,16	0,16	0,2			
5.	z_5	0,1	0,1	0,16	0,16				
6.	z_6	0,1	0,1	0,1					
7.	z_7	0,04	0,06						
8.	z_8	0,02							

Рис. 1.55 Алгоритм побудови ефективного коду Хаффмена для прикладу 1.21

За відомою послідовністю об’єднання символів абетки можна побудувати ієрархічно дерево для коду, який формується, і, відповідно, знайти двійковий код для кожного символу. Структура такого дерева для прикладу, який розглядається, показана на рис. 1.56. Там же показані двійкові коди сформованих символів. Наприклад, код символу z_7, імовірність появи якого складає 0,04, становить 10100. Спосіб формування коду Хаффмена був досконало описаний у прикладі 1.17 і для прикладу, який розглядається, він ні чим не відрізняється. Тобто, кожній гілці дерева, відповідно до імовірності появи заданої групи символів, присвоюється відповідна вага, яка може дорівнювати або 0, або 1. Тоді, після побудови дерева, для формування кодів необхідно пройти весь шлях від кореня до вершини, яка відповідає заданому символу, і послідовно записувати вагу кожної гілки. Саме таким чином і формуються двійкові коди символів.

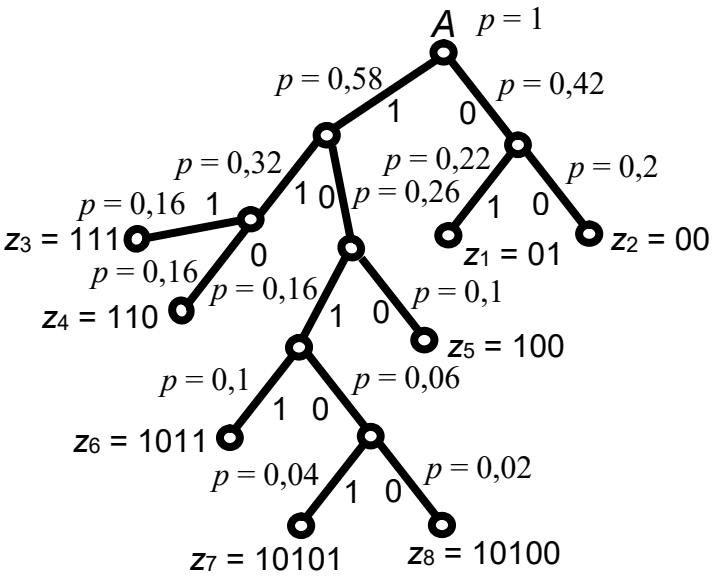


Рис. 1.56 Ієрархічне дерево для побудови коду Хаффмена для прикладу 1.21

Схема кодера для формування коду Хаффмена згідно із алгоритмом для прикладу 1.19, показана на рис. 1.57 [5]. Розглянемо особливості роботи цієї схеми.

Схема, наведена на рис. 1.57, складається із двох матричних МШ1 та МШ2. Матричний шифратор МШ1 побудований на основі регістра зсуву 1, а матричний шифратор МШ2 – на основі регістра зсуву 2. Із структури наведеної принципової електричної схеми зрозуміло, що шифратор МШ1 є головним, а шифратор МШ2 призначений для керування зчитуванням інформації. Зрозуміло також, що кількість горизонтальних електричних ліній зв’язку у обох шифраторах відповідає кількості символів абетки, а кількість вертикальних ліній – найбільшій кількості розрядів у числових послідовностях коду Хаффмена, який формується.

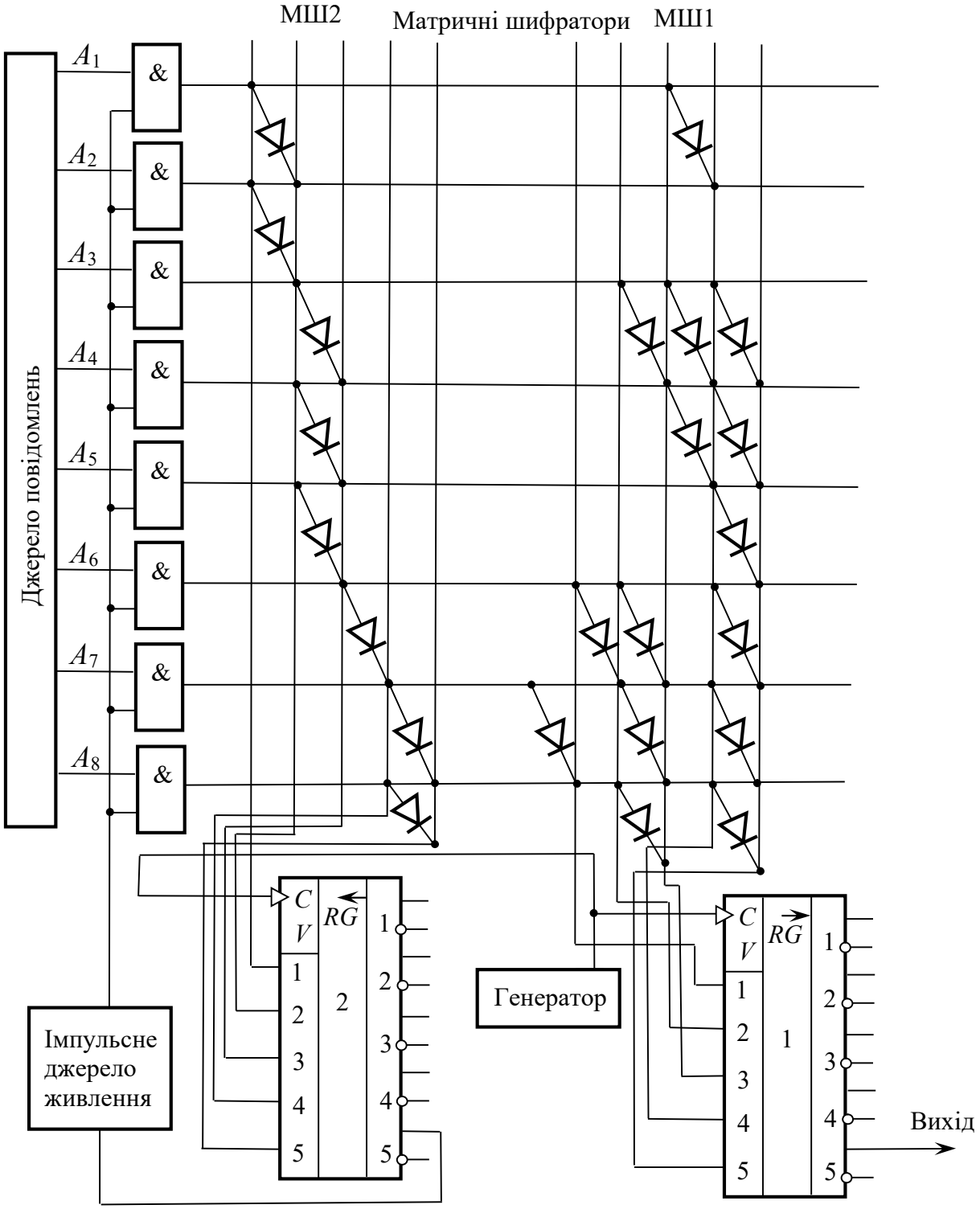


Рис. 1.57 Принципова електрична схема кодера для створення коду Хаффмена для прикладу 1.21

Вмикання діодів у вузлах горизонтальної електричної лінії з номером i забезпечує запис у регістр зсуву 1 кодової комбінації, яка відповідає символу z_i. Роль допоміжного шифратора МШ2 у схемі, наведеній на рис. 1.57,

полягає у тому, що діоди, які підключені до горизонтальних шин, забезпечують запис одиниці в ті комірки регістра 2, номер яких відповідає числу розрядів у коді символу z_i.

Кодування чергового знака z_i, переданого джерелом повідомлень, виконується шляхом подання через логічні елементи І імпульсу від імпульсного джерела живлення. Зрозуміло, що сигнал формується лише на тих горизонтальних лініях шифратора МШ1, які пов'язані через діоди із сигналами символу z_i. В результаті в регістр зсуву 1 записується кодова комбінація, яка відповідає символу z_i, а в регістр зсуву 2 – одиниця, яка несе інформацію про завершення кодової комбінації.

Імпульси від генератора необхідні для того, щоб порозрядно передати записану кодову комбінацію до каналу зв'язку. Таким же чином, через подання імпульсів від генератора, здійснюється зсув одиниці у регістрі 2. Імпульс, який відповідає цій одиниці, з'являється на виході регістра 2 у той момент, коли із регістра 1 виведено останній символ кодової комбінації. У такому разі цей символ може бути використаний для переходу до кодування наступного знаку.

Зрозуміло, що описаний алгоритм роботи кодера можна описати на мові скінченних автоматів, яка була розглянута у підрозділі 7.3 другої частини посібника. Блок-схема цього алгоритму наведена на рис 1.58.

Принципова електрична схема декодувального пристрою для коду Хаффмена є більш складною і наведена на рис. 1.59 [5]. Ця схема була розроблена американськими інженерами Гілбертом і Муром.

Схема декодера працює наступним чином [5]. Символи кодової комбінації, яку необхідно декодувати, послідовно проходять через тригерні пристрої згідно із сигналами тактового генератора імпульсів. Головна проблема декодування коду Хаффмена полягає у тому, що, як видно на рис. 1.56, декілька різних символів можуть починатися із послідовностей нулів, які йдуть підряд. Зокрема, це символи $z_2 = 00$, $z_5 = 100$ та $z_8 = 10100$.

Тому, безпосередньо за вмістом регістра неможливо визначити початок таких комбінацій, щоб правильно їх декодувати.

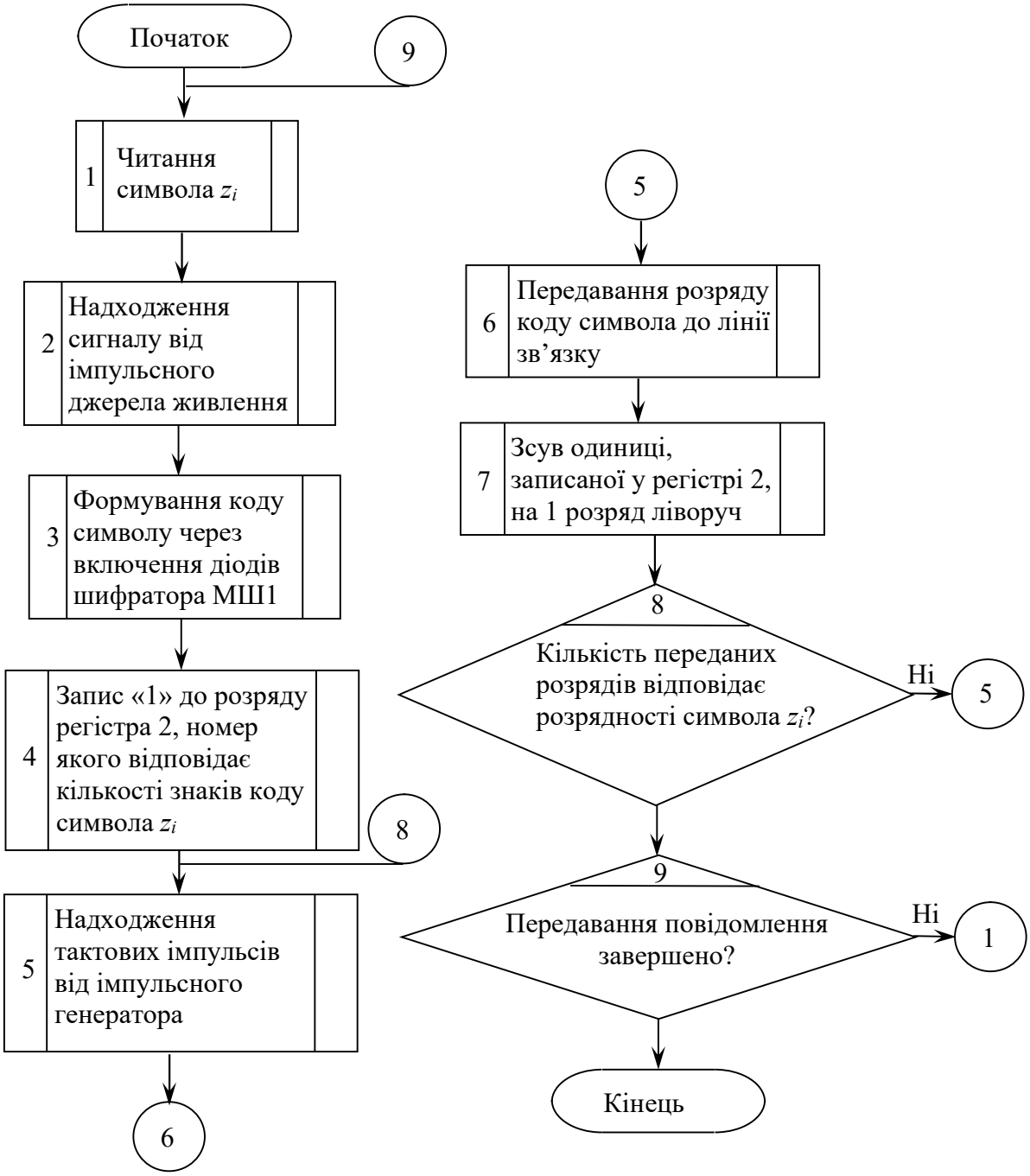


Рис. 1.58 Блок-схема алгоритму роботи схеми кодера, наведеної на рис. 1.57

З цієї причини для однозначного визначення початку кожної нової кодової комбінації кількість комірок регістру обирається на одиницю більшою, ніж кількість позицій у найдовших символах коду. Оскільки, як видно на рис. 1.56, для прикладу 1.21 найдовші символи коду мають 5 позицій, обираємо 6 комірок регістру, побудованих на основі тригерних пристроїв. Відповідна схема підключення тригерних пристроїв показана на рис. 1.59.

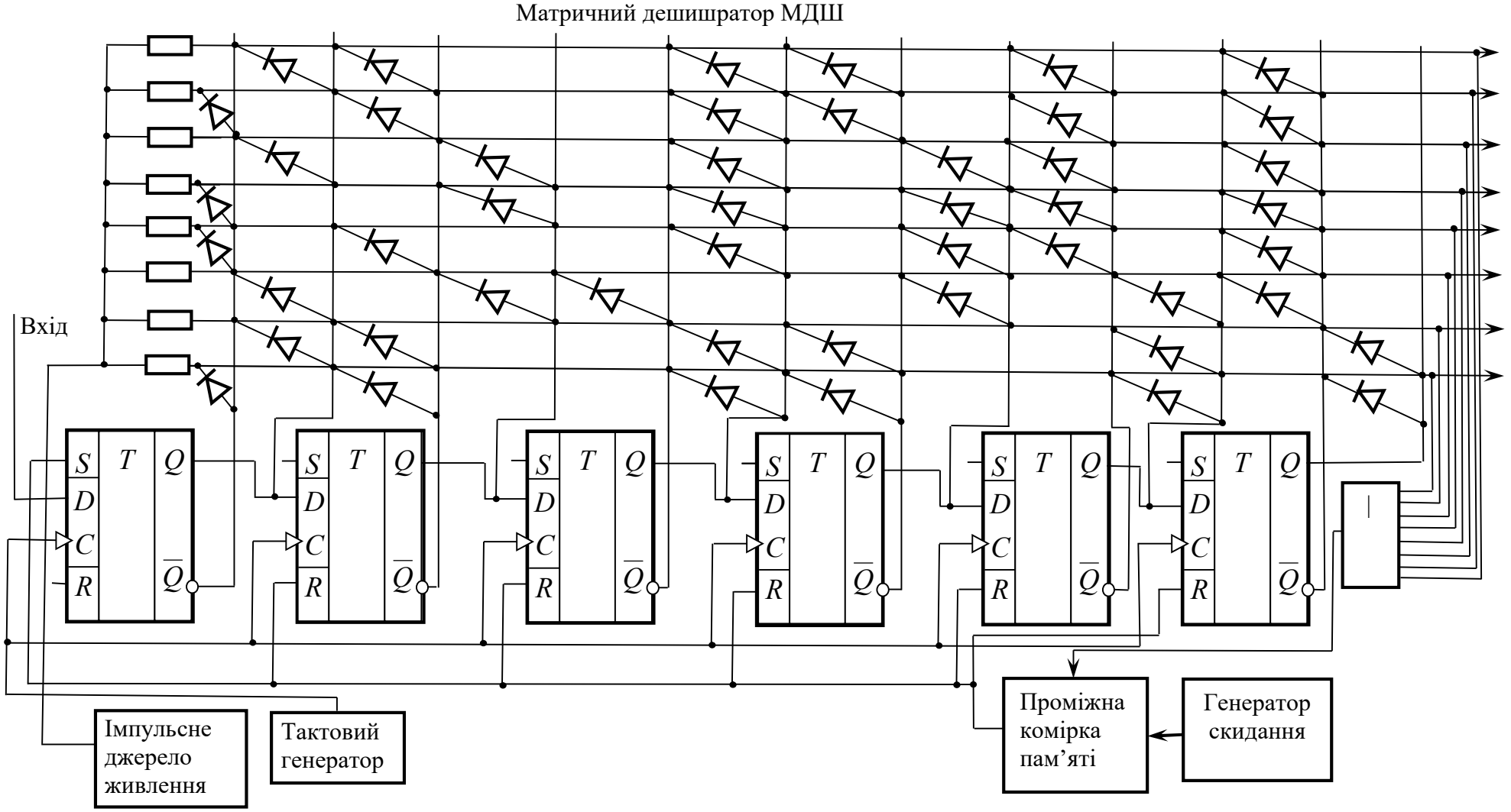


Рис. 1.59 Принципова електрична схема дешифратора коду Хаффмена, описаного у прикладі 1.21

У схемі декодера, яка розглядається, до додаткової першої комірки регістра безпосередньо перед надходженням в нього кодової комбінації записується одиниця. Тоді переміщення цієї одиниці по коміркам регістра свідчить про початок кодової комбінації, що дозволяє визначити кількість розрядів у ній.

Після кожного тактового імпульсу надходить імпульс з джерела живлення, який подає електричну енергію на матричний дешифратор МДШ, який побудований згідно із кодовими комбінаціями, що використовуються, проте у старшому, шостому розряді коду, ставиться одиниця.

Під час надходження до регістру останнього розряду з кодової комбінації поточного символу, який декодується, черговий імпульс від імпульсного джерела приводить до появу імпульсу напруги на шині матричного дешифратора із номером i, що сигналізує про закінчення приймання символу z_i. Цей імпульс, згідно із схемою, наведеною на рис 1.59, через логічну схему АБО записується до проміжної комірки пам'яті. Тому під час дії чергового імпульсу зчитування регістр за допомогою генератора скидання встановлюється у початковий стан. Тобто, в усі комірки регістру, крім першої, записується значення нуль, а в першу комірку – значення одиниці.

Апаратна реалізація декодера коду Хаффмена, як і реалізація його кодера, безпосередньо пов'язана із теорією скінченних автоматів яка була розглянута у підрозділі 7.3 другої частини посібника [50]. Алгоритм роботи декодувального пристрою, принципова електрична схема якого наведена на рис. 1.59, показана на рис. 1.60.

Цілком зрозуміло, що головним недоліком апаратної реалізації кодера та декодера коду Хаффмена, схеми яких наведені на рис. 1.57 та 1.59, є їхня орієнтація на конкретний набір символів абетки із заданими ймовірностями їхньої появи. Для кодів Хаффмена цієї проблеми можна уникнути, якщо використовувати сучасні цифрові мікросхеми ПЛІС, у яких передбачені можливості перепрограмування логічних зв'язків [1]. Способи реалізації інших сучасних ефективних кодів розглядатимуться у підрозділі 1.5.8.

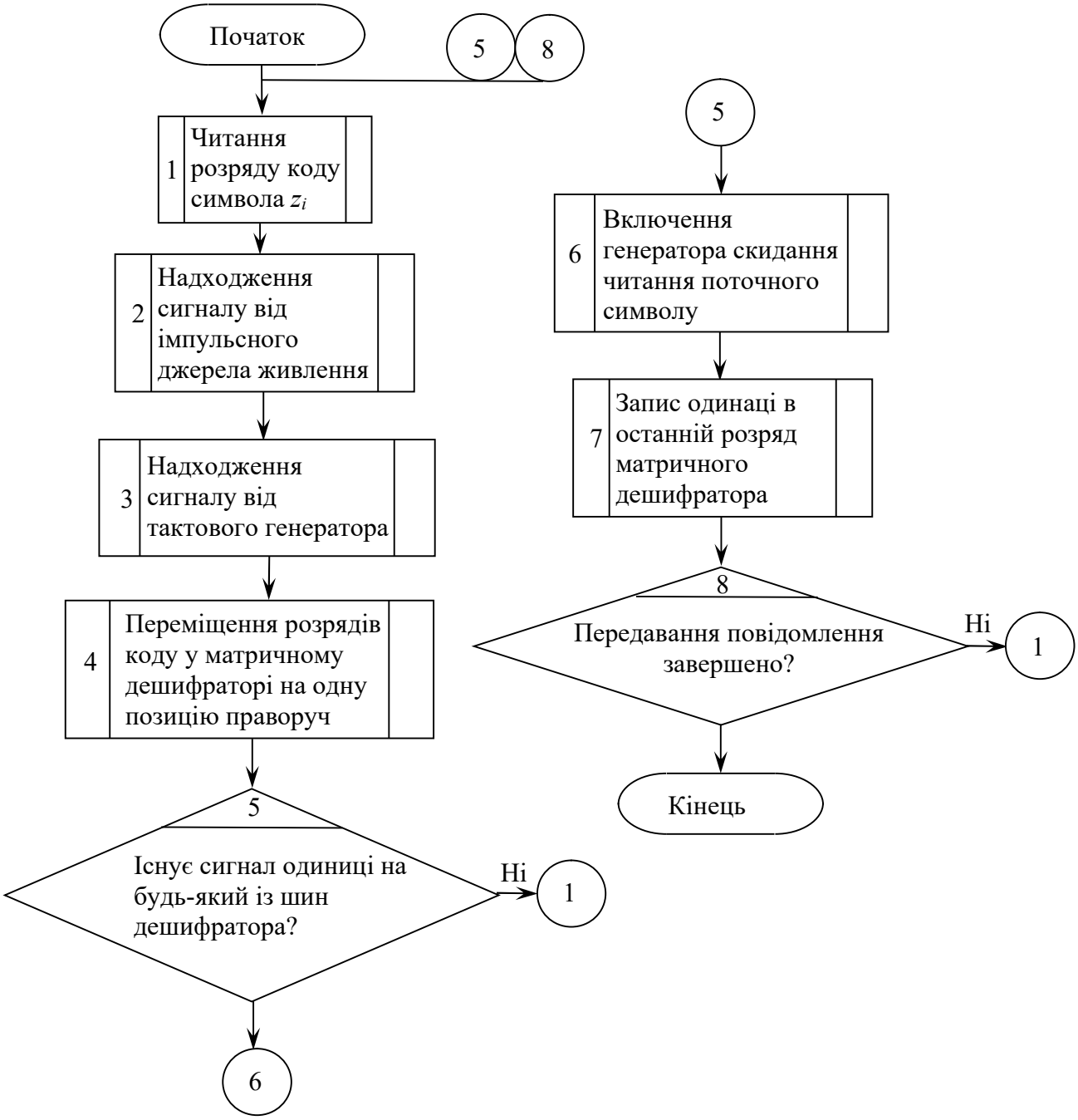


Рис. 1.60 Блок-схема алгоритму роботи схеми декодера, наведеної на рис. 1.54

1.5.7 Реалізація алгоритму побудови коду Хаффмена з використанням засобів матричного програмування

Перед вивченням цього підрозділу необхідно повторити підрозділи 2.4.4 та 7.2 другої частини посібника.

Зрозуміло, що ручний спосіб отримання коду Хаффмена, оснований на побудові ієрархічного дерева та описаний у підрозділі 1.5.5, є дуже простим та наочним, але в той же час вкрай громіздким та неефективним з технічної

точки зору для реалізації в електронних системах обробки інформації. Розглянемо тепер алгоритми автоматичного формування коду Хаффмена та можливості їхньої реалізації у комп'ютерних програмах. Теоретичним підґрунтям для створення таких алгоритмів є теорія сортування елементів групи, розглянута у підрозділі 2.4 другої частини посібника [48], а також теорія графів та дерев, розглянута у підрозділі 7.2 другої частини посібника [50].

Головна проблема написання комп'ютерних програм для автоматичного формування коду Хаффмена за умови відомих ймовірностей появи символів абетки полягає у тому, що алгоритм формування шляхів у дереві, або синтезу структури дерева за відомими вхідними даними, вкрай погано формалізований і не піддається строгій автоматизації. Такі алгоритми зазвичай є значно складнішими, ніж алгоритми Дайкстри для пошуку найкоротшого шляху у дереві, які були розглянуті у підрозділі 7.2.6 другої частини посібника. Алгоритми аналізу графів через матриці ідентифікації та суміжності за методом Флойда – Уоршелла, розглянуті у підрозділі 7.2.3 другої частини посібника, у класичному вигляді також є непридатними для розв'язування поставленої задачі пошуку кодів символів через ієрархічне дерево, оскільки ієрархічне дерево, яке створюється для побудови коду Хаффмена, є зваженим і частина його ребер має вагу, рівну 0. У даному разі, щоб уникнути неоднозначності описання двох різних ситуацій, а саме, ребер від однієї вершини до іншої із вагою 0 та відсутності зв'язків між відповідними вершинами зваженого графа, можна обрати для зв'язків, які відсутні, інше позначення. Наприклад, позначимо відсутність зв'язків між вершинами числом –1. Із такою модифікацією за матрицею, аналогічною матриці ідентифікації, можна аналізувати зв'язки між вершинами ієрархічного дерева коду Хаффмена і через такий аналіз знаходити коди символів. Як і в класичному алгоритмі Флойда – Уоршелла, слід шукати шлях від вершини, яка відповідає символу, код якого формується, до кореня дерева. Дійсно, якщо шлях від вершини одного символу приводить нас до вершини іншого, він є хибним, оскільки алгоритм побудови ієрархічного

дерева коду Хаффмена реалізований таким чином, що інших зв'язків між символами, крім зв'язку через корінь дерева, не існує. Для побудова матриці ідентифікації ієрархічного дерева коду Хаффмена необхідно відповідним чином змінити його структуру. У деревах, наведених на рис. 1.52, 1.53, 1.54 та 1.56, на гілках, крім їхньої ваги, вказані сумарні значення ймовірностей символів, що у значній мірі спрощує розуміння принципу побудови коду Хаффмена. Проте на цих рисунках відсутня нумерація всіх вершин графа, а ребра графа пронумеровані лише на рис. 1.52. Там нумерація ребер була введена для спрощення аналізу шляху від кінцевих вершин, які відповідають символам, до кореня графа. Для того, щоб за ієрархічним деревом коду Хаффмена можна було побудувати матрицю ідентифікації, необхідно зробити наступні перетворення структури дерева.

1. Видалити всі значення ймовірностей з ребер дерева, оскільки на даному етапі аналізу структури дерева вони є зайвими. Тепер нам достатньо знати вагу ребра, яка приймає значення 0 або 1.

2. Пронумерувати всі вершини і ребра сформованого дерева. Для спрощення подальшого аналізу матриці ідентифікації скористаємося таким способом нумерації вершин. Якщо вершина відповідає символу із номером i, будемо вважати, що номер цієї вершини також дорівнює i. Таким чином, всі кінцеві вершини ієрархічного дерева коду Хаффмена, крім кореневої вершини, будуть пронумеровані. Далі слід надати номера внутрішнім вершинам та кореню дерева. Зрозуміло, що нумерація цих вершин буде починатися з $n + 1$, де n – кількість символів, які кодуються. Коли всі m внутрішніх вершин дерева будуть пронумеровані, кореню дерева надається останній номер, якій буде складати $N = n + m + 1$. Щодо нумерації ребер графа, тут не існує будь-яких загальних правил, проте зазвичай ребра нумерують від найбільш віддалених зовнішніх вершин до кореня дерева, тобто знизу до гори. У цьому разі кореню дерева відповідає максимальне значення як номера вершини, так і номерів ребер, які до нього підходять. Така впорядкована нумерація вузлів та ребер ієрархічного дерева спрощує алгоритм пошуку бітів кодів символів через аналіз матриці ідентифікації,

який буде розглянутий далі.

У разі, якщо всі описані перетворення дерева зроблені, матриця ідентифікації для ієрархічного дерева коду Хаффмена будується за наступним алгоритмом.

1. Формується прямокутна матриця із кількістю стовпчиків N та кількістю рядків M, де N – кількість вершин у дереві, а M – кількість ребер.

2. Якщо існує зв'язок між вершинами із номерами i та k через ребро із номером l, тоді у рядку l у стовпчиках i та k записується значення ваги ребра, 0 або 1. Якщо ребро l безпосередньо не пов'язано із вершиною j, у стовпчик j рядка l ставиться значення –1.

За сформованою матрицею ідентифікації можна знайти розряди кодів Хаффмена для символів $z_1 - z_n$ за наступним ітераційним алгоритмом.

1. У стовпчику i матриці ідентифікації I шукаємо елемент матриці, який не дорівнює –1. Ця операція проводиться для всіх значень $i \le n$.

2. Вписуємо значення знайденого елемента матриці $I(l,i)$ в перший розряд коду символу z_i.

3. У визначеному рядку l, для якого $I(l,i) \neq -1$, шукаємо інший елемент матриці ідентифікації із номером k, для якого $I(l,i) = I(l,k)$, $k > n$. Оскільки всі ребра пов'язані лише із двома вершинами дерева, зрозуміло, що такий елемент буде лише один.

4. У стовпчику k шукаємо інший елемент матриці ідентифікації $I(m,k)$, який не дорівнює –1.

5.У рядку m шукаємо інший елемент матриці ідентифікації $I(m,r)$ такий, що $I(m,k) = I(m,r)$ за умови $r > n$.

6. Якщо $r > k$, вважаємо, що $k = r$, вписуємо значення знайденого елемента матриці ідентифікації $I(m,k)$ до наступного розряду коду символу z_i та переходимо до пункту 4.

7. Якщо $r < k$, збільшуємо m на одиницю та переходимо до пункту 4.

8. Якщо $k = N$ – завершення формування коду символу z_i та збільшення значення i на 1.

9. Якщо $i < n$ – перехід до пункту 1 алгоритму, у противному випадку – закінчення формування коду.

Блок – схема описаного алгоритму наведена на рис. 1.61.

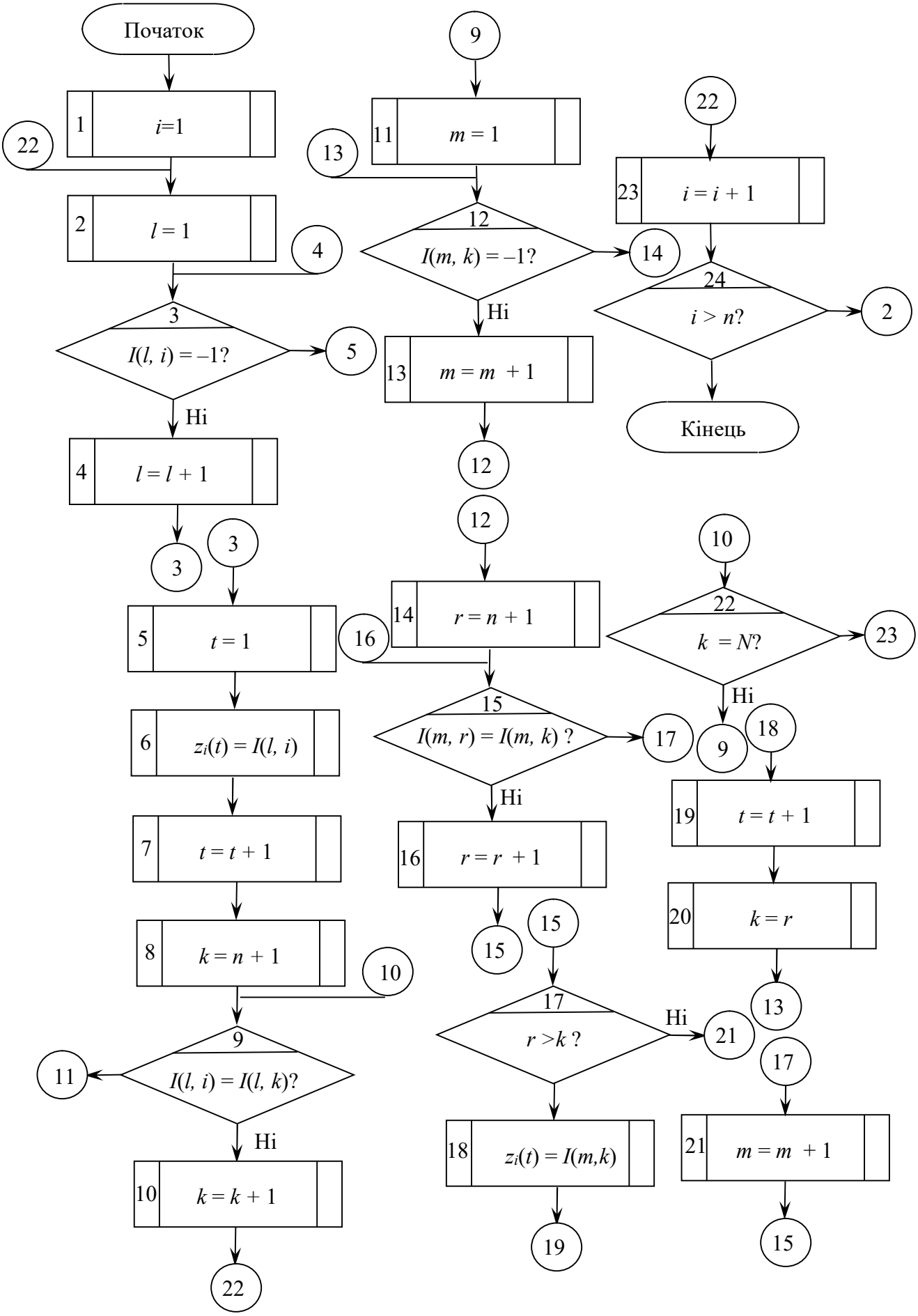


Рис. 1.61 Алгоритм формування коду Хаффмена через матрицю ідентифікації

Особливо важливе значення мають пункти алгоритму 4 – 7, які відображені блоками розгалуження на блок-схемі алгоритму, наведеній на рис. 1.61. Річ у тому, що необхідно шукати єдиний шлях від кінцевої вершини до кореня дерева, а не шляхи між кінцевими вершинами. Припустимо, що через зміну значень номера рядка m та номера стовпця k знайдено шлях від вершини, яка відповідає символу i до вершини, яка відповідає j. Проте такий шлях не відповідає умовам побудови коду Хаффмена та має бути відкинутим. У цьому разі необхідно збільшити значення m та пересуватися за рядками далі, поки не буде знайдений шлях до вершини. Тому розглянутий вище спосіб нумерації вершин та ребер дерева від кінцевих вершин до кореня дозволяє відкидати хибні шляхи між вершинами у разі зменшення значення k. Критичною умовою, за якою подальший пошук припиняється, є умова $k < n$.

За визначених умов пошук шляхів до вершини ієрархічного дерева коду Хаффмена через матрицю ідентифікації дещо нагадує піднімання альпіністів на високі гори та підкорення неприступних гірських вершин. Тут правило одне: ні в якому разі вниз, шлях лише один – вгору!

Пошук коду символу вважається закінченим, якщо $k = N$, тобто, на даній ітерації шлях до вершини дерева цілком завершений. Значення біту 0 та 1 вписуються у ланцюжок бітів символу z_i після остаточної перевірки коректності знайденого рядка за умовою $k > n$. Для підрахунку кількості символів використовується змінна t.

Розглянемо приклад формування кодів Хаффмена через матрицю ідентифікації.

Приклад 1.22. З використанням матриці ідентифікації побудувати коди Хаффмена для символів абетки, заданої у прикладі 1.21.

Для розв’язування цієї задачі насамперед необхідно зробити необхідні перетворення ієрархічного дерева коду Хаффмена, наведеного на рис. 1.56, пронумерував всі його ребра та вершини та видаливши значення сумарних ймовірностей появи символів. Таке ієрархічне дерево, еквівалентне наведеному на рис. 1.56, зображене на рис. 1.62. На рис. 1.62 ребра дерева позначені символом L.

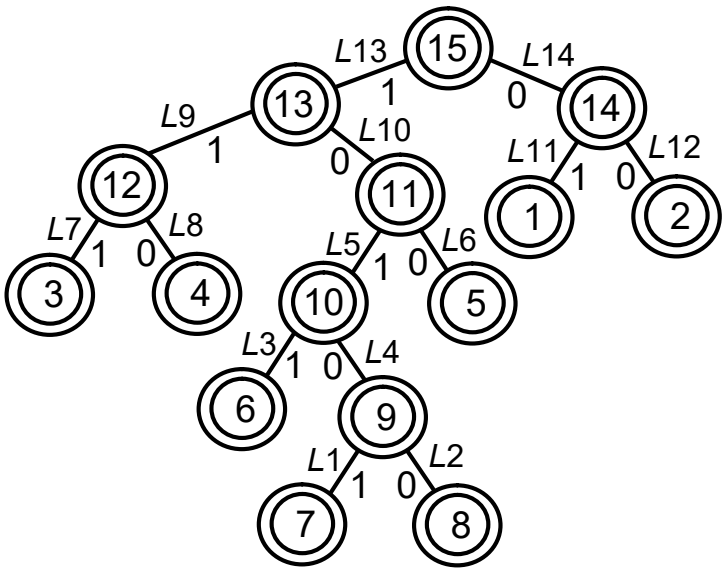


Рис. 1.62 Ієрархічне дерево коду Хаффмена, еквівалентне наведеному на рис. 1.56, з введеною нумерацією ребер та вершин

Матриця ідентифікації для дерева, структура якого зображена на рис. 1.62, представлена на рис. 1.63. Розглянемо, як ця матриця отримана через аналіз окремих її рядків.

Вершини

Ребра	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	-1	-1	-1	-1	-1	-1	1	-1	1	-1	-1	-1	-1	-1	-1
2	-1	-1	-1	-1	-1	-1	-1	0	0	-1	-1	-1	-1	-1	-1
3	-1	-1	-1	-1	-1	1	-1	-1	-1	1	-1	-1	-1	-1	-1
4	-1	-1	-1	-1	-1	-1	-1	-1	0	0	-1	-1	-1	-1	-1
5	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	-1	-1	-1	-1
6	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	0	-1	-1	-1	-1
7	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1
8	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1
9	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	-1	-1
10	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	0	-1	-1
11	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1
12	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1
13	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	1
14	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0

Рис. 1.63 Матриця ідентифікації для ієрархічного дерева, наведеного на рис. 1.62, з ілюстрацією алгоритму пошуку бітів для першого, другого, третього, п’ятого та сьомого символів

Розглянемо спосіб формування матриці ідентифікації, аналізуючи окремі її рядки. Наприклад, перший рядок відповідає ребру $L1$. Із структури дерева, наведеної на рис. 1.62, видно що ребро $L1$ з'єднує вершини 7 та 9 та має вагу 1. Тому у сьому та дев'яту колонку першого рядка вписуємо значення 1, а інші елементи першого рядка мають значення –1. Рядок 13 відповідає ребру $L13$, яке з'єднує вершини 13 та 15 та має вагу 1, тому у тринадцятому та п'ятнадцятому стовпчиках тринадцятого рядка матриці ідентифікації записані одиниці, а інші елементи цього рядка дорівнюють –1.

Спосіб формування кодів символів через пошук шляху від кінцевої вершини, яка відповідає заданому символу, до кореня дерева, також наочно показаний на рис. 1.63 для першого, другого, третього, п'ятого та сьомого символів абетки. Розглянемо, яким чином знайдені через матрицю ідентифікації шляхи пов'язані із пошуком розрядів кодів символів. Наприклад, для символу z_1 у першому стовпчику матриці ідентифікації стоїть 1 у рядку 11, а інші елементи цього стовпчика дорівнюють –1. Тому ставимо у перший розряд символу z_1 значення 1 та шукаємо, у якому іншому стовпчику одинадцятого рядка матриці також стоїть 1. Це стовпчик 14. Тепер необхідно шукати, які інші елементи 14 стовпчика не дорівнюють –1. Такі елементи є у рядках 12 та 14, і вони дорівнюють 0. Проте, якщо проаналізувати рядок 12, зрозуміло, що це відгалуження шляху є хибним і не веде до вершини дерева. Дійсно, інший елемент дванадцятого рядка, рівний 0, стоїть у другому стовпчику, і на цьому шлях обривається. Оскільки нове значення стовпчика є меншим, ніж загальна кількість символів n, вважаємо цей шлях хибним. Аналізуючи чотирнадцятий рядок матриці, бачимо що 0 стоїть у п'ятнадцятому стовпчику, а це – корінь дерева. Тобто, вписуємо 0 у другий розряд коду. Тому код символу z_1 становить 01, що відповідає результату, отриманому у прикладі 1.21. Проаналізуємо тепер за матрицею ідентифікації шлях від сьомої вершини. У першому рядку сьомої колонки матриці ідентифікації стоїть значення 1. Тому вписуємо 1 у перший розряд коду символу z_7 та шукаємо, у якому іншому стовпчику першого рядка стоїть одиниця. Це стовпчик 9. У дев'ятому стовпчику стоять нулі у другому та

четвертому рядках, проте легко зрозуміти, що шлях через другий рядок є хибним. Дійсно, інший 0 стоїть у другому рядку у стовпчику 8, і на цьому шлях обривається. Оскільки значення стовпчика є меншим за загальну кількість символів n, цю ділянку шляху, згідно із алгоритмом аналізу матриці ідентифікації, необхідно відхилити. Аналізуючи четвертий рядок, знаходимо 0 у десятому стовпчику, вважаємо цей шлях істинним та вписуємо 0 у другий розряд коду символу z_7. Тепер необхідно перейти до аналізу стовпчика 10. Тут знаходимо 1 у п'ятому рядку, а інші елементи цього стовпчика дорівнюють –1. Тому ставимо 1 у третій розряд коду символу z_7 та шукаємо 1 в інших стовпчиках рядка 5. Це стовпчик 11. В одинадцятому стовпчику 2 інших елементи дорівнюють 0, це елементи шостого та десятого рядка. Проте, аналізуючи шлях через шостий рядок, бачимо, що інший нульовий елемент цього рядка стоїть у п'ятій колонці. Оскільки значення стовпчика є меншим за загальну кількість символів n, у даному разі шлях через шостий рядок на веде до кореня дерева і є хибним. З іншого боку, у десятому рядку 0 стоїть у колонці 13. Вважаючи цю єдину можливу ділянку шляху істиною, ставимо 0 у четвертий розряд коду символу z_7 та аналізуємо колонку 13. У цій колонці 1 стоїть у 13 рядку. Інша одиниця тринадцятого рядка стоїть у колонці 15, а це – корінь дерева. Тому вписуємо 1 в останній, п'ятий розряд коду символу z_7 і остаточно вважаємо, що код символу z_7 складає 10101, що відповідає результату, отриманому у прикладі 1.21.

Найпростішим способом формування кодів символів через матрицю ідентифікації є читання інформаційних символів на лініях, які відповідають горизонтальним лініям шляху. Повертаючись до аналогії з альпіністами, можна сказати, що, формуючи код символу, ми поступово підіймаємося крутими сходинками, шукаючи скарби одиниць та нулів серед терен мінус одиниць у стовпчиках матриці, і підіймаємося у цих пошуках все вище і вище, лише з однією метою – досягти вершини. Цей метод аналізу шляхів від вузлів, який відповідає символам із заданим номером, до вершини дерева, також наочно показаний на рис. 1.63. На цьому рисунку також наочно видно, що шляхи від усіх кінцевих вершин, які відповідають символам, збігаються

або до тринадцятого, або до чотирнадцятого рядка п’ятнадцятої колонки. Це пов’язано із тим, що, згідно із рис. 1.62, вершина 15 відповідає кореню дерева, а з’єднані із нею 2 ребра: це ребро $L13$ із вагою 1 та ребро $L14$ із вагою 0.

Як видно із наведених міркувань, аналіз матриці ідентифікації дійсно є ефективним способом пошуку кодових послідовностей за алгоритмом коду Хаффмена, а алгоритм, наведений на рис. 1.61, формалізований у числовій формі та може бути легко реалізованим у сучасних комп’ютерних програмах. Єдиний, проте суттєвий недолік цього алгоритму полягає у тому, що для побудови матриці ідентифікації необхідно знати структуру ієрархічного дерева коду Хаффмена, яка тісно пов’язана із ймовірностями появи символів абетки. Тобто, алгоритм аналізу матриці ідентифікації пристосований лише для формування кодів символів на основі відомої структури ієрархічного дерева, а методика побудови самого дерева, яка є найбільш цікавою та важливою для реалізації в універсальних програмних засобах, у цьому алгоритмі не передбачена. Оскільки ієрархічне дерево коду Хаффмена будується через сумування ймовірностей символів абетки, будемо формувати алгоритм синтезу дерева саме через аналіз послідовностей елементів, які сумуються, та через їхнє подальше упорядкування. Зрозуміло, що саме за допомогою такого підходу можна отримати повну інформацію про структуру ієрархічного дерева коду Хаффмена та алгоритмізувати процедуру його побудови.

Тому надалі будемо використовувати для побудови коду Хаффмена дещо інший підхід, оснований на формуванні функціональних зв’язків між матрицями різної розмірності. Саме такий підхід дозволяє через відповідні матричні перетворення перейти від заданого вектору розподілу ймовірностей символів до значень розрядів коду Хаффмена, які відповідають зваженим гілкам орієнтованого дерева або шляхам до кореня дерева у матриці ідентифікації [13, 14].

Починати необхідно з формування на основі впорядкованого вектора ймовірностей появи елементів матриці сум ймовірностей, аналогічній

наведеним на рис. 1.61 та 1.55. Якщо елемент із малим значенням імовірності вже включений у суму, відповідні елементи матриці сум будемо вважати рівними нулю, що у значній мірі спрощує операцію сумування. У матриці сум послідовно відображені всі ітерації сумування елементів, тобто, формується вона рекурентно через початковий вектор ймовірностей появи символів. Перший рядок матриці містить впорядковані імовірності всіх елементів, а останній рядок – одиницю в першій колонці та 0 в інших колонках. Наприклад, для абетки символів, яка розглядалась у прикладі 1.21, матриця сум має розмірність 8x8, вигляд цієї матриці показаний на рис. 1.64.

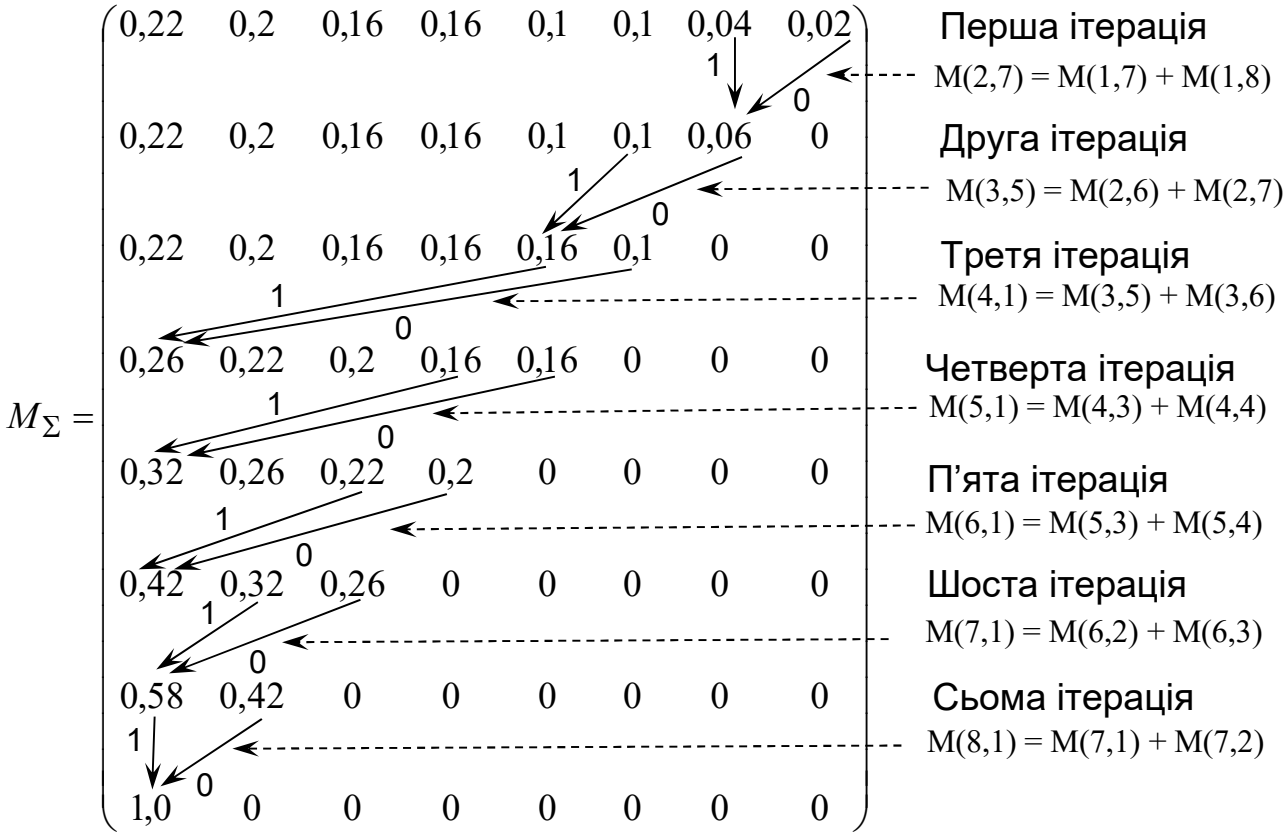


Рис. 1.64. Матриця сум ієрархічного дерева коду Хаффмена для прикладу 1.21 та спосіб її формування

Проблема полягає у тому, що під час сумування ймовірностей змінюється порядок елементів у матриці сум, і додатково необхідно чітко знати, які елементи входять у суми. Проте, як видно з рис. 1.64, ця інформація вже закладена у сумах, які формуються, через номера елементів у

структурі початкового вектора. Наприклад, на другій ітерації сума шостого та сьомого елементів записується не на шосту, а на п’яту позицію, відповідно п’ятий елемент початкового вектора зміщується праворуч і переходить на шосту позицію. Інше суттєве пересування елементів початкового вектор має місце на третій ітерації, коли сума п’ятого та шостого елементів переходить на першу позицію. Відповідно перший елемент стає другим, третій – четвертим, а четвертий – п’ятим. Тобто, позиції всіх елементів початкового вектора, які ще не сумувалися, стають на одиницю більшими. Також треба мати на увазі, що насправді шостий елемент на третій ітерації вже є сумою шостого, сьомого та восьмого елементів, тобто, враховувати кроки попередніх ітерацій. На четвертій ітерації сума третього та четвертого елементів переміщується на першу позицію, тому порядок елементів буде такий: третій елемент, до якого ми додали четвертий, за ним шостий, який вже являє собою суму п’ятого, шостого, сьомого та восьмого елементів початкового вектора, а за ним – перший та другий елементи, які мають найбільше значення імовірності появи символів і тому ще не сумувалися. П’ята ітерація є простішою. На ній перший елемент початкового вектора сумується з другим і ця сума переходить на першу позицію, оскільки вона перевищує значення третього та шостого елементів. Тому на шостій ітерації третій елемент сумується з шостим, а на останній, сьомій ітерації – третій з першим.

Сутність таких пересувань елементів вектора ймовірностей появи символів в процесі їхнього сумування наочно показана на рис. 1.65.

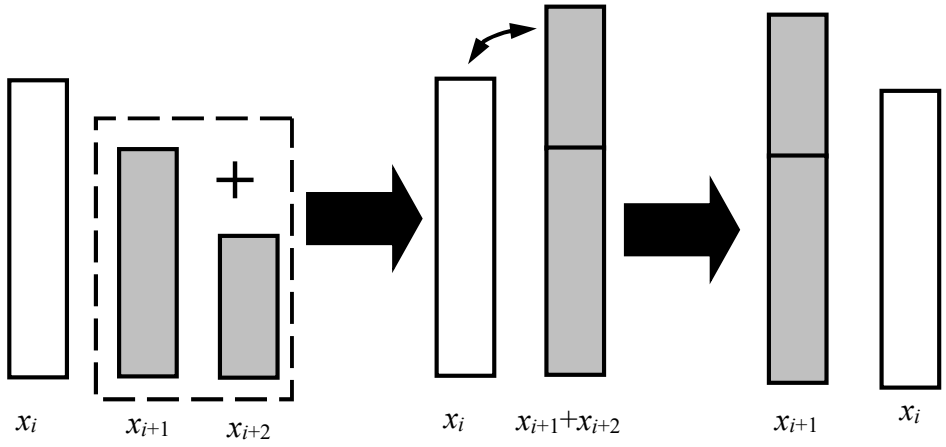


Рис. 1.65 Наочна ілюстрація зміни порядку елементів початкового вектора під час формування матриці сум ймовірностей появи елементів

Але для коректного відображення ієрархічної структури зваженого дерева коду Хаффмена також необхідно враховувати вагу елементів, які сумуються. Це легко зробити наступним чином. Коли сумуються два сусідні елементи із матриці сум, елементу, який стоїть праворуч, присвоюється значення 0, а лівому елементу – значення 1. Відповідна процедура присвоєння ваги елементам, які сумуються, також наочно показана на рис. 1.64.

З урахуванням наведених вище міркувань із матриці сум формується матриця номерів елементів, які сумуються. В цій матриці вказуються лише номери двох сусідніх елементів, які сумуються на кожній ітерації, при цьому у першому рядку стоїть елемент із вагою 1, а у другому – елемент із вагою 0. Для прикладу, який розглядається, така матриця має наступний вигляд:

$$S_{el} = \begin{pmatrix} 7 & 6 & 6 & 3 & 1 & 3 & 3 \\ & & & & & & \\ 8 & 7 & 5 & 4 & 2 & 6 & 1 \end{pmatrix}. \qquad (1.98)$$

Головну ідею формування матриці номерів елементів, які сумуються, також легко зрозуміти із рис. 1.65. Незважаючи на те, що елементи x_i та x_{i+1} змінили свої позиції після сортування, не можна вважати, що елемент з номером i+1 отримав номер i. Для коректного відображення структури дерева обов’язково необхідно зберігати нумерацію елементів початкового вектора. Наприклад, якщо в матриці сум, наведеній на рис. 1.59, шостий елемент переходить на першу позицію, він залишається шостим і ні в якому разі не становиться першим. Лише за таких умов сортування сум ймовірностей появи елементів можна безпомилково відобразити складну структуру ієрархічного дерева коду Хаффмена через матрицю номерів елементів, які сумуються.

Виникає питання. Чи не втрачаємо ми за таких умов інформацію про елементи, які вже просумували? Знову звернемося до рис. 1.65. Дійсно, коли ми сумуємо елементи x_{i+1} та x_{i+2}, ми даємо новому елементу номер i+1, а інформація про елемент x_{i+2} нібито втрачається. Але це не зовсім так, оскільки зв’язок між елементами x_{i+1} та x_{i+2} буде зафіксований у матриці

номерів елементів, які сумуються S_{el}. Тому інформацію про цей зв'язок можна легко поновити, якщо проаналізувати всі попередні стовпчики матриці S_{el}. Може також виникнути питання, із яких міркувань новому сформованому елементу ми надаємо номер i+1, а не i+2? Тут слід відзначити, що оскільки зв'язок між елементами з номерами i+1 та i+2 у матриці S_{el} вже встановлений, номер нового елементу не має особливо важливого значення, проте зазвичай зручніше зберігати номер більшого елементу із матриці сум, а меншого – відкидати.

Проведемо аналіз визначеної матриці S_{el}, та з'ясуємо, чи коректно вона відображає структуру ієрархічного дерева коду Хаффмена. Зрозуміло, що якщо номера двох елементів стоять у відповідному стовпчику матриці номерів елементів, які сумуються, необхідно у поточний розряд елементу із верхнім номером вписати 1, а у розряд елементу із нижнім номером – 0. Після цього необхідно шукати у попередніх стовпчиках матриці S_{el} всі елементи, які пов'язані із поточними, та вписувати в їхні розряди одиниці та нулі, залежно від положення елементу в першому або у другому рядку на поточній ітерації. Положення знайдених елементів у стовпчиках, сформованих на попередніх ітераціях, під час виконання цієї процедури не враховується.

Суттєва проблема полягає у тому, що треба шукати не лише безпосередні, але й більш глибокі ієрархічні зв'язки. Тобто, якщо ми з'ясували, що елемент n пов'язаний із елементом k, надалі у попередніх стовпчиках слід шукати як елемент n, так і елемент k. Це у деякій мірі ускладнює процес пошуку, оскільки треба пам'ятати всі ланцюжки зв'язаних елементів, адже саме ці зв'язки безпосередньо відображають структуру дерева коду Хаффмена. Наприклад, якщо елемент 6 у визначеній матриці S_{el} пов'язаний із елементом 7, а у попередньому стовпчику стоїть зв'язок з елемента 7 на елемент 8, тоді елемент 6 зв'язаний не лише з сьомим, а і з восьмим елементом. Розглянемо, наприклад, третій стовпчик матриці S_{el}, в якому у верхньому рядку стоїть елемент з номером 6, а у нижньому – елемент з номером 5. Спочатку необхідно поставити 1 у відповідний розряд

коду шостого елементу та 0 у відповідний розряд коду п’ятого. Оскільки цифри 5 у попередніх стовпчиках немає, 0 в коди інших елементів на цей ітерації не вписуємо. Проте у другому стовпчику зверху стоїть цифра 6, і вона пов’язана із цифрою 7, яка стоїть внизу. Це означає, що у відповідний розряд сьомого елементу необхідно вписати значення 1, оскільки цифра 6 у третьому стовпчику стоїть на верхній позиції. Подальший аналіз показує, що цифри 6 у першому стовпчику немає, проте у верхньому рядку першого стовпчика стоїть цифра 7. Це означає, що необхідно вписати 1 в елемент, пов’язаний за першим стовпчиком із елементом номер 7, тобто, у восьмий елемент.

Зрозуміло, що описаний спосіб аналізу ієрархічної структури дерева коду Хаффмена через матрицю номерів елементів, які сумуються, тісно пов’язаний із алгоритмами сортування елементів матриць у дискретній математиці, які були описані у підрозділі 2.4.4 другої частини посібника [48].

Згідно із наведеними вище теоретичним міркуваннями розглянемо тепер спосіб формування кодових послідовностей для абетки символів, заданої у прикладі 1.21.

Приклад 1.23. З використанням матриці номерів елементів, які сумуються, сформувати код Хаффмена для абетки символів, заданої у прикладі 1.21.

Матриця S_{el} номерів елементів, які сумуються, для абетки символів, що розглядається, задана співвідношенням (1.98). Почнемо аналіз цієї матриці з першого стовпчика. Цей перший крок є найпростішим, оскільки зв’язки між елементами вектора ймовірностей появи символів на цьому етапі ще відсутні. У верхньому рядку матриці номерів елементів стоїть цифра 7, а у нижньому – цифра 8, тобто, просто ставимо 1 у перший розряд сьомого елементу та 0 у перший розряд восьмого. Відповідна процедура формування кодів символів на першій ітерації наочно показана на рис. 1.66. На другій ітерації насамперед необхідно вписати 1 в шостий символ та 0 в сьомий, проте тут слід мати на увазі дві суттєвих особливості формування кодів символів. Ці особливості пов’язані з тим, що на попередній, першій ітерації, вже був

використаний сьомий елемент. Тому насамперед слід зазначити, що цифра 0 буде стояти не в першому, а у другому біті сьомого символу. А по друге, через першу колонку матриці номерів елементів, які сумуються, елемент 7 пов'язаний з елементом 8, у зв'язку з чим необхідно також поставити 0 у другий розряд коду восьмого символу. На рис. 1.66 наочно показані всі операції щодо занесення символів, які виконуються на другій та на наступних ітераціях, а також зв'язки між елементами матриці, що відповідають цим операціям. Біти, які у даній колонці не вписується до коду символу, а передається за ланцюжком до наступної колонки, на рис. 1.66 обведені квадратиком, а колонки матриці, які відповідають даній ітерації, обведені прямокутниками із штриховою лінією.

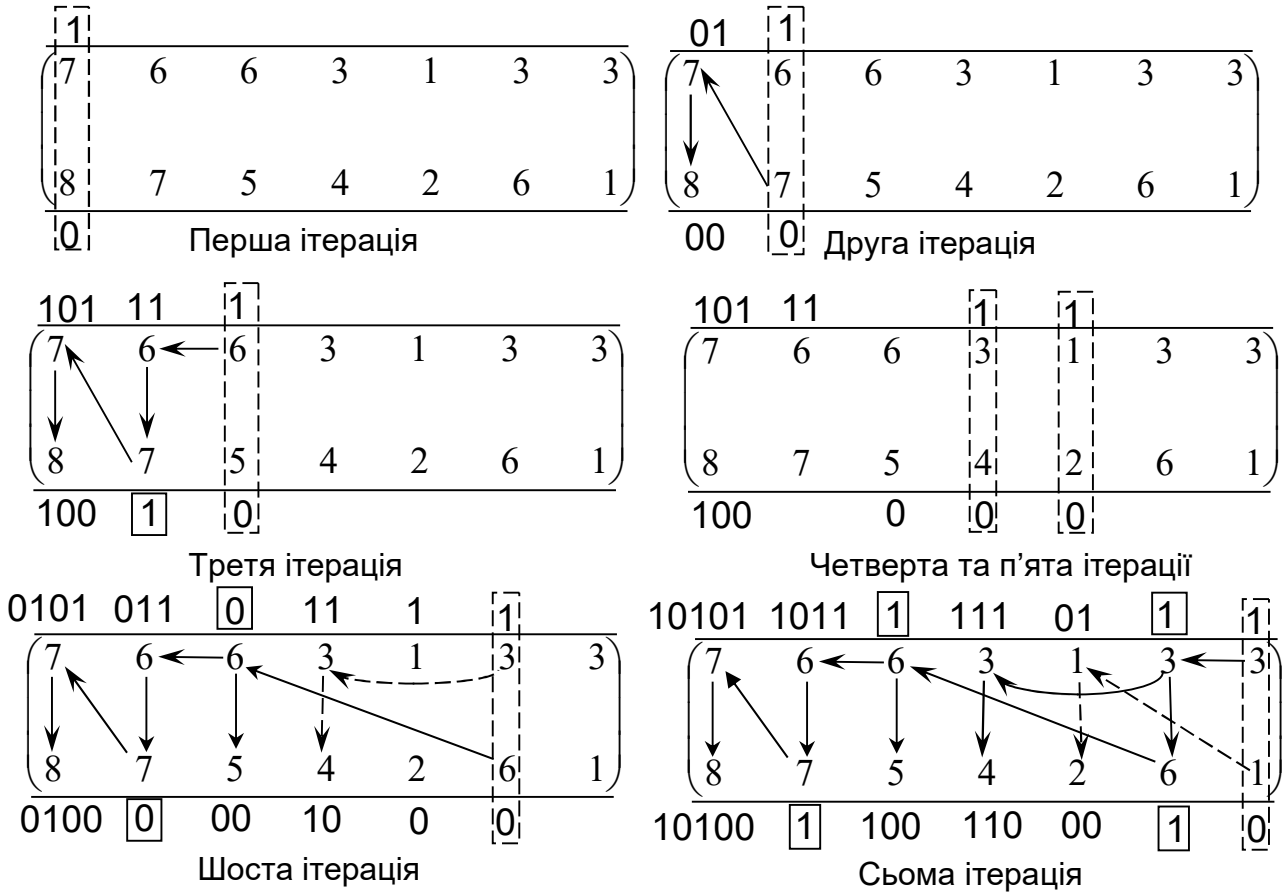


Рис. 1.66 Наочна ілюстрація ітераційного процесу формування бітових послідовностей коду Хаффмена через аналіз стовпчиків матриці елементів, які сумуються

Розглянемо тепер послідовно процес формування кодів символів на інших ітераціях. Із рис. 1.66 видно, що в третій колонці матриці номерів елементів, які сумуються, зверху стоїть цифра 6, а знизу – цифра 5. Цифра 5 у попередніх колонках ще не зустрічалась. Це означає, що код п’ятого символу на третій ітерації тільки починає формуватися, тому ставимо 0 у перший розряд п’ятого символу. Цифра 6 стоїть також зверху у другій колонці, тому ставимо 1 в другий розряд коду шостого символу, і на цій ітерації він становить 11. Крім цього, цифра 6 через другу колонку пов’язана із цифрою 7, а цифра 7 – через першу колонку із цифрою 8. Тому на третій ітерації необхідно дописати 1 у треті розряди кодів сьомого та восьмого символів. Після виконання цієї операції код сьомого символу складає 101, а код восьмого – 100.

Четверта та п’ята ітерації є вкрай простими, вони багато в чому схожі на першу. На цих ітераціях починають формуватися коди четвертого, третього, другого та першого символів. У четвертій колонці матриці номерів елементів, які сумуються, зверху стоїть цифра 3, а внизу – цифра 4. Відповідно, записуємо 0 у перший розряд четвертого символу та 1 у перший розряд третього символу. У п’ятій колонці матриці номерів елементів, які сумуються, зверху стоїть цифра 1, а внизу – цифра 2. Відповідно, записуємо 0 у перший розряд другого символу та 1 у перший розряд першого символу.

Найбільш складними є останні, шоста та сьома ітерації, які відображують розгалужену структуру всього ієрархічного дерева коду Хаффмена. У шостій колонці матриці номерів елементів, які сумуються, зверху стоїть цифра 3, а внизу – цифра 6. Насамперед це означає, що необхідно вписати 1 до коду третього елементу та 0 до коду шостого елементу. Проте цифра 3 стоїть також у третій колонці матриці, і через цю колонку третій елемент пов’язаний із четвертим. Тому на даній ітерації код третього символу становить 11, а код четвертого символу – 10. Цифра 6 стоїть також у третій та у другій колонці матриці номерів елементів, які

сумуються. Це означає що 0 записується не лише у третій розряд шостого елементу, й у другий розряд п’ятого елементу. Тоді код шостого символу на шостій ітерації становить 011, а код п’ятого символу – 00. Проте на цьому процедура записи нулів не завершується, оскільки через другу колонку шостий елемент пов’язаний із сьомим, а через першу – сьомий із восьмим. Тому необхідно дописати нулі в отримані на попередніх ітераціях коди сьомого та восьмого елементів. Як видно із рис. 1.66, в результаті виконання цієї операції код сьомого елементу становить 0101, а восьмого – 0100.

В останній, сьомій колонці матриці номерів елементів, які сумуються, зверху стоїть цифра 3, а внизу – цифра 1, тому дописуємо 1 до коду третього символу та 0 до коду першого. Як видно із рис. 1.66, в результаті виконання цієї операції код першого елементу складає $z_1 =$ 01, а код третього – $z_3 = 111$.

Проте цифра 1 стоїть також у п’ятій колонці матриці, через цю колонку перший елемент зв’язаний із другим. Тому остаточно сформований код другого елемента становить $z_2 = 00$. Це також наочно видно на рис. 1.66.

Тепер аналізуємо попередні зв’язки для третього символу, оскільки у верхньому рядку сьомої колонки стоїть цифра 3. Третій символ коду зв’язаний через другу шосту колонку матриці із шостим, а через четверту колонку – із четвертим елементом. Тому відразу дописуємо 1 у коди цих символів, і остаточно отримуємо: $z_4 = 110$, $z_6 = 1011$. Але тепер необхідно прослідити всі зв’язки для шостого елементу. Через третю колонку шостий елемент пов’язаний із п’ятим, тому ставимо в останній, третій розряд п’ятого символу 1. Як видно на рис. 1.66, остаточний код п’ятого символу – $z_5 = 100$. Проте, оскільки ієрархічне дерево є дуже розгалуженим, четверта ітерація ще не закінчена. Слід мати на увазі, що шостий елемент через другу колонку зв’язаний із сьомим, а сьомий через першу колонку – з восьмим елементом, у зв’язку з чим на сьомій ітерації дописуємо 1 в коди цих елементів. Остаточно маємо: $z_7 = 10101$, $z_8 = 10100$. Тобто, отримані бітові послідовності для кодів символів $z_1 – z_8$ повністю співпадають із результатом, отриманим у прикладі 1.21.

Недоліком алгоритму формування коду Хаффмена, основаного на аналізі матриці номерів елементів, які сумуються, є постійна необхідність пошуку зв’язків між елементами за рядками матриці, що вкрай ускладнює його програмну реалізацію. Наочність цього методу, проілюстрованого на рис. 1.66, дозволяє побачити і цей суттєвий недолік. Дійсно, постійний пошук зв’язків між елементами, особливо на останніх ітераціях, пов’язаний із необхідністю аналізу складних деревоподібних ієрархічних структур. Образно кажучи, якщо на перших ітераціях аналізу матриці номерів елементів, які сумуються, заблукати в трьох соснах, тобто, забути про зв’язок між будь-якими із трьох або чотирьох елементів, досить важко, то на останніх ітераціях серед лісу розгалужених дерев дійсно легко заблукати. Як вийти із цієї ситуації? Вихід тут дуже простий, адже всі головні зв’язки між елементами вже були встановлені на перших ітераціях. У цьому разі ми маємо поступати не як нерозумні діти, які блукають по лісу, не знаючи шляху, а як досвідчені туристи. А вони завжди беруть із собою у похід карту та орієнтуються за нею, щоб запам’ятати, яким шляхом вони пройшли. До того ж слід мати на увазі, що алгоритми пошуку є дуже ресурсоємними і за цією причиною у комп’ютерних програмах не варто використовувати їх багаторазово [59]. Складність цих алгоритмів легко зрозуміти і через аналіз простих життєвих ситуацій. Дійсно, якщо студент збирається писати курсову роботу і починає спочатку шукати ручку, потім зошит, потім необхідні книги, які він невідомо куди поклав, справи у нього швидко не підуть. Швидше із поставленою задачею впориться охайний студент, який завжди пам’ятає, де що в нього лежить. Ця особливість алгоритмів пошуку окремо розглядалась у підрозділі 1.4 другої частини посібника [48]. Подібну ситуацію довгого пошуку дуже яскраво описав відомий російський поет Владислав Ходасевич в одному із своїх сатиричних віршів:

Перешагни, перескочи,
Перелети, пере- что хочешь –
Но вырвись: камнем из пращи,

Звездой, сорвавшейся в ночи...

Сам потерял – теперь ищи...

Бог знает, что себе бормочешь,

Ища пенсне или ключи.

Тому поступимо в даному випадку не як малі нерозумні діти, а як досвідчені туристи. Будемо ретельно відмічати шлях по карті, щоб не заблукати у лісі зв'язаних елементів на останніх ітераціях формування коду.

Такою картою в даному випадку для нас буде матриця зв'язків між елементами, які кодуються. Повернемося до рис. 1.65. Оскільки ми сумуємо елемент x_{i+1} з елементом x_{i+2}, необхідно у матриці зв'язків між елементами показати, що елемент x_{i+2} тепер зв'язаний з елементом x_{i+1}. Після цього про існування елементу x_{i+2} можна нібито забути, але лише у тому сенсі, що у наступних стовпчиках матриці елементів, які сумуються, він нам не буде зустрічатися. Але про цей елемент обов'язково необхідно пам'ятати під час заповнення бітів кодів символів, оскільки якщо у наступних стовпчиках матриці елементів, які сумуються, у першому або другому рядку буде стояти елемент x_{i+1}, відповідний біт, 0 або 1, слід також вписувати у код елементу x_{i+2}.

Повернемося до рис. 1.66 та розглянемо восьмий елемент. Він вже на першій ітерації зв'язаний з сьомим елементом і тому у наступних стовпчиках число 8 не зустрічається. Проте оскільки у другому стовпчику в другому рядку стоїть число 7, цифра 0 на другій ітерації вписується не лише у другий розряд сьомого елементу, але й у другий розряд восьмого елементу. Відповідний ітераційний процес наочно показаний на рис. 1.61. Щоб надалі не аналізувати структуру всього ієрархічного дерева за матрицею номерів елементів, які сумуються, будемо записувати посилання з номера верхнього елементу цієї матиці на суміжний до нього номер нижнього елементу до іншої матриці, яку назвемо матрицею зв'язків між елементами. Цю матрицю будемо змінювати на кожній ітерації наступним чином. Якщо елемент x_{i+2} на попередній ітерації зв'язаний із елементом x_{i+1}, а на поточній ітерації елемент

x_{i+1} зв’язується з елементом x_i, необхідно зв’язати елемент x_i як з елементом x_{i+1}, так і з елементом x_{i+2}. Тобто, на кожній ітерації формування коду Хаффмена з’являються нові зв’язки між елементами, але ті, які були встановлені раніше, також зберігаються у матриці, що формується.

Будемо формувати матрицю зв’язків між елементами наступним чином. Спочатку запишемо квадратну матрицю розмірністю n x n та заповнимо її нулями. Якщо елемент із номером i зв’язується із елементом із номером k, тобто, у матриці елементів, які сумуються, число i на поточній ітерації стоїть зверху, а число k – внизу, необхідно число k вписати як перший елемент рядка i до матриці зв’язків між елементами коду Хаффмена. Це перший етап формування матриці зв’язків, але на наступних ітераціях виконання цієї операції може бути недостатнім. Дійсно, якщо вказати зв’язок з елементу i на елемент k, а з елементу k – на елемент j і так далі, аналіз структури дерева на останніх ітераціях суттєво не спрощується. Тому варто формувати матрицю зв’язків між елементами іншим способом. Якщо на попередніх ітераціях елемент k був зв’язаний з елементом j, а на поточній ітерації i слід зв’язати з елементом k, тоді слід вказати, що елемент i також за ієрархією дерева зв’язаний з елементом j. Тобто, у цьому разі до рядка i поряд із номером елемента k вписується номер елемента j, а із рядка k після запису до рядка i цифра j видаляється через її заміну на нуль. Дійсно, елементи j та k вже не будуть зустрічатись на наступних ітераціях в матриці номерів елементів, які сумуються, тому для спрощення алгоритму формування кодів символів у матриці зв’язків між елементами варто вказати лише один ієрархічний зв’язок з елементу i на елементи k та j. Порядок посилань у даному випадку не є суттєвим, але зрозуміло, що за умови описаного способу формування матриці перші числа рядка i будуть відповідати ближнім елементам, а останні – віддаленим. Наявність такого зв’язку означає, що якщо у наступних стовпчиках матриці елементів, які сумуються, буде стояти число i, відповідний біт, 0 або 1, слід також вписати до кодів елементів j та k. Таким чином, алгоритм формування матриці

зв’язків між елементами можна записати наступним чином.

1. Спочатку всі елементи матриці дорівнюють 0.

2. Якщо в матриці номерів елементів, які сумуються, зверху стоїть число *i*, а знизу – число *k*, необхідно спочатку записати 1 до коду елементу *i* та 0 до коду елементу *k*.

3. Аналізуючи матрицю зв’язків між елементами коду, переглядаємо рядки *i* та *k*. Якщо існують елементи, пов’язані із *i* – додаємо в коди цих елементів цифру 1, а якщо існують елементи, пов’язані із *k* – додаємо в коди цих елементів цифру 0.

4. В рядок *i* матриці зв’язків між елементами коду вписуємо число *k*.

5. Аналізуючи матрицю зв’язків між елементами коду, знову переглядаємо рядок *k* та переносимо всі елементи цього рядка до рядка *i*.

6. До всіх елементів рядка *k* записуємо нулі замість попередніх значень.

Тобто, спосіб формування матриці зв’язків між елементами відповідає відомому життєвому правилу: «Той, хто виконав свою справу, може бути вільним». Якщо відповідний елемент вже увійшов до загальної суми, можна про нього забути, залишивши лише спогади про нього у матриці зв’язків. Загальні правила такі.

1. Якщо через матрицю зв’язків знайдений зв’язок із поточного елементу *k*, який входить до суми, на елемент із відповідним номером *j*, тоді відповідний біт слід вписати не лише до елементу *k*, але й до елементу *j*.

2. Якщо на поточній ітерації елемент *k* у матриці елементів, які сумуються, стоїть внизу та зв’язується з елементом *i*, тоді спочатку до рядка *i* матриці зв’язаних елементів ставиться число *k*, а потім число *j* також переноситься до рядка *i*, а у рядку *k* замість нього ставиться 0.

3. Якщо за умов, визначених у пункті 2, елемент *k* через матрицю зв’язків пов’язаний із кількома елементами, наприклад, *j*, *l* та *m*, тоді номера всіх цих елементів переписуються до рядка *i*, а у рядку *k* замість цих елементів ставляться нулі.

Розглянемо особливості використання матриці зв’язків між елементами

для випадку формування коду Хаффмена за абеткою, заданою у прикладі 1.21.

Приклад 1.24. З використанням матриці зв'язків між елементами сформувати код Хаффмена за абеткою символів, заданою у прикладі 1.21.

Ітераційна процедура формування матриці зв'язків між елементами та кодів символів абетки наочно показана на рис. 1.61.

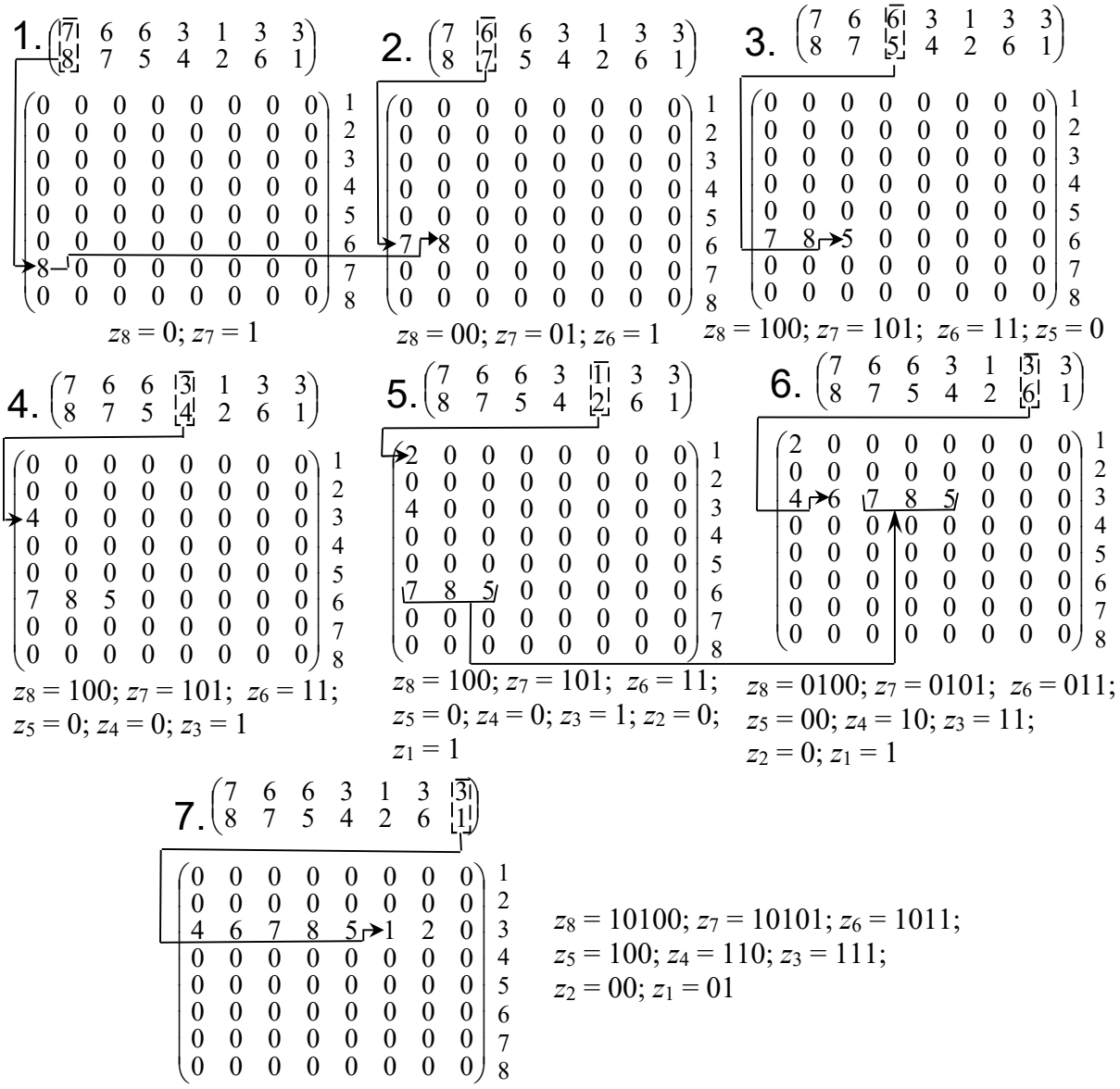


Рис. 1.67 Наочна ілюстрація принципу формування коду Хаффмена через аналіз матриці номерів елементів, які сумуються, та через формування матриці зв'язків між елементами для прикладу 1.24

Розберемо спосіб формування кодів символів через аналіз матриці

номерів елементів, які сумуються, та матриці зв’язків між елементами, окремо для кожної ітерації. У першій колонці матриці елементів, які сумуються, зверху стоїть число 7, а внизу – число 8. Це означає, що встановлюється зв’язок між сьомим та восьмим елементами і надалі елемент 8 у матриці елементів, які сумуються, використовуватися не буде. Тому необхідно поставити 1 в код сьомого елементу, 0 в код восьмого, а у першу колонку сьомого рядка матриці зв’язків між елементами ставиться число 8. Після виконання всіх цих дій переходимо до другої ітерації. У другій колонці матриці елементів, які сумуються, зверху стоїть число 6, а внизу – число 7. Тому на цій ітерації насамперед ставимо 1 у код шостого елементу, та 0 у другий розряд сьомого елементу, в перший розряд якого вже була записана одиниця на першій ітерації. Тепер, аналізуючи сьомий рядок матриці зв’язків між елементами, бачимо, що сьомий елемент зв’язаний із восьмим, тому необхідно також додати 0 до коду восьмого елементу, після чого слід перетворити матрицю зв’язків між елементами, відобразивши в ній зв’язки, які були встановлені на поточній ітерації. Спочатку опрацюємо ті зв’язки, які були встановлені на поточній ітерації через матрицю номерів елементів, які сумуються. Для цього необхідно вписати число 7 до першого стовпчика шостого рядка матриці зв’язків між елементами. Але аналізуючи сьомий рядок, бачимо, що в ньому стоїть число 8, і це означає, що сьомий елемент вже зв’язаний із восьмим. Необхідно тепер перенести цей зв’язок на шостий елемент, для цього переносимо число 8 у другий стовпчик шостої колонки, а у перший стовпчик сьомої колонку ставимо 0. Відповідні перетворення матриці зв’язків між елементами показані на рис. 1.67.

На третій ітерації, згідно із описаною стандартною послідовністю дій, спочатку аналізуємо третю колонку матриці елементів, які сумуються. Тут можна побачити, що елемент 5 пов’язаний із елементом 6. Насамперед це означає, що необхідно дописати 1 до коду шостого елементу та 0 до коду п’ятого елементу. Тепер, аналізуючи матрицю зв’язків між елементами, бачимо, що шостий елемент зв’язаний із сьомим та восьмим. Тому необхідно

поставити 1 також у коди сьомого та восьмого елементів.

Як і у прикладі 1.23, найпростішими є четверта та п’ята ітерації. На четвертій у першому рядку матриці номерів елементів, які сумуються, стоїть число 3, а у другому – число 4. Відповідно, спочатку вписуємо 1 до першого розряду коду третього елементу та 0 до першого розряду коду четвертого. Оскільки четвертий елемент необхідно зв’язати з третім, ставимо 4 у перший стовпчик третього рядка матриці зв’язків між елементами. Аналогічно, на п’ятій ітерації в матриці номерів елементів, які сумуються, зверху стоїть число 1, а внизу – число 2. Це означає, що необхідно вписати 1 до коду першого елементу та 0 до коду другого елементу, а також зафіксувати цей новий зв’язок у матриці зв’язків між елементами, поставивши число 2 у перший стовпчик першого рядка цієї матриці. На шостій ітерації, яка для даного прикладу є найбільш складною, за матрицею номерів елементів, які сумуються, шостий елемент пов’язаний із третім. Насамперед це означає, що необхідно вписати 1 та до коду третього та 0 до коду шостого елементу. Але, аналізуючи матрицю зв’язків між елементами, бачимо, що з третім елементом пов’язаний четвертий, а з шостим – сьомий, восьмий та п’ятий елементи. Тому необхідно також дописати 1 в код четвертого елемента та 0 в коди п’ятого, сьомого та восьмого. Після цього, фіксуючи у матриці зв’язків між елементами зв’язок між шостим елементом та третім, ставимо у третій рядок число 6 та дописуємо із шостого рядка числа 7, 8 та 5. Відповідні перетворення матриці зв’язків між елементами наочно показані на рис. 1.66.

На сьомій ітерації в матриці номерів елементів, які сумуються, зверху стоїть число 3, а знизу – число 1. Це означає, що до коду третього елементу необхідно дописати 1 та 0 до коду першого. Проте за матрицею зв’язків між елементами перший елемент пов’язаний із другим, а третій – із елементами 4, 6, 7, 8 та 5. Тому на цій, останній ітерації, необхідно дописати 0 у код другого елементу та 1 у коди четвертого, шостого, сьомого то восьмого елементів. Після цього у третій рядок матриці зв’язків між елементами дописуються числа 1 та 2 і ставиться 0 у перший стовпчик першого рядка цієї

матриці. Про закінченість формування коду свідчить дві важливі ознаки сформованої матриці зв’язків між елементами.

1. Всі рядки матриці, крім третього, заповнені нулями.

2. Третій рядок містить перелік всіх елементів, від першого до восьмого, крім третього. Це означає, що третій елемент через ієрархічні зв’язки пов’язаний з усіма іншими елементами, тобто, ми дійшли до вершини дерева.

З першого погляду здається, що алгоритм формування коду Хаффмена на основі матриці номерів елементів, які сумуються, та матриці зв’язків між елементами, є вкрай ускладненим і не зовсім наочним. Дійсно, для звичайної людини простіше побудувати код Хаффмена на основі ієрархічного дерева або на основі матриці ідентифікації. Ці алгоритми мають наочну графічну інтерпретацію, і тому людині працювати саме із такими засобами побудови кодів елементів значно легше. За цією причиною на початковому етапі розвитку обчислювальної техніки, у п’ятдесятих роках минулого століття, коли комп’ютерні засоби кодування інформації ще були слабо розвинені, фахівці в галузі електронної та обчислювальної техніки віддавали перевагу саме таким методам кодування. Проте суттєва перевага методу, основаного на аналізі матриці номерів елементів, які сумуються, та матриці зв’язків між цими елементами, полягає у тому, що він базується на класичних алгоритмах сортування, порівняння та пошуку, які легко формалізуються в регулярних мовах, розглянутих у підрозділі 7.3.1 другої частини посібника [50]. А із цього можна зробити важливий висновок, що такий алгоритм також легко реалізувати у комп’ютерній програмі або у роботі програмованих електронних схем з використанням стандартних засобів програмування. Іншою суттєвою перевагою цього алгоритму є його універсальність та орієнтованість на розв’язування складної інженерної задачі, пов’язаної із синтезом структури ієрархічного дерева коду Хаффмена за відомими значеннями ймовірностей появи символів. Це вигідно відрізняє його від іншого розглянутого алгоритму, пов’язаного із використанням матриці

ідентифікації, який, як було відмічено вище, орієнтований лише на аналіз, а не на синтез структури дерева.

У додатку Ж наведений код програми, в якій реалізований алгоритм формування коду Хаффмена через аналіз матриці номерів елементів, які сумуються через матрицю зв'язків між елементами вектора вхідної послідовності. Програма написана мовою програмування системи MatLab. Головними особливостями цієї програми, які визначають її структуру та функціональні можливості, є наступні.

1. Вхідними даними для формування коду Хаффмена є лише вектор значень ймовірностей символів.

2. Передбачена можливість введення не відсортованого масиву значень ймовірностей, процедура їхнього впорядкування через сортування за спаданням у програмі може бути виконана автоматично.

3. Передбачена можливість кодування послідовностей із двох або трьох символів.

4. Передбачена можливість декодування бітових послідовностей, сформованих на основі створеного коду. У разі необхідності виконання цієї операції вхідними даними для програми є два вектори: вектор значень ймовірностей символів та вектор одиниць та нулів, сформованої на основі відомого коду Хаффмена для заданої абетки.

5. Передбачений коректний вихід із програми з інформацією про помилку у разі невірно заданих даних. Наприклад, це можуть бути невірно задані значення ймовірностей або некоректна бітова послідовність для заданого вектора ймовірностей появи символів.

6. Програма має просту систему допомоги, в якій описані параметри її виклику та використані в ній імена змінних.

7. Оскільки програма є досить великою та складною для розуміння, у багатьох її рядках введені відповідні коментарі, які у деякій мірі полегшують розуміння написаного програмного коду.

Іншою важливою особливістю написаної програми є те, що вона

створена майже без використання засобів матричного та логічного програмування, які були описані у підрозділі 1.2.2, тобто, стандартними засобами структурного програмування. Незважаючи на те, що у навчальних посібниках [13, 14] була показана можливість ефективного використання засобів матричного програмування для виконання операцій пошуку та сортування елементів, аналіз таких складних матричних структур із великою кількістю зв'язків між елементами, як матриця номерів елементів, які сумуються, з використанням методики матричного програмування вкрай ускладнюється. Крім цього, як було показано у посібниках [13, 14], сьогодні недоліками такої методики програмування є необхідність використання масивів даних великого об'єму для запису проміжних результатів обчислень та необхідність викликів великої кількості процедур, зокрема викликів однієї процедури із іншої та рекурентних викликів. Тому у разі необхідності обробки великих масивів числової інформації використання методів матричного програмування може приводити до непомірного зростання часу проведення розрахунків, що у значній мірі знижує ефективність роботи програмних засобів. Але найголовнішим недоліком використання методів матричного програмування у складних задачах обробки числових даних є те, що велика кількість викликів зовнішніх функцій користувача, аналогічних функціям recvectbin, recmat або sortbooble [13, 14], на відповідному етапі вже не спрощує, а іноді навіть ускладнює розуміння написаного програмного коду. Тому для розв'язування таких складних задач програмування, як аналіз матриці номерів елементів, які сумуються, через матрицю зв'язків між елементами, сьогодні найбільш ефективним способом залишається використання стандартних методів структурного програмування. Несумнівними перевагами цих методів є наступні [13, 14].

1. Простота прямого доступу до великих масивів даних через їхню безпосередню індексацію.

2. Можливість простої та ефективної реалізації рекурентних обчислень через багаторазове присвоювання та переприсвоювання значень відповідних

змінних.

3. Можливість створення компактних масивів даних необхідного об’єму для збереження результатів обчислень та подальшого посилання на значення будь-яких елементів цих масивів під час проведення розрахунків на наступних ітераціях.

4. Значна економія часу проведення розрахунків та об’єму оперативної пам’яті за рахунок можливості своєчасного видалення проміжних результатів обчислень, які вже не потрібні для подальших розрахунків.

Проаналізуємо більш досконало структуру програми, наведеної у додатку Ж, та головні особливості її написання. Для реалізації описаного вище алгоритму аналізу матриці номерів елементів, які сумуються, через матрицю зв’язків між елементами, у програмі введені наступні змінні.

Вхідні змінні:

vh – вхідний вектор значень ймовірностей.

vc – вхідний вектор бітів символів для розв’язування задачі декодування.

ns – кількість символів абетки для формування коду Хаффмена. У програмі припускається формування коду для послідовностей із двох або трьох символів.

nop – кількість операцій, які необхідно виконати. Значення **nop=1** відповідає розв’язуванню лише задачі формування коду. У разі **nop=2** розв’язується також задача декодування для вхідного вектора **vc**.

Вихідні змінні:

HUFF – матриця-шаблон для запису кодів символів. Початкові значення всіх елементів цієї матриці дорівнюють 5, що дозволяє легко шукати останній записаний біт коду та вписувати у наступний розряд нове значення.

LSH – вихідний вектор, в якому вказується кількість бітів у коді кожного символу.

Eff – усереднений параметр ефективності коду Хаффмена, який

розраховується на основі співвідношення (1.95) наступним чином:

$$E_{\text{к}} = \frac{\bar{I}(X)}{v_{\text{б}}} = \bar{I}(X) \cdot \tau_{\text{б}} = \frac{H(X) \cdot \tau_{\text{б}}}{\bar{T}}, \qquad (1.99)$$

де $\tau_{\text{б}}$ – час передавання одного біту символу через канал зв'язку, $v_{\text{б}}$ – швидкість передавання одного біту кодової послідовності. Перевага співвідношення (1.99) над співвідношенням (1.95), яке використовувалось у посібнику раніше, полягає у тому, що значення усередненого параметру ефективності коду Хаффмена не залежить від швидкості передавання одного біту в каналі зв'язку, а характеризує середню степінь зайнятості каналу, незалежно від його пропускної здатності. Зазвичай для кодів Хаффмена усереднений параметр ефективності перевищує значення 0,9. Зрозуміло, що згідно із теоремою Шеннона 1.2 параметр ефективності коду не може бути більшим за 1.

CodeOut – вихідний вектор значень номерів закодованих елементів для заданої кодової послідовності.

Проміжні змінні, необхідні для організації коректної роботи програми:

n – кількість елементів у векторі **vh**.

vsort – вектор значень ймовірностей появи символів після їхнього упорядкування за спаданням.

MH – матриця сум ймовірностей символів.

VN – відсортований вектор сум ймовірностей символів для кожної ітерації. Використовується для формування матриці сум **MH**.

DHM – матриця номерів елементів початкового вектора за порядком їхнього розташування у матриці сум на відповідній ітерації.

SEL(2, k) – матриця S_{el} номерів елементів, які сумуються.

k – кількість ітерацій.

MCE (n,n) – матриця зв'язаних елементів M_{ce}, яка розраховується ітераційно через значення елементів у матриці S_{el}.

ConnElUp – вектор зв'язаних елементів для верхнього елемента відповідної колонки матриці S_{el}, якій формується через відповідний рядок матриці M_{ce}.

ConnElUDown – вектор зв'язаних елементів для нижнього елемента відповідної колонки матриці S_{el}, якій формується через відповідний рядок матриці M_{ce}.

ConnElem – зв'язаний елемент, для поточної ітерації за матрицею S_{el}. Тобто, елемент, номер якого стоїть у нижньому рядку матриці S_{el} на поточній ітерації.

Wrt – ознака проведення запису бітів символів на поточній ітерації.

ii, **ee**, **tt**, **jjj**, **kk**, **kkk**, **kk1, kk2, rr, dd, qq** – змінні-лічильники для організації циклів за кількістю повторень.

Алгоритм формування кодів символів, реалізований у програмі huffman, наведений на рис. 1.68. Для описання роботи алгоритму використані наступні функції дискретної математики [48, 54]:

sorthl(**r**) – функція сортування елементів вектора **r** за спаданням;

ord($\mathbf{r_2}$,$\mathbf{r_1}$) – функція пошуку порядкового номера елементів сформованого вектора $\mathbf{r_2}$ відносно початкового вектора $\mathbf{r_1}$.

Крім цього, алгоритм запису розрядів до кодів символів реалізований окремим функціональним блоком F1, а алгоритм формування матриці зв'язків між елементами – окремим функціональним блоком F2, що спрощує формування загальної блок схеми алгоритму та її розуміння. Блоки F1 та F2 пов'язані із головним блоком алгоритму через параметри виклику функцій P1, P2 та P3, проте у цих допоміжних блоках передбачена також можливість використання всіх змінних, визначених у головному блоці. Такий підхід цілком відповідає концепції функціонального програмування системи науково-технічних розрахунків MatLab. Блок-схема алгоритму функціонального блоку F1 наведена на рис. 1.64, а функціонального блоку F2 – на рис. 1.70.

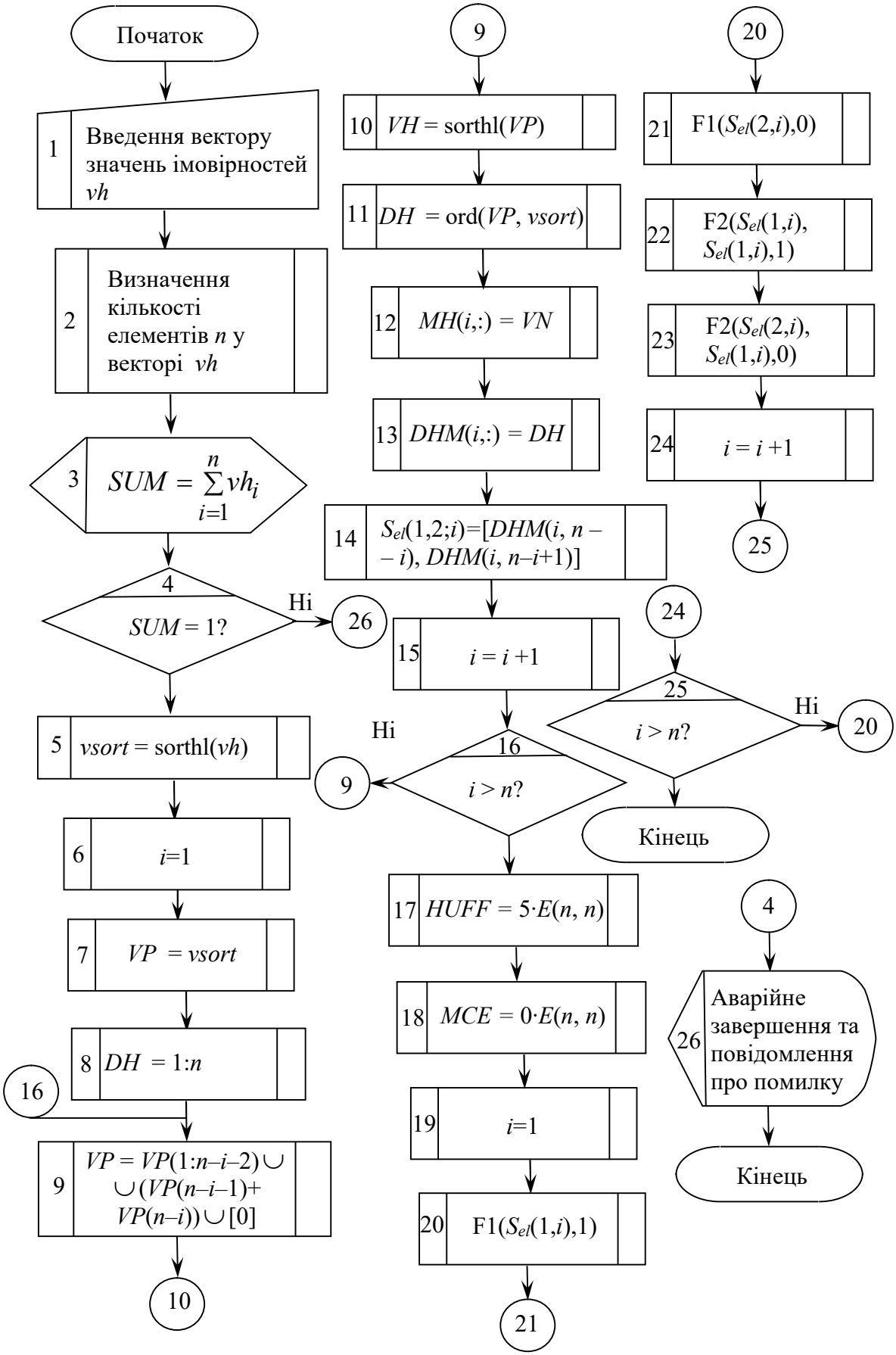


Рис. 1.68 Узагальнений алгоритм роботи програми **huffman**

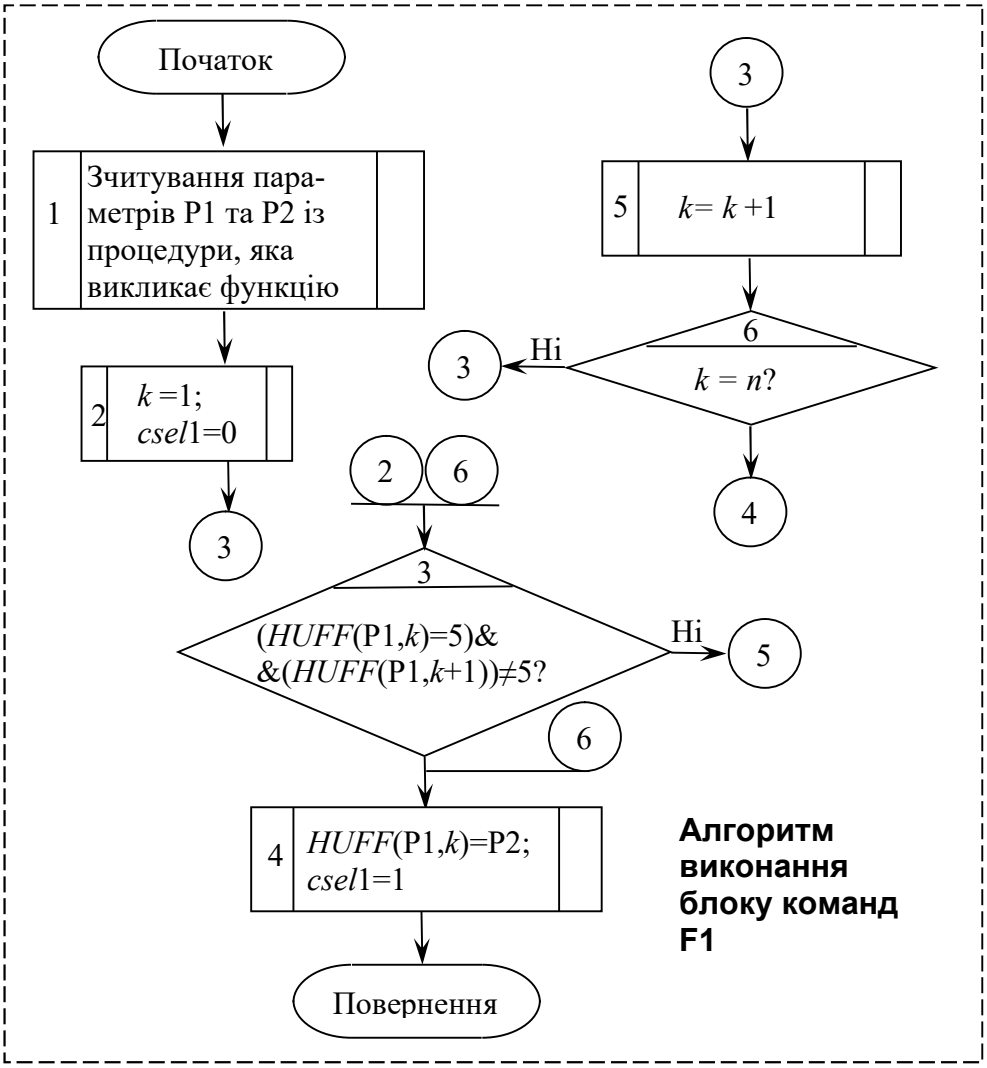


Рис. 1.69 Алгоритм запису нулів та одиниць до матриці кодів Хаффмена

Декодер коду Хаффмена, реалізований у програмі huffman, працює наступним чином. Через аналіз змінної **LSH** шукається найменша можлива довжина кодових послідовностей **LMIN** та найбільша **LMAX**. Після цього організується цикл від значення **LMIN** до значення **LMAX** та серед визначених кодових послідовностей шукаються такі, які відповідають початку заданого вектора бітів **vc**. Якщо така послідовність знайдена, номер знайденого елементу записується у вихідний вектор **CodeOut** та пошук продовжується далі, до кінця вектора **vc**. У разі, якщо у заданій послідовності бітів на будь-якій ітерації не знайдено жодного символу коду, який відповідає записаним бітам, така послідовність вважається помилковою.

Результати роботи програми huffman для різних значень ймовірностей символів абетки та різних бітових послідовностей також наведені у додатку Ж.

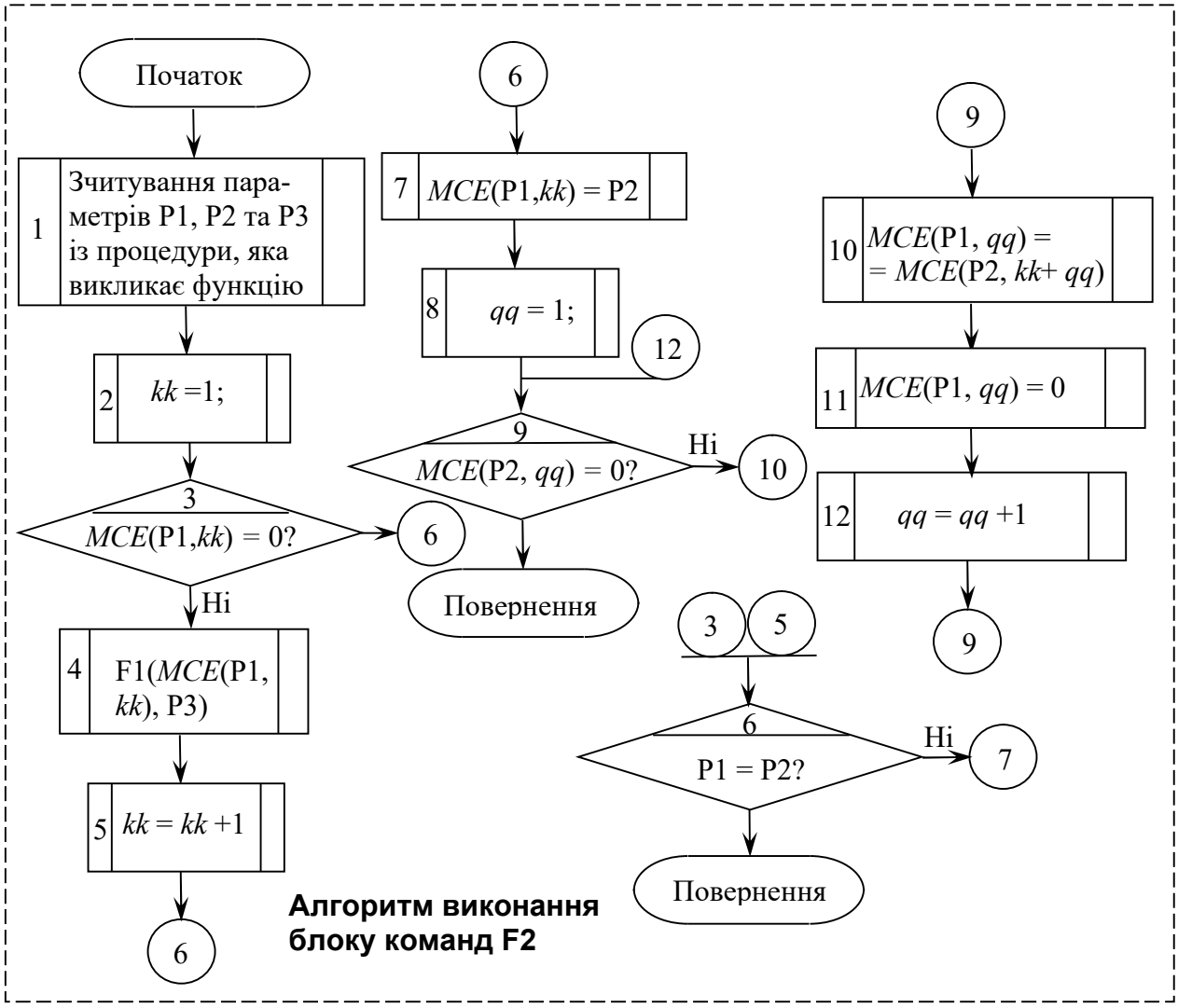
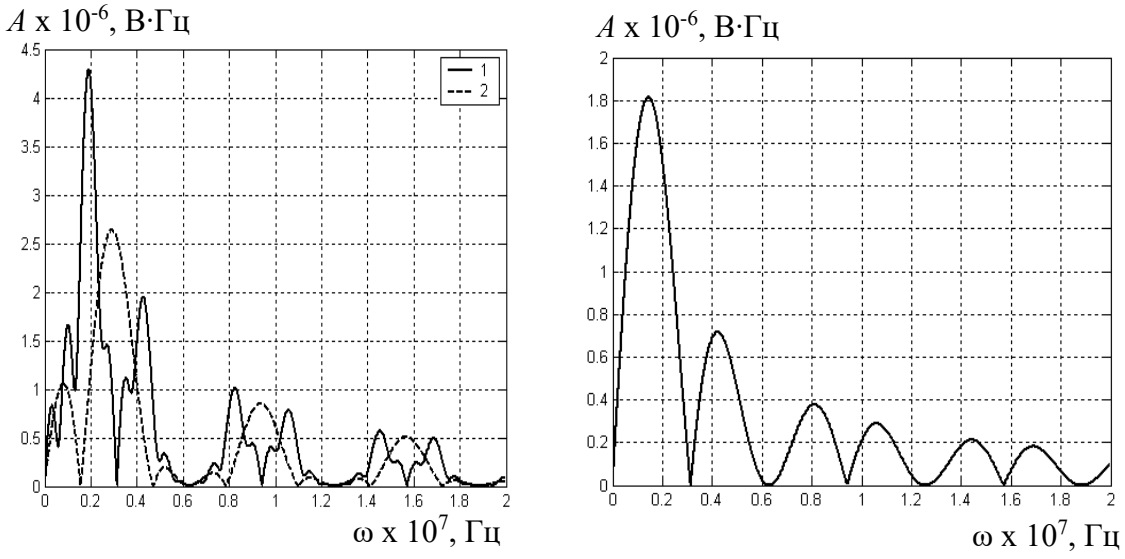


Рис. 1.70 Алгоритм формування матриці зв’язаних елементів

Код комп’ютерної програми, наведеної у додатку Ж, містить кілька сотень рядків і є досить непростим для розуміння. Для розуміння таких складних програм, написаних засобами структурного програмування, важливо знати всі особливості алгоритму, який реалізується, а також засоби формування та використання упорядкованих структур даних, притаманних даній системі програмування. Вміння аналізувати алгоритмічні особливості поставленої задачі та створювати подібні програми дійсно є високим інженерним мистецтвом. Для цього, насамперед, необхідні базові знання з різних розділів дискретної математики, які були розглянуті у трьох томах другої частини посібника [48 – 50], але навіть цих глибоких знань не достатньо для створення складних програм без наявності творчого

інженерного підходу та оригінального мислення. Недарма видатний американський вчений та програміст Дональд Кнут у серії своїх популярних книг називає програмування мистецтвом [59]. Це мистецтво потребує поглиблених знань з різних розділів дискретної та обчислювальної математики, теорія алгоритмів, теорії інформації, а також практичних знань відповідних методів та засобів програмування, реалізованих у конкретних комп'ютерних системах та мовах програмування. Важливим є також обрання відповідного стилю програмування, яке зазвичай базується на порівнянні переваг та недоліків різних можливих стилів. Наприклад, вище у цьому підрозділі був проведений такий порівняльний аналіз засобів матричного та структурного програмування системи MatLab. Також обрання стилю програмування диктується вимогами до програм, які створюються, серед яких головними є універсальність, надійність роботи та простота розуміння написаного програмного коду [13, 14]. Програмні засоби аналізу та обробки даних є однією із найважливіших компонент сучасної кодувальної електронної апаратури, що забезпечує високу швидкодію електронних засобів кодування та низьку ймовірність обчислювальних помилок під час формування кодів. Реалізація різних алгоритмів кодування та декодування двійкових послідовностей у сучасних засобах моделювання електронних систем дозволяє аналізувати роботу складних систем, в яких використані різні способи кодування сигналів, через параметричний виклик відповідних функцій, а у разі необхідності з використанням програми, наведеної у додатку В, аналізувати спектри цих сигналів. Такий наскрізний аналіз роботи електронних системи на різних ієрархічних рівнях дозволяє отримувати повну інформацію про поведінку системи за різних умов. Тому і подальший розгляд у цьому посібнику всіх типів блочних та ітераційних завадостійких кодів, а також криптографічних засобів кодування та методів стиснення даних, буде завершуватись аналізом алгоритмів формування відповідних кодів та реалізацією цих алгоритмів засобами програмування системи MatLab.

Наприкінці цього підрозділу розглянемо частотний спектр для кодової послідовності коду Хаффмена та порівняємо його із спектром тих же самих символів у разі використання натурального коду. Як і в попередніх прикладах, будемо вважати, що для подання сигналу у каналі зв'язку використовується код AMI, який був розглянутий у підрозділі 1.2.5.2. Спектр сигналу будемо будувати для абетки символів, яка розглядалася у прикладах 1.21 – 1.24, за умови, що передаються символ 7 та символ 8, незважаючи на те, що імовірність появи такої послідовності символів є невисокою. Тоді, у разі використання натурального коду необхідно сформувати цифровий сигнал із бітовою послідовністю 01111000, а за умови використання сформованого коду Хаффмена – сигнал із бітовою послідовністю 1010110100. Графічні залежності для спектрів цих сигналів за умови використання для їхнього кодування коду AMI наведені на рис. 1.71, а.



а) б)

Рис. 1.71 Порівняння спектрів сигналів коду Хаффмена (1) та натурального коду (2) у разі використання для подання сигналу у каналі зв'язку коду AMI. Абетка символів задана на рис. 1.50; а – для послідовності символів x_7, x_8; б – для послідовності символів x_1, x_1, x_2

За умов, які розглядаються, у разі використання натурального коду послідовність із сьомого та восьмого символів кодується вісім'ю бітами, а у

разі використання коду Хаффмена – десятьма бітами, тому спектри сигналів натурального коду та коду Хаффмена майже співпадають за частотним діапазоном, проте амплітуда сигналу для коду Хаффмена є більшою. Оскільки найбільш імовірні символи коду мають меншу кількість бітів, їхні кодові послідовності будуть мати менший частотний діапазон та меншу амплітуду. Наприклад, на рис. 1.66, б, показаний спектр сигналу коду Хаффмена для послідовності символів x_1, x_1, x_2, якій відповідає послідовність із шести бітів 010100.

1.5.8 Інші різновиди ефективних кодів

Основні граничні співвідношення (1.67 – 1.95), які основані на теоремі Шеннона для каналу зв'язку без завад та були розглянуті у підрозділах 1.5.2 та 1.5.4, дозволяють уникнути надлишковості кодування у всіх телекомунікаційних системах та мережах. Проте, розглянуті у підрозділах 1.5.4, 1.5.5 та 1.5.6 коди Шеннона – Фано та коди Хаффмена, незважаючи на їхню оптимальність з точки зору теорії інформації, мають і суттєві недоліки, зокрема [47]:

1. велика затримка під час занесення повідомлень до буферу пам'яті;

2. необхідність апріорного знання імовірності появи символів повідомлення;

3. можливість використання таких способів кодування лише для дискретних систем зв'язку без пам'яті, структура яких розглядалась у підрозділі 2.3.1 першої частини посібника (рис. 2.41) [1].

Для запобігання перших двох із перелічених недоліків були розроблені ефективні коди із апріорі невідомим розподілом ймовірностей. Відповідний метод кодування відомий як метод «купки книжок», або як ущільнення із сортуванням. Тобто, імовірності появи символів у повідомленні визначаються через його сортування та підрахунок кількості бітів кожного із символів абетки.

Борис Якович Рябко
(народився 1949)

Серед ефективних кодів, в яких використовується метод ущільнення інформації із сортуванням, найбільш відомим та розповсюдженим є спосіб кодування, запропонований Б. Рябко [47]. Сутність алгоритму кодування за методом Рябко можна описати наступним чином. Нехай абетка джерела повідомлення містить K символів, тобто символи з номерами 1, 2, ..., K. Кодувальний алгоритм зберігає послідовність символів, яка являє собою відповідне їх пересування відносно первинної абетки. Під час надходження на вхід деякого символу із номером i кодувальний алгоритм зберігає його префіксний код. Після цього символ пересувається на початок послідовності і номера всіх символів, які надійшли перед ним, збільшуються на 1. Саме таким чином штучно створюється ситуація, коли символи, що зустрічаються найчастіше, переміщуються на початок списку, і цим символам присвоюються більш короткі коди. В результаті, як і коди Шеннона – Фано та Хаффмена, спосіб кодування Рябко дозволяє зменшити об’єм вихідного потоку інформації, який формується передавальним пристроєм. Загалом способи побудови ефективних кодів базуються на теорії регулярних мов, яка була розглянута у підрозділі 7.3.1 другої частини посібника [50].

Існують також способи формування ефективних кодів, які через використання динамічної пам’яті джерела повідомлення та алгоритмів

синтаксичного аналізу дозволяють уникнути і третього із перелічених вище недоліків. Дійсно, у посимвольному кодуванні, яке розглядалося раніше, не використовуються можливості ущільнення інформації, які пов'язані із ланцюговим повторюванням символів, тобто із пам'яттю каналу зв’язку. Сьогодні найбільш розвиненим, відомим та розповсюдженим алгоритмом ущільнення даних, який базується на кодуванні ланцюгів символів, є алгоритм Лемпеля – Зіва, який також називають кодом Лемпеля – Зіва – Велча (англійський термін – код LZW). Він, зокрема, використовується у сучасних програмах для архівації даних. Головна ідея алгоритму Лемпеля – Зіва – Велча полягає в тому, що ланцюги символів записуються до відповідного словника. Тоді у вихідному потоці даних кодові послідовності, які вже зустрічалися, замінюються на посилання на їх координати у словнику. Головна ідея алгоритму Лемпеля – Зіва проілюстрована на рис. 1.72 [47].

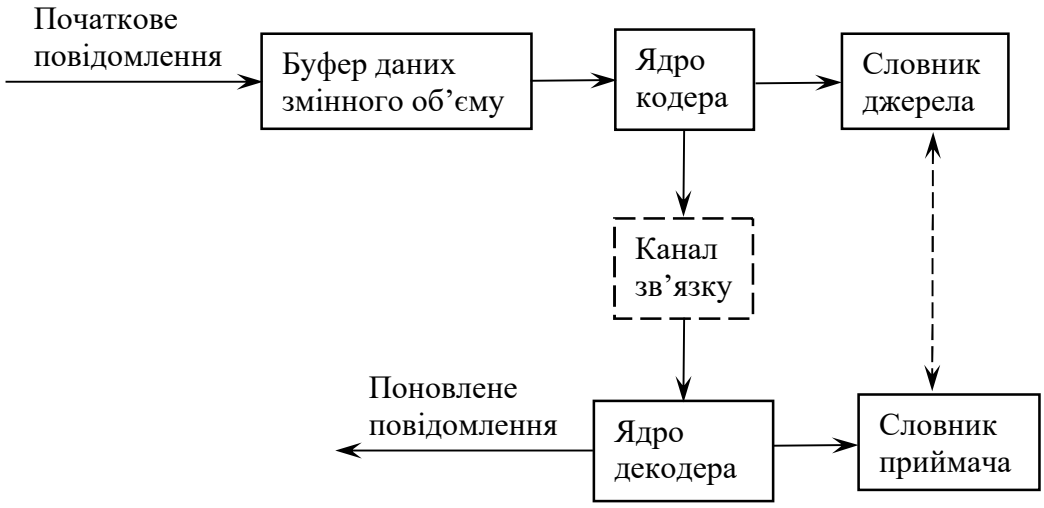
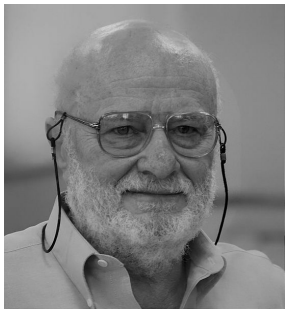


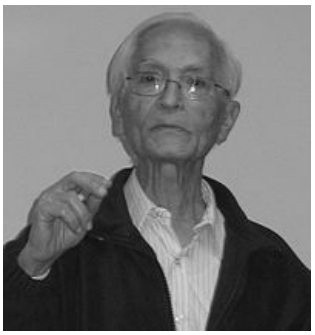
Рис. 1.72 Ілюстрація ідеї алгоритму стиснення даних Лемпеля – Зіва та способу її реалізації

Згідно із наведеною схемою ущільнення даних та їхнього передавання повідомлення, яке ущільнюється, послідовно входить до буфера джерела. Спочатку ядро кодера виділяє у буфері блоки символів максимальної довжини (зазвичай це 16 символів) та з використанням алгоритмів порівняння намагається знайти відповідні послідовності символів у словнику. Потім такий пошук повторюється для більш коротких

послідовностей. У разі, якщо відповідна послідовність знайдена, в канал зв'язку передаються її координати, а якщо навіть для двох символів відповідних співпадань зі словником не існує, такі символи передаються окремо.



Авраам Лемпель
(народився 1936)



Яков Зів
(народився 1931)

На приймальній стороні ядро декодера приймає кодовані повідомлення та відновляє їх початковий зміст відповідно до власного словника. Поновлені ланцюги символів відразу записуються до словника приймача, і таким чином здійснюється синхронізація словників джерела та приймача у динамічному режимі. Головними принципами кодування за методом Лемпеля – Зіва є наступні [47].

1. Коди координат ланцюгів символів та коди окремих символів мають відповідні бітові ознаки. Наприклад, координати ланцюгових символів починаються з символу 0, а коди символів – з символу 1.

2. Оскільки ланцюги символів зазвичай знаходяться на початку словника та є короткими, можна отримати додаткове ущільнення інформації через використання кодів Хаффмена для значень довжини ланцюгових символів та їх адрес у словнику.

3. Для кодів Лемпеля – Зіва канал є узагальненим поняттям. Тобто, джерелом та приймачем інформації можуть бути комп'ютерні файли.

4. Обмеженість об'єму словника призводить до того, що нові ланцюги символів замінюють попередні. Тому у разі використання словників малого об'єму відсоток стиснення інформації зменшується, проте меншим стає і час,

який витрачається на її декодування. Навпаки, у разі великого об'єму словника кількість інформації, яка передається, є меншою, проте для декодування таких повідомлень необхідний більший час.

Алгоритм кодування Лемпеля – Зіва використовується у більшості сучасних комп'ютерних програм-архіваторів, зокрема zip, arj, rar. Різниця у швидкості та ефективності кодування та декодування інформації визначається особливостями алгоритму та об'ємом словника. Зазвичай у програмах-архіваторах можна ручним способом змінити ступінь ущільнення інформації, відповідно до цього змінюється і час архівації та розархивації файлів. Тобто, степінь стиснення інформації у комп'ютерних системах та час її розархивації є суперечливими параметрами, і пошук оптимального методу архивації є складною оптимізаційною задачею, яка має суперечливо-компромісний характер.

Крім класичної схеми алгоритму Лемпеля – Зіва на практиці застосовується також алгоритм Лемпеля – Зіва – Велча, який є більш ефективним. Вікно програми WinRAR, у якому визначається метод стиснення даних, показано на рис. 1.73.

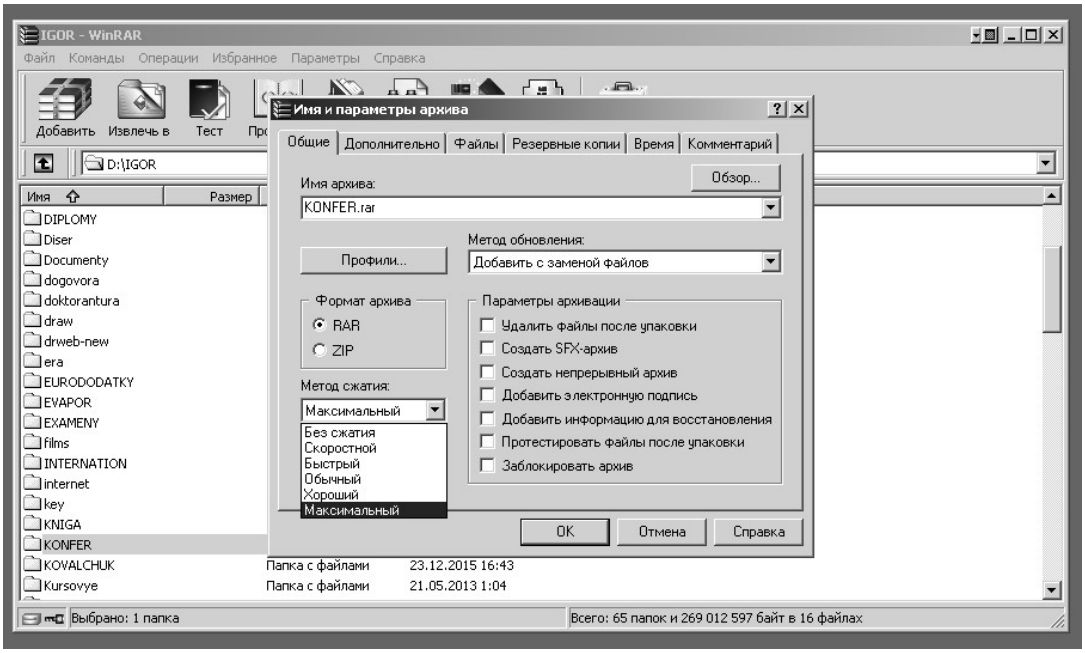


Рис. 1.73 Вікно програми-архіватора RAR із визначенням методу стиснення даних

Більш досконало сучасні методи кодування із ущільненням інформації та відповідні алгоритми будуть розглянуті у підрозділі 4.1 цієї частини посібника.

Контрольні питання та завдання до розділу 1

1. Що являє собою кодування аналогових сигналів?

2. Чи можна розглядати модуляцію сигналу як спосіб його кодування? Поясніть свою відповідь.

3. Що являє собою різницеве кодування аналогових сигналів?

4. Що являє собою квадратурна модуляція?

5. Чи можна вважати аналогово-цифрове перетворення способом кодування сигналу? Поясніть свою відповідь.

6. У чому полягає головна задача кодування сигналів в електронних системах?

7. Надайте узагальнене визначення кодування сигналів. Поясніть свою відповідь та наведіть доречні приклади.

8. Що являє собою декодування сигналів?

9. Де розташований кодер в узагальненій структурі електронної системи?

10. Де розташований декодер в узагальненій структурі електронної системи?

11. Чим відрізняються способи кодування аналогових та цифрових сигналів?

12. Чому способи кодування цифрових сигналів зазвичай є більш ефективними, ніж способи кодування аналогових сигналів? Свою відповідь обґрунтуйте.

13. На яких теоретичних засадах базуються способи кодування цифрових сигналів?

14. Чим відрізняються натуральні двійкові коди від завадостійких кодів? Поясніть свою відповідь та наведіть доречні приклади.

15. Поясніть узагальнену класифікацію цифрових кодів, наведену на

рис. 1.1.

16. Що являють собою арифметичні коди?

17. Що являють собою комбінаторні коди?

18. Поясніть фізичний зміст співвідношення (1.1).

19. Що являють собою одиничні коди?

20. Що являють собою двійкові коди?

21. Що являють собою багатопозиційні коди?

22. Що являють собою рівномірні та нерівномірні коди? Наведіть доречні приклади.

23. Що являє собою натуральне кодування? Наведіть приклади натуральних кодів.

24. Що являють собою послідовні коди? Наведіть доречні приклади.

25. Що являють собою паралельні коди? Наведіть доречні приклади.

26. Що являють собою систематичні та несистематичні коди? Наведіть доречні приклади.

27. Що являє собою роздільні та нероздільні коди? Наведіть доречні приклади.

28. До класу яких кодів, рівномірних чи нерівномірних, відносяться ефективні коди? Свою відповідь обґрунтуйте.

29. Як і у яких випадках під час формування кодованих сигналів використовуються методи синхронізації?

30. Що являють собою засоби стохастичного кодування та у яких випадках вони використовуються?

31. Яким чином рівномірні та нерівномірні коди використовуються у сучасних системах зв’язку?

32. У чому полягають особливості використання послідовних та паралельних кодів у сучасних системах зв’язку?

33. Які головні параметри цифрових кодів?

34. Як обчислюється кількість символів абетки для двійкових сигналів?

35. Як обчислюється кількість символів абетки для багатопозиційних

сигналів?

36. Що являють собою ітеративні коди і як вони використовуються у сучасних електронних системах та системах зв’язку?

37. Що являють собою неперервні коди і як вони використовуються у сучасних електронних системах та системах зв’язку?

38. Що являють собою згорткові коди і як вони використовуються у сучасних електронних системах та системах зв’язку?

39. Які способи кодування цифрової інформації, паралельні чи послідовні, зазвичай використовуються у сучасних системах оброблення інформації та її збереження?

40. Яка елементарна одиниця інформації використовується комп’ютерами під час проведення обчислень?

41. Що являє собою потенціальне кодування двійкових сигналів?

42. Які переваги та недоліки потенціального кодування двійкових сигналів?

43. Що являє собою імпульсне кодування двійкових сигналів?

44. Які переваги та недоліки імпульсного кодування двійкових сигналів?

45. У чому полягає недолік синхронного модемного зв’язку між комп’ютерами через телефонний канал?

46. Що являє собою асинхронний модемний зв’язок?

47. Як у сучасній комп’ютерній техніці використовуються сигнали із кодоімпульсною модуляцією?

48. Які способи кодування сигналів використовуються сьогодні у цифрових системах зв’язку?

49. Поясніть форми цифрових сигналів, які наведені на рис. 1.3.

50. Поясніть форми подання цифрової інформації, які наведені на рис. 1.4.

51. У яких електронних системах та системах зв’язку використовується фазова модуляція двійкових сигналів?

52. Поясніть графічні залежності, які наведені на рис. 1.5.

53. Поясніть принцип роботи схеми формування двійкового сигналу із

фазовою модуляцією, наведеної на рис. 1.6.

54. Поясніть принцип роботи спрощеної схеми фазового модулятора, яка наведена на рис. 1.7.

55. Який спектр має двійковий сигнал із фазовою модуляцією і які головні спектральні параметри такого сигналу?

56. Яким способом можна обчислити спектр двійкового сигналу із фазовою модуляцією? Наведіть доречні приклади.

57. Поясніть графічні залежності, які наведені на рис. 1.8 та 1.9.

58. Який головний недолік двійкових сигналів із фазовою модуляцією?

59. Поясніть фазові діаграми сигналів, які наведені на рис. 1.10.

60. Чому у двійкових сигналах із фазовою модуляцією не використовують засоби синхронізації?

61. Що являє собою диференціальна фазова модуляція як спосіб кодування сигналів?

62. Поясніть структурну схему каналу зв'язку, наведену на рис. 1.11.

63. Поясніть принцип роботи диференціального кодера.

64. Поясніть принцип роботи цифрової схеми, наведеної на рис. 1.12.

65. Що являє собою логічна функція «виключного або» для двох аргументів? Наведіть приклади використання цієї функції.

66. Як у схемі диференціального кодера використовується операція затримки сигналу на 1 біт?

67. Поясніть принцип роботи диференціального кодера, логічна схема якого наведена на рис 1.13.

68. Поясніть принцип роботи диференціального декодера, логічна схема якого наведена на рис 1.14.

69. Як працює диференціальний декодер у випадку інверсії біт у інформаційному потоці?

70. Поясніть принцип роботи диференціального декодера, логічна схема якого наведена на рис 1.15. Порівняйте цю схему із схемою, наведеною на рис. 1.14.

71. Побудувати диференціальні коди без зсуву фази та із зсувом фази на 1 розряд для наступних кодових послідовностей:

а) 10010101; б) 10110101; в) 11110111; г) 10110101; д) 10111101;

е) 10011101; є) 10111101; ж) 10110111; з) 11110101; і) 10111111.

72. Який головний недолік диференціальної фазової модуляції цифрових сигналів?

73. Як працює диференціальний декодер у випадку одиночної бітової помилки у потоці даних?

74. Що являє собою «ефект розмноження помилок»?

75. Поясніть принцип роботи диференціального декодера, логічна схема якого наведена на рис 1.16. Порівняйте цю схему із схемою, наведеною на рис. 1.14.

76. Яким чином обчислюється імовірність бітової помилки для сигналів із фазовою модуляцією?

77. Поясніть фізичний зміст формули (1.5).

78. Що являє собою функція помилок Гаусса $\Phi(x)$?

79. Поясніть графічну залежність, яка наведена на рис. 1.17.

80. Що являє собою принцип матричного програмування та чим він суттєво відрізняється від класичного принципу структурного програмування?

81. Як з використанням принципу матричного програмування записуються рекурентні співвідношення?

82. Поясніть рекурентні співвідношення для диференціального кодування сигналів (1.6). Наведіть приклади використання цих співвідношень.

83. Як у математиці та у теорії кодування позначаються вхідні та вихідні вектори кодека та декодека? У чому полягає суттєва різниця цих позначень та як вони пов’язані із теорією відношень та із теорією функцій? Поясніть співвідношення (1.7).

84. Як записується у рекурентній формі алгоритм декодування диференціального коду? Поясніть співвідношення (1.8).

85. Що являють собою арифметико-логічні вирази і як вони

використовуються у засобах матричного програмування? Наведіть приклади використання арифметико-логічних виразів. Поясніть співвідношення (1.9).

86. Чим суттєво відрізняються арифметико-логічні функції двійкової арифметики від арифметико-логічних функцій для роботи із дійсними числами? Наведіть власні приклади арифметико-логічних функцій двійкової арифметики.

87. У чому полягають головні особливості реалізації арифметико-логічних функцій під час запису рекурентних відношень двійкової арифметики з використанням засобів матричного програмування? Наведіть приклади таких арифметико-логічних функцій.

88. Поясніть код програми **`recvectbin`**, який наведений у додатку Б. Чим ця програма суттєво відрізняється від програми **`recvect`**, призначеної для проведення рекурентних обчислень над дійсними числами?

89. У чому полягає універсальність процедури **`recvectbin`**? Свою відповідь обґрунтуйте.

90. Організувати диспут, під час якого студенти обговорюють переваги та недоліки засобів матричного програмування. Для цього розділіть групу на дві частини. Перша підгрупа має приводити аргументи щодо переваг використання матричних алгоритмів, а представники другої вважають, що зручніше застосовувати стандартні засоби структурного програмування. Під час проведення диспуту студенти повинні використовувати заздалегідь підготовлені комп'ютерні презентації.

91. Поясніть двійкові рекурентні співвідношення (1.13) та (1.14).

92. Поясніть код програми diffcode, наведеної у додатку Б.

93. Що являє собою квадратурна модуляція сигналу і в яких електронних системах та системах зв'язку вона використовується?

94. Які переваги та недоліки квадратурної модуляції?

95. Поясніть кругові діаграми, які наведені на рис. 1.18.

96. Поясніть параметри сигнально-кодової конструкції із квадратурною амплітудною модуляцією, яка наведена в таблиці 1.1.

97. Поясніть схему модулятора, яка формує сигнал із квадратурною амплітудною модуляцією, що наведена на рис. 1.19, а.

98. Поясніть схему демодулятора, яка декодує сигнал із квадратурною амплітудною модуляцією, яка наведена на рис. 1.19, б.

99. Що являє собою сигнал із зірковою квадратурною модуляцією?

100. Що являє собою сигнал із круговою квадратурною модуляцією?

101. Поясніть фазові діаграми, які наведені на рис. 1.20.

102. Як обчислюється спектр сигналу із квадратурною амплітудною модуляцією?

103. У яких електронних системах та системах зв'язку використовується квадратурна амплітудна модуляція?

104. У яких електронних системах та системах зв'язку використовується багаторівнева фазова модуляція?

105. Що являють собою імпульсні коди із внутрішньою синхронізацією? Наведіть приклади таких кодів.

106. Що являє собою манчестерський код та за яким принципом він формується?

107. Поясніть форму сигналу, наведеного на рис. 1.21.

108. У яких сучасних системах зв'язку використовуються манчестерський код?

109. Які головні параметри двійкового сигналу, закодованого манчестерським кодом?

110. Що являє собою різницевий манчестерський код та за яким принципом він формується?

111. Поясніть форму сигналу, наведеного на рис. 1.22.

112. У яких сучасних системах зв'язку використовуються різницеві манчестерські коди?

113. Які головні параметри двійкового сигналу, закодованого різницевим манчестерським кодом?

114. Що являє собою код Міллара та за яким принципом він

формується?

115. Поясніть форму сигналу, наведеного на рис. 1.23.

116. Які головні параметри двійкового сигналу, закодованого кодом Міллара?

117. Що являють собою коди із інверсією кодових посилань та за яким принципом вони формуються?

118. Поясніть форму сигналу, наведеного на рис. 1.24.

119. Які головні параметри двійкового сигналу, закодованого кодом із інверсією кодових посилань?

120. Що являють собою уніполярні коди та як вони використовуються у сучасних системах зв'язку?

121. Які переваги та недоліки уніполярних кодів?

122. Що являє собою уніполярний код без повернення до нуля (NRZ)?

123. Поясніть форму сигналу, наведеного на рис. 1.25.

124. Які головні параметри двійкового сигналу, закодованого кодом без повернення до нуля?

125. Що являє собою уніполярний код із поверненням до нуля?

126. Чим відрізняється уніполярний код із поверненням до нуля від коду без повернення до нуля?

127. Поясніть форму сигналу, наведеного на рис. 1.26.

128. Які головні параметри двійкового сигналу, закодованого кодом із поверненням до нуля?

129. Що являє собою потенціальний код із інверсією за умови одиниці?

130. Поясніть форму сигналу, наведеного на рис. 1.27.

131. У яких системах зв'язку використовується код із інверсією за умови одиниці?

132. Чим суттєво відрізняється код із інверсією за умови одиниці від інших уніполярних кодів?

133. Що являють собою біполярні потенціальні коди та чим вони суттєво відрізняються від уніполярних?

134. Що являє собою біполярний код без повернення до нуля?

135. Поясніть форму сигналу, наведеного на рис. 1.28.

136. Що являє собою біполярний код із поверненням до нуля?

137. Поясніть форму сигналу, наведеного на рис. 1.29.

138. Які головні параметри сигналу, закодованого біполярним кодом без повернення до нуля?

139. Які головні параметри сигналу, закодованого біполярним кодом із поверненням до нуля?

140. Чому у разі використання біполярних потенціальних кодів у системах зв'язку бажано уникати постійної складової сигналу?

141. Що являє собою код без повернення до нуля із інверсією за умови одиниці (NRZI, або AMI)?

142. Поясніть форму сигналу, наведеного на рис. 1.30.

143. Які головні параметри сигналу, закодованого кодом AMI?

144. Які головні переваги та недоліки сигналу, закодованого кодом AMI?

145. Що являє собою код B8ZS і який спосіб його формування?

146. Що являє собою код HDB3 і який спосіб його формування?

147. Що являє собою сигнал забороненої одиниці у кодах B8ZS та HDB3?

148. Якою послідовністю символів замінюються довгі послідовності нулів у коді B8ZS?

149. Якою послідовністю символів замінюються довгі послідовності нулів у коді HDB3?

150. Поясніть часові діаграми, наведені на рис. 1.31.

151. У яких сучасних системах зв'язку використовується код B8ZS?

152. У яких сучасних системах зв'язку використовується код HDB3?

153. Чи можна вважати коди B8ZS та HDB3 завадостійкими? Свою відповідь обґрунтуйте.

154. Вважаючи, що час передавання одного біта становить τ, побудувати часові залежності сигналу для наведених нижче кодових

послідовностей із десяти біт за умови використання кодів B8ZS та HDB3.

а) 1000000100; б) 1000100000; в) 1000000000; г) 1000000001;

д) 1000010001; е) 1001000001; є) 1100000000; ж) 1100000001;

з) 1000000011; і) 1100000100.

155. Що являють собою багаторівневі потенціальні коди та чим вони відрізняються від уніполярних та біполярних?

156. Що являє собою багаторівневий потенціальний код 2B1Q і який спосіб його формування?

157. Поясніть форму сигналу, наведеного на рис. 1.32.

158. Що являє собою багаторівневий потенціальний код MLT-3 та яким чином він формується?

159. Поясніть форму сигналу, наведеного на рис. 1.33.

160. У яких сучасних системах зв'язку використовується код MLT-3?

161. Поясніть графічні залежності, наведені на рис. 1.34.

162. Проведіть порівняльний аналіз спектрів сигналів, закодованих кодами 2B1Q, B8ZS, манчестерським кодом, біполярним імпульсним кодом, кодом NRZ та кодом MLT-3.

163. Організувати диспут, під час якого студенти обговорюють переваги та недоліки різних способів оцінювання спектрів двійкових сигналів, як-то: перетворення Фур'є у дійсній формі, перетворення Фур'є у комплексній формі, інтеграл Фур'є, використання функцій Уолша, дискретне перетворення Фур'є, швидке перетворення Фур'є, перетворення Лапласа та z – перетворення. Під час проведення диспуту студенти повинні використовувати заздалегідь підготовлені комп'ютерні презентації.

164. Які способи кодування сигналів використовуються у сучасних електронних системах та системах зв'язку і чому?

165. Які способи кодування сигналів використовуються у сучасних оптичних системах зв'язку і чому?

166. Які способи кодування сигналів використовувались у комп'ютерних мережах стандартів Ethernet та Token Ring?

167. Які способи кодування сигналів використовуються у комп’ютерних мережах сучасних стандартів Fast Ethernet та Gigabit Ethernet?

168. Організувати диспут, під час якого студенти обговорюють переваги та недоліки різних способів кодування сигналів у каналах зв’язку. Необхідно обрати один із відомих способів кодування та обговорити його переваги та недоліки. Під час проведення диспуту студенти повинні використовувати заздалегідь підготовлені комп’ютерні презентації.

169. Яким чином можна розраховувати спектри реальних сигналів за умови відомої бітової послідовності та відомого способу кодування?

170. Поясніть співвідношення (1.15) – (1.17) та коди програм, які наведені у додатку В.

171. Поясніть графічні залежності, які наведені на рис. 1.35 – 1.37.

172. Для яких цілей здійснюється цифрове кодування сигналів в електронних системах та у системах зв’язку?

173. Чим суттєво відрізняються способи фізичного та логічного кодування сигналів?

174. Що являє собою код ASCII і для яких цілей він використовується?

175. Які набори символів входять до абетки коду ASCII?

176. Як символи коду ASCII використовуються під час написання комп’ютерних програм?

177. Чи можна вважати код ASCII двійковим кодом і чому?

178. Чи можна вважати код ASCII комбінаторним кодом і чому?

179. Що являє собою код із доповненням до одиниці і як він використовуються у комп’ютерній техніці?

180. Яким чином у комп’ютерній техніці записуються від’ємні числа?

181. Як виконується арифметичні операції над від’ємними числами у комп’ютерній техніці?

182. У чому полягає неоднозначність подання додатних та від’ємних чисел у комп’ютерній техніці?

183. У разі виконання яких комп’ютерних операцій програмісту

необхідно розрізняти від'ємні числа від додатних і яким чином це можна зробити?

184. Поясніть співвідношення (1.19). Як воно використовується для формування додаткового коду від'ємних чисел?

185. Чим відрізняються у комп'ютерній арифметиці коди від'ємних цілих чисел від кодів від'ємних дрібних чисел? Наведіть приклади запису таких чисел у додатковому коді.

186. Записати наступні цілі від'ємні числа у коді із доповненням до одиниці, вважаючи, що для кодування чисел використовується один байт. Визначити, яким додатним числам відповідають сформовані бітові послідовності.

а) –41; б) –78; в) –96; г) –63; д) –54; е) –71; є) –91; ж) –69; з) –71; і) –89.

187. Поясніть спосіб отримання додаткового коду дрібного числа, наведений у прикладі 1.1.

188. Поясніть співвідношення (1.20) та (1.21). Чим вони відрізняються одне від одного і у яких випадках вони використовуються?

189. Чи можна сказати, що співвідношення (1.20) є окремим випадком співвідношення (1.21)? Свою відповідь обґрунтуйте.

190. Вважаючи, що розрядна сітка обчислювального пристрою має 8 розрядів, записати у форматі із фіксованою комою наступні від'ємні дрібні числа:

а) –29,34; б) –65,78; в) –93,21; г) –74,35; д) –79,51;

е) –62,38; є) –43,25; ж) –97,125; з) –37,15; і) –71,35.

191. Що являє собою зсунутий код і як він використовується у двійковій арифметиці? Наведіть приклади формування зсунутого коду.

192. Поясніть співвідношення (1.22).

193. Чому за умови використання зсунутого коду спрощується операція порівняння від'ємних чисел?

194. Що являє собою зсунутий код із від'ємним нулем і як він використовується у двійковій арифметиці? Наведіть приклади формування

зсунутого коду із від'ємним нулем.

195. Поясніть співвідношення (1.23).

196. Які дрібні числа у двійковій арифметиці вважаються скінченними дробами, а які – нескінченними, і чому?

197. Яким чином у комп'ютерній арифметиці точність обчислень пов'язана із розрядністю дрібної частини чисел? Поясніть свою відповідь.

198. Організувати диспут, під час якого студенти обговорюють переваги та недоліки різних способів запису чисел у комп'ютерній арифметиці. Необхідно обрати один із відомих способів запису від'ємних або дрібних чисел та обговорити його переваги та недоліки. Під час проведення диспуту студенти повинні використовувати заздалегідь підготовлені комп'ютерні презентації.

199. Що являють собою рефлексні коди та яким чином вони використовуються у комп'ютерній техніці?

200. Яким чином формуються двійкові послідовності коду Грея?

201. Поясніть формули (1.24). Яким чином ці формули використовуються для формування коду Грея?

202. Поясніть спосіб отримання коду Грея із двійкового числа, який наочно показаний на рис. 1.38, а.

203. Поясніть формули (1.25). Яким чином ці формули використовуються для зворотного перетворення коду Грея до двійкового числа?

204. Поясніть спосіб отримання двійкового числа із його коду Грея, який наочно показаний на рис. 1.38, б.

205. Поясніть, за яким алгоритмом здійснюється переведення двійкового числа до коду Грея та зворотне перетворення коду Грея до двійкового числа (приклад 1.2).

206. Поясніть співвідношення (1.26). Яким чином це співвідношення використовується для отримання із коду Грея еквівалентного десяткового числа?

207. Які переваги та недоліки коду Грея порівняно із звичайним двійковим кодом? Обґрунтуйте свою відповідь та наведіть доречний приклад.

208. Чому і за яких умов у разі використання кодів Грея зазвичай імовірність обчислювальних помилок є меншою, ніж у разі використання звичайних двійкових кодів?

209. Чому і за яких умов у разі використання кодів Грея потужність, яку споживає обчислювальний пристрій, зазвичай є значно меншою, ніж у разі використання звичайних двійкових кодів?

210. Поясніть спосіб обчислення імовірності помилок та потужності, яку споживає обчислювальний пристрій, за умови використання коду Грея та звичайного двійкового коду, який розглянутий у прикладі 1.4.

211. Переведіть десяткові числа, задані у пунктах а – і, до двійкової системи числення та сформуйте для них коди Грея. Кількість розрядів сформованого числа має бути мінімальною.

а) 38; б) 56; в) 48; г) 42; д) 84; е) 75; є) 79; ж) 93; з) 110; і) 117.

Зробити зворотне перетворення, тобто отримати із сформованого коду Грея початкову послідовність біт двійкового числа.

212. Розрахувати потужність, яка необхідна для переходу електронного обчислювального пристрою з числа n_1 до числа n_2 у разі використання звичайного двійкового коду та коду Грея. Потужність перемикання одного біту вважати рівною $P_б$. Розрахувати імовірність обчислювальної помилки під час такого перемикання за умови, що імовірність помилки в одному біті складає $p_б$.

а) $n_1 = 15$, $n_2 = 16$, $P_б = 1$ мВт, $p_б = 10^{-3}$;

б) $n_1 = 23$, $n_2 = 24$, $P_б = 3$ мВт, $p_б = 5 \cdot 10^{-4}$;

в) $n_1 = 31$, $n_2 = 32$, $P_б = 7$ мВт, $p_б = 2 \cdot 10^{-4}$;

г) $n_1 = 47$, $n_2 = 48$, $P_б = 0{,}5$ мВт, $p_б = 3 \cdot 10^{-4}$;

д) $n_1 = 63$, $n_2 = 64$, $P_б = 0{,}7$ мВт, $p_б = 2{,}5 \cdot 10^{-4}$;

е) $n_1 = 95$, $n_2 = 96$, $P_б = 0{,}3$ мВт, $p_б = 4 \cdot 10^{-4}$;

є) $n_1 = 79$, $n_2 = 80$, $P_б = 2$ мВт, $p_б = 5{,}5 \cdot 10^{-4}$;

ж) $n_1 = 72$, $n_2 = 73$, $P_б = 10$ мВт, $p_б = 5 \cdot 10^{-3}$;

з) $n_1 = 39$, $n_2 = 40$, $P_б = 15$ мВт, $p_б = 6 \cdot 10^{-3}$;

і) $n_1 = 111$, $n_2 = 112$, $P_б = 20$ мВт, $p_б = 8 \cdot 10^{-3}$.

213. Поясніть співвідношення (1.27) та (1.28). Як ці співвідношення пов'язані із арифметико-логічними виразами (1.9) та чи є вони рекурентними?

214. Поясніть код програми, наведеної у додатку Д.

215. Розіграйте із своїми одногрупниками коротку сценку, у якій один студент поясняє іншому особливості формування кодів Грея та його переваги над двійковим кодом.

216. Що являє собою код 4B/5B?

217. Чи можна вважати код 4B/5B надлишковим і чому?

218. Чи можна вважати код 4B/5B завадостійким і чому?

219. У яких системах зв'язку використовується код 4B/5B?

220. Порівняйте частотний спектр коду 4B/5B із спектрами кодів 2B1Q, B8ZS, манчестерських кодів, біполярних кодів та коду NRZ.

221. Поясніть графічну залежність, яка наведена на рис. 1.37.

222. Що являє собою код 8B/6T?

223. Чи можна вважати код 8B/6T надлишковим і чому?

224. Чи можна вважати код 8B/6T завадостійким і чому?

225. Яким чином формуються коди 4B/5B та 8B/6T у сучасному електронному обладнанні?

226. Вважаючи, що час передавання одного біта повідомлення становить τ, побудуйте графічні залежності напруги сигналу від часу для бітових послідовностей, що наведені у пунктах а – і, за умови використання однополярного коду із поверненням до нуля, однополярного коду без повернення до нуля, коду із інверсією за умови одиниці, біполярного коду із поверненням до нуля, біполярного коду без повернення до нуля, коду AMI (NRZI), потенціального коду 2B1Q, коду 4B/5B, манчестерського коду, різницевого манчестерського коду та коду із інверсією кодових посилань. З використанням графічних залежностей, які наведені на рис. 1.34 та 1.40,

оцініть частотні діапазони, які необхідні для передавання таких сигналів без істотних спотворень.

а) 10100010; б) 11101010; в) 11100010; г) 10110110; д) 11110110;

е) 10101010; є) 11111010; ж) 10100011; з) 10111110; і) 10111110.

227. Поясніть код програми, наведеної у додатку Е.

228. Поясніть графічні залежності, наведені на рис. 1.41.

229. Що являють собою алгоритми скрамблювання та як вони використовуються у сучасних системах зв'язку та у комп'ютерних мережах?

230. Поясніть співвідношення (1.29) та (1.30).

231. Організувати диспут, під час якого студенти обговорюють переваги та недоліки різних методів логічного кодування сигналів, зокрема, кодів 4B/5B, 8B/6T, 2B1Q, B8ZS, HDB3 та MLT-3. Порівняти спектри цих сигналів. Під час проведення диспуту студенти повинні використовувати заздалегідь підготовлені комп'ютерні презентації.

232. Що являє собою операція „виключного або" для трьох аргументів? Наведіть приклади використання цієї функції.

233. Поясніть, як використовується алгоритм скрамблювання на практиці (приклад 1.5).

234. Поясніть співвідношення (1.31).

235. Для чого у сучасній комп'ютерній техніці та у системах зв'язку використовуються алгоритми скрамблювання?

236. У стандартах яких сучасних комп'ютерних мереж використовуються алгоритми скрамблювання?

237. Чи можна вважати способи кодування сигналів B8ZS та HDB3 алгоритмами скрамблювання і чому?

238. Яким чином і на основі яких пристроїв реалізуються алгоритми скрамблювання у сучасній електронній апаратурі?

239. Що являють собою твірні поліноми і як вони використовуються для реалізації алгоритмів скрамблювання в електронних схемах?

240. Розіграйте із своїми одногрупниками коротку сценку, у якій один

студент поясняє іншому особливості алгоритмів скремблювання та необхідність їх використання у сучасній цифровій електронній апаратурі.

241. Поясніть співвідношення (1.32), (1.33). Як вони пов’язані із кодом програми, наведеної у додатку Є.

242. Поясніть графічні залежності, які наведені на рис. 1.45.

243. Що являє собою генератор псевдовипадкових послідовностей?

244. Поясніть структурну схему системи, наведеної на рис. 1.42.

245. Для чого у системах зв’язку, які працюють на основі алгоритмів скремблювання, використовуються засоби синхронізації?

246. Поясніть принцип роботи системи передавання інформації із скремблюванням, структурна схема якої наведена на рис. 1.43.

247. Поясніть принцип роботи системи передавання інформації із скремблюванням, структурна схема якої наведена на рис. 1.44.

248. Яким чином алгоритми скремблювання використовуються у сучасних безпроводових системах зв’язку?

249. Чи можуть алгоритми скремблювання забезпечувати криптографічний захист інформації, яка передається? Свою відповідь обґрунтуйте.

250. До якого класу кодів можна віднести алгоритми скремблювання? Свою відповідь обґрунтуйте.

251. З використанням алгоритму скремблювання, заданого співвідношеннями (1.29) та (1.30), отримати коди для наступних бітових послідовностей:

а) 10000010; б) 10000000; в) 11100000; г) 10010000; д) 10000110;

е) 10000001; є) 10000010; ж) 10100000; з) 10000011; і) 00000111.

З використанням співвідношень (1.31) зробити зворотне перетворення, тобто отримати із визначеного скремблер-коду початкову бітову послідовність.

252. Поясніть принцип роботи програми, наведеної у додатку Є.

253. Що являють собою *m*-послідовності та як вони використовуються

у системах зв’язку?

254. Які властивості *m*-послідовностей Вам відомі? Поясніть сутність цих властивостей?

255. Як *m*-послідовності та їхні властивості пов’язані із теорією груп? Свою відповідь обґрунтуйте та наведіть доречні приклади.

256. Як *m*-послідовності та їхні властивості пов’язані із теорією поліномів? Свою відповідь обґрунтуйте та наведіть доречні приклади.

257. Поясніть приклади 1.6 – 1.8.

258. Поясніть структурні схеми кодувальних систем, які наведені на рис. 1.46 та 1.47.

259. Поясніть сутність теореми 1.1.

260. Як визначається кореляційна функція *m*-послідовностей? Поясніть співвідношення (1.35) – (1.37).

261. Поясніть приклад 1.9, таблицю 1.4 та рис. 1.48, 1.49.

262. Що являють собою послідовності Лежандра і як вони пов’язані із *m*-послідовностями? Наведіть власні приклади формування послідовностей Лежандра.

263. Поясніть співвідношення (1.38) – (1.40).

264. Що являє собою границя Велча і як вона оцінюється?

265. Поясніть співвідношення (1.41) – (1.60).

266. Що являє собою параметр *TSC* для широкосмужних систем зв’язку? Наведіть власні приклади обчислення цього параметра.

267. Поясніть приклади 1.10 та 1.11 та співвідношення (1.61).

268. Що являє собою децимація *m*-послідовності? Наведіть доречні приклади виконання цієї операції.

269. Поясніть властивості 1.4 – 1.6 та співвідношення (1.62), (1.63). Як ці властивості *m*-послідовностей пов’язані із методом формування кодів Голда? Свою відповідь обґрунтуйте та наведіть доречні приклади.

270. Поясніть співвідношення (1.64) та структурну схему кодувального пристрою, яка наведена на рис. 1.50.

271. Як оцінюється максимальне значення функції кореляції для кодів Голда? Поясніть співвідношення (1.65), (1.66).

272. Поясніть приклади 1.12 та 1.13.

273. У чому полягають недоліки кодів Голда? Які інші ітеративні блокові коди використовуються у сучасних системах зв'язку?

274. Організувати диспут, під час якого студенти обговорюють переваги та недоліки використання *m*-послідовностей для кодування цифрових сигналів. Під час проведення диспуту студенти повинні використовувати заздалегідь підготовлені комп'ютерні презентації.

275. Що є головною задачею ефективного кодування і як ця задача вирішується?

276. Які головні параметри ефективних кодів?

277. Що являє собою середня розрядність символу як параметр ефективного коду?

278. Чому системи передавання інформації, в яких використовуються ефективні коди, не потребують засобів синхронізації?

279. Які існують способи формування ефективних кодів та чим вони відрізняються?

280. Поясніть емпіричний спосіб формування ефективного коду, який описаний у прикладі 1.14.

281. Поясніть фізичну сутність теореми Шеннона 1.2 для каналу зв'язку без завад.

282. Поясніть фізичну сутність співвідношення (1.67) та його зв'язок із теоремою Шеннона 1.2.

283. Поясніть фізичну сутність співвідношення (1.68) та його зв'язок із теоремою Шеннона 1.2.

284. Що являють собою типові послідовності символів?

285. Що являють собою нетипові послідовності символів?

286. Поясніть співвідношення (1.69) – (1.77). Як вони пов'язані із доведенням теореми Шеннона 1.2? Свою відповідь обґрунтуйте.

287. Поясніть фізичну сутність співвідношень (1.78) та (1.79).

288. Які важливі умови необхідно виконувати під час формування ефективного коду?

289. Що являє собою метод невизначених множників Лагранжа і як він використовується для розв’язування оптимізаційних задач?

299. Яким чином метод невизначених множників Лагранжа може бути ефективно використаний для пошуку параметрів ефективного коду?

291. Поясніть фізичний зміст співвідношення (1.80).

292. Поясніть фізичний зміст співвідношення (1.81).

293. Поясніть, яким чином із співвідношення (1.81) можна отримати співвідношення (1.82).

294. Поясніть, яким чином із співвідношення (1.81) отримані співвідношення (1.83) та (1.84).

295. Поясніть, яким чином із співвідношення (1.81) отримано співвідношення (1.85).

296. Поясніть, яким чином із співвідношень (1.83) та (1.85) отримано співвідношення (1.86).

297. Поясніть, яким чином із співвідношення (1.86) отримано співвідношення (1.87).

298. Поясніть, яким чином із співвідношень (1.80), (1.81) та (1.87) отримано співвідношення (1.88).

299. Поясніть, яким чином із співвідношень (1.80), (1.85) та (1.89) отримані співвідношення (1.90).

300. Поясніть фізичну сутність співвідношень (1.90).

301. Як для ефективних кодів пов’язані між собою такі параметри, як імовірність появи символу, час передавання символу та швидкість передавання інформації?

302. Які переваги та недоліки ефективних кодів?

303. Які головні принципи побудови коду Шеннона – Фано?

304. Як розраховуються параметри коду Шеннона – Фано?

305. Поясніть фізичну сутність співвідношень (1.91) – (1.95).

306. Поясніть спосіб побудови коду Шеннона – Фано у разі кодування одного символу, описаний у прикладі 1.15.

307. Поясніть спосіб побудови коду Шеннона – Фано у разі кодування послідовностей із двох символів, описаний у прикладі 1.16.

308. Розіграйте із своїми одногрупниками коротку сценку, у якій один студент поясняє іншому спосіб формування кодів Шеннона – Фано та можливості використання таких кодів у сучасних електронних системах.

309. Що являє собою алгоритм побудови коду Хаффмена і які головні особливості його практичної реалізації?

310. Поясніть спосіб побудови коду Хаффмена, описаний у прикладі 1.17.

311. Поясніть спосіб формування ієрархічного дерева для коду Хаффмена, приклад якого наведений на рис. 1.52.

312. Яким чином розраховуються параметри коду Хаффмена?

313. Поясніть спосіб розрахунку параметрів коду Хаффмена, описаний у прикладі 1.18.

314. Що являє собою однопрохідний код Хаффмена і який спосіб його формування?

315. Що являє собою двопрохідний код Хаффмена і який спосіб його формування?

316. Поясніть спосіб побудови коду Хаффмена, описаний у прикладі 1.19, та структури ієрархічних дерев, наведені на рис. 1.53, а, б.

317. Поясніть спосіб побудови коду Хаффмена, описаний у прикладі 1.20 та структуру ієрархічного дерева, наведену на рис. 1.54.

318. Поясніть спосіб побудови коду Хаффмена для прикладу 1.21 та структуру ієрархічного дерева, наведеного на рис. 1.56.

319. Побудувати код Шеннона – Фано та код Хаффмена за умови наступних ймовірностей символів абетки:

а) $p(x_1) = 0{,}25,\ p(x_2) = 0{,}25,\ p(x_3) = 0{,}15,\ p(x_4) = 0{,}35$;

б) $p(x_1) = 0{,}1,\ p(x_2) = 0{,}5,\ p(x_3) = 0{,}1,\ p(x_4) = 0{,}3$;

в) $p(x_1) = 0{,}1,\ p(x_2) = 0{,}7, p(x_3) = 0{,}1, p(x_4) = 0{,}1$;

г) $p(x_1) = 0{,}1,\ p(x_2) = 0{,}1,\ p(x_3) = 0{,}6, p(x_4) = 0{,}3$;

д) $p(x_1) = 0{,}05,\ p(x_2) = 0{,}05, p(x_3) = 0{,}6, p(x_4) = 0{,}3$.

е) $p(x_1) = 0{,}3,\ p(x_2) = 0{,}2, p(x_3) = 0{,}2, p(x_4) = 0{,}3$.

є) $p(x_1) = 0{,}25,\ p(x_2) = 0{,}25, p(x_3) = 0{,}1, p(x_4) = 0{,}4$;

ж) $p(x_1) = 0{,}25,\ p(x_2) = 0{,}2,\ p(x_3) = 0{,}4, p(x_4) = 0{,}15$;

з) $p(x_1) = 0{,}25,\ p(x_2) = 0{,}2,\ p(x_3) = 0{,}35, p(x_4) = 0{,}15$;

і) $p(x_1) = 0{,}1,\ p(x_2) = 0{,}7,\ p(x_3) = 0{,}1, p(x_4) = 0{,}1$;

к) $p(x_1) = 0{,}2,\ p(x_2) = 0{,}2,\ p(x_3) = 0{,}4, p(x_4) = 0{,}2$.

Для якого випадку задачу розв’язати неможливо. Поясніть чому. Якщо задача має розв’язок, знайдіть його також для кодування послідовності із двох символів та порівняйте ефективність використання отриманих кодів.

320. Поясніть, чому якщо імовірності появи символів абетки визначені правильно і ці символи є незалежними, то послідовність із двох символів також є коректною кодовою послідовністю для побудови кодів Шеннона – Фано та Хаффмена?

321. У разі, якщо кількість символів абетки дорівнює n та всі символи є незалежними, якою може бути мінімальна кількість рівноімовірних послідовностей із двох символів? За виконання якої умови ця кількість дійсно буде мінімальною?

322. Поясніть принцип роботи кодера, принципова електрична схема якого наведена на рис. 1.57. Обґрунтуйте коректність роботи цієї схеми з точки зору двійкової арифметики та теорії скінченних автоматів.

323. Поясніть блок схему алгоритму, наведену на рис. 1.58.

324. Поясніть принцип роботи декодера, принципова електрична схема якого наведена на рис. 1.59. Обґрунтуйте коректність роботи цієї схеми з точки зору двійкової арифметики та теорії скінченних автоматів.

325. Поясніть блок схему алгоритму, наведену на рис. 1.60.

326. Поясніть спосіб формування коду Хаффмена через матрицю ідентифікації та блок схему алгоритму, наведену на рис. 1.61.

327. Розіграйте із своїми одногрупниками коротку сценку, у якій один студент поясняє іншому особливості формування коду Хаффмена через матрицю ідентифікації.

328. Поясніть приклад 1.22 та рис. 1.62, 1.63.

329. Чому матриця ідентифікації не може бути використана для автоматичного формування ієрархічного дерева коду Хаффмена з використанням електронних обчислювальних засобів? Свою відповідь обґрунтуйте.

330. Що являє собою матриця сум ймовірностей елементів для коду Хаффмена та як вона формується? Поясніть рис. 1.64 та 1.65. Наведіть власний приклад формування матриці сум ймовірностей елементів.

331. Що являє собою матриця номерів елементів, які сумуються, для коду Хаффмена, та як вона формується? Наведіть власний приклад формування матриці номерів елементів, які сумуються.

332. Чи може бути використана матриця номерів елементів, які сумуються, для автоматичного формування коду Хаффмена з застосуванням електронних програмованих обчислювальних засобів? У чому полягає складність такого використання.

333. Поясніть приклад 1.23 та рис. 1.66.

334. Що являє собою матриця зв’язків між елементами між елементами кодової послідовності для коду Хаффмена та як вона формується?

335. Яким чином матриця зв’язків між елементами для коду Хаффмена пов’язана із матрицею номерів елементів, які сумуються?

336. Яким чином матриця зв’язків між елементами для коду Хаффмена може бути використана для автоматичного формування цього коду з використанням електронних програмованих обчислювальних засобів?

337. Організувати диспут, під час якого студенти обговорюють спосіб автоматичного формування коду Хаффмена з використанням матриці номерів елементів, які сумуються, та матриці зв’язків між елементами. Під час проведення диспуту студенти повинні використовувати заздалегідь

підготовлені комп’ютерні презентації.

338. Поясніть приклад 1.24 та рис. 1.67.

339. Що являє собою усереднений параметр ефективності коду Хаффмена? Поясніть співвідношення (1.99).

340. Поясніть блок схеми алгоритмів, наведені на рис. 1.68 – 1.70.

341. Поясніть код програми, наведеної у додатку Ж, а також результати тестування цієї програми через командні рядки системи MatLab.

342. Яким чином визначаються частотні спектри для бітових послідовностей коду Хаффмена? Поясніть графічні залежності, наведені на рис. 1.71.

343. Які головні недоліки мають класичні способи побудови ефективних кодів, зокрема коди Шеннона – Фано та коди Хаффмена, і як цих недоліків можна уникнути?

344. Для яких типів каналів зв’язку використовуються коди Шеннона – Фано та коди Хаффмена?

345. Що являє собою алгоритм кодування, запропонований Рябко?

346. Які головні недоліки коду Шеннона – Фано та коду Хаффмена не притаманні алгоритму кодування Рябко?

347. Яких головних недоліків коду Шеннона – Фано та коду Хаффмена не вдається запобігти у алгоритмі кодування Рябко?

348. Чи є необхідною наявність пам’яті у передавальному пристрої каналу зв’язку у разі застосування алгоритму кодування Рябко? Свою відповідь обґрунтуйте.

349. Що являє собою алгоритм кодування Лемпеля – Зіва та як він використовується у сучасних системах зв’язку та у комп’ютерній техніці?

350. Чи є необхідною наявність пам’яті у передавальному пристрої каналу зв’язку у разі використання алгоритму кодування Лемпеля – Зіва?

351. Чи є необхідною наявність пам’яті у приймальному пристрою каналу зв’язку у разі використання алгоритму кодування Лемпеля – Зіва?

352. У чому полягає головна ідея алгоритму кодування Лемпеля – Зіва?

353. Поясніть структурну схему, яка наведена на рис. 1.72.

354. Як формуються словники передавальних пристроїв у системах кодування, основаних на алгоритмі Лемпеля – Зіва?

355. Як формуються словники приймальних пристроїв у системах кодування, основаних на алгоритмі Лемпеля – Зіва?

356. Яким чином синхронізуються словники приймального та передавального пристрою у системах кодування, основаних на алгоритмі Лемпеля – Зіва?

357. Як у кодах, сформованих за алгоритмом Лемпеля – Зіва, поодиночні символи відрізняють від ланцюгових?

358. Що являють собою адреси символів у словниках у системах кодування, де використовується алгоритм Лемпеля – Зіва?

359. Чому у системах кодування, де використовується алгоритм Лемпеля – Зіва, зазвичай використовують також код Хаффмена?

360. Чому поняття каналу зв'язку у системах кодування з ущільненням інформації зазвичай є досить умовним?

361. Як залежить час кодування та декодування інформації від об'єму словника за умови використання алгоритму Лемпеля – Зіва?

362. Як залежить степінь стиснення інформації від об'єму словника за умови використання алгоритму Лемпеля – Зіва?

363. Чому задача побудови оптимальних ефективних систем кодування з ущільненням інформації зазвичай має суперечливо-компромісний характер?

364. Організувати диспут, під час якого студенти обговорюють сучасні способи кодування інформації з використанням алгоритмів стиснення даних. Під час проведення диспуту студенти повинні використовувати заздалегідь підготовлені комп'ютерні презентації.

Розділ 2 Головні принципи завадостійкого кодування сигналів та способи формування лінійних та циклічних блокових кодів

У цьому розділі розглядаються теоретичні основи завадостійкого кодування сигналів та способи формування лінійних та циклічних кодів. Як теоретичне підґрунтя формування завадостійких кодів розглядається теорема Шеннона, наводиться доведення теореми Котельникова – Найквіста, а також у загальному вигляді описуються способи формування лінійних блокових кодів та оцінюються їхні параметри. Показується, що способи формування завадостійких кодів тісно пов'язані із розглянутими у другій частині посібника математичними методами теорії кодування, а саме, із методами теорії чисел, теорії множин, комбінаторного аналізу, із теорією поліномів, матриць та векторних просторів, із теорією ймовірностей, із статистичними методами оцінки радіотехнічних сигналів, а також із методами теорії скінченних автоматів. Розглянуті також способи формування лінійних та циклічних кодів, зокрема, коду із перевіркою на парність, коду із повторенням елементів, прямокутного коду, кодів Хеммінга та циклічних кодів. Наведені схеми кодувальних та декодувальних електронних пристроїв, алгоритми кодування та декодування сформованих кодових послідовностей, а також та комп'ютерні програми, призначені для формування кодів Хеммінга та циклічних кодів та декодування їхніх послідовностей, написані мовою програмування системи MatLab.

2.1 Головні теоретичні положення завадостійкого кодування сигналів

2.1.1 Теорема Шеннона про канал зв'язку із завадами

Перед вивченням цього підрозділу необхідно повторити підрозділ 1.5.2, підрозділ 4.3 першої частини посібника, а також підрозділи 1.3, 5.2 та 7.3 другої частини посібника

У підрозділі 4.3 першої частини посібника без доведення була сформульована теорема Шеннона про канал зв'язку із завадами [1], а у підрозділі 1.5.2 цієї частини посібника була досконало розглянута теорема Шеннона про канал зв'язку без завад. Оскільки вся сучасна теорія

завадостійкого кодування базується на теоремі Шеннона для каналу зв’язку із завадами, розглянемо тепер цю теорему більш досконало як теоретичне підґрунтя завадостійкого кодування.

Річ у теорему, що теорема Шеннона про канал зв’язку із завадами не обмежується співвідношенням (4.30), наведеним у першій частині посібника [1]. Ця теорема має більш глибокий зміст як з теоретичної, так і з практичної точки зору, і у загальному вигляді формулюється наступним чином [5, 64, 65].

Теорема 2.1. У разі будь-якої продуктивності джерела повідомлень, меншої за пропускну здатність каналу зв’язку, завжди існує такий спосіб кодування повідомлення, який дозволяє забезпечити передавання всієї інформації, що створюється джерелом повідомлення, із будь-якою малою імовірністю помилки.

Твердження, зворотне до теореми 2.1, також називається теоремою Шеннона для каналу зв’язку із завадами та формулюється наступним чином [5].

Теорема 2.2. Не існує такого способу кодування повідомлень, який дозволив би передавати інформацію із будь-якою малою імовірністю помилки, якщо продуктивність джерела повідомлення перевищує пропускну здатність каналу зв’язку.

Зрозуміло, що сформульовані теореми 2.1 та 2.2 є дуже схожими на теорему 1.1 про канал зв’язку із завадами, і тому доведення цих теорем також базується на співвідношеннях, аналогічних (1.67) – (1.77), які мають бути дещо змінені для урахування наявності завад у каналі зв’язку. Слід відзначити також, що способи доведення теорем 2.1 та 2.2 після публікації фундаментальних робіт Шеннона [7] дещо змінювались, переусвідомливалось та ускладнювались із розвитком теорії інформації та із розробкою нових способів кодування, проте головна ідея цих доказів завжди залишалась незмінною. Величезне значення теорем 2.1 та 2.2 для подальшого розвитку теорії завадостійкого кодування полягало у тому, що до робіт Шеннона вважалося, що не існує таких способів кодування повідомлень, які б дозволяли безпомилково передавати інформацію за умови наявності завад. Шеннон у своїй фундаментальній праці «Зв’язок за умови

наявності шуму» вперше теоретично показав, що такі способи кодування повідомлень насправді існують, і необхідною умовою їх існування є лише обмежена швидкість передавання інформації, менша за пропускну здатність каналу зв'язку [7, 64, 65].

Сутність доведення теорем 2.1 та 2.2 полягає у тому, що шукають середню ймовірність помилки за всіма можливими методами кодування та показують, що теоретично ця помилка може бути як завгодно малою [5, 7]. Розглянемо один із можливих способів доведення теорем 2.1 та 2.2 [5].

Як і для теореми 1.2, наведеної у підрозділі 1.5.2, будемо розглядати типові та нетипові послідовності символів. Тоді можна вважати, що за умови заданої тривалості повідомлення T та продуктивності джерела повідомлень $\bar{I}(Z)$ кодуванню підлягає лише певна кількість типових послідовностей $N(Z)$, а саме [5, 64, 65]:

$$N(Z) = 2^{\frac{T}{\tau_i} H(Z)}, \qquad (2.1)$$

де $H(Z)$ – ентропія джерела повідомлень для визначеної абетки Z, а середня тривалість символів повідомлення τ_i визначається з використанням теорії скінченних автоматів, описаної у підрозділі 7.3 другої частики посібника [75], наступним чином. Будемо вважати, що тривалість формування знака z_i, який формується джерелом повідомлення у стані S_q, становить τ_{qz_i}. Тоді, за умови відомих значень імовірності стану джерела повідомлень $p(S_q)$ та імовірності передавання знака z_i $p(z_i)$ у заданому стані S_q, можна визначити середню тривалість передавання символів повідомлення через наступне співвідношення [5, 64, 65]:

$$\tau_i = \sum_{q=1}^{R} p(S_q) \sum_{i=1}^{l} p_q(z_i) \tau_{qz_i}. \qquad (2.2)$$

Будемо вважати, що надходження в канал будь-якого із m різних елементарних вхідних сигналів є рівноймовірним і статистичний зв'язок між цими сигналами на вході каналу відсутній. Тоді можна сформувати відповідну кількість $N(u)$ рівноймовірних вхідних послідовностей із тривалістю T, яка, згідно із методами комбінаторного аналізу, розглянутими

у підрозділі 1.3 першої частини посібника [1], обчислюється наступним чином [5, 64, 65]:

$$N(Z) = 2^{\frac{T}{\tau_k}\log_2 m}. \qquad (2.3)$$

Якщо умова існування способу кодування, згідно із співвідношенням (2.3), виконується, тоді:

$$C_{\text{д}} = \frac{\log_2 m - H_V(U)}{\tau_k} > \frac{H(Z)}{\tau_i} = \bar{I}(Z), \qquad (2.4)$$

звідки

$$2^{\frac{T}{\tau_k}(\log_2 m - H_V(U))} > 2^{\frac{T}{\tau_i}H(Z)}, \quad N(U) >> N(Z), \qquad (2.5)$$

де V – множина сигналів на виході каналу зв'язку, а U – на його вході, $C_{\text{д}}$ – пропускна здатність каналу зв'язку [5, 64, 65].

Із отриманих співвідношень (2.5) випливає, що для множини вхідних сигналів U та визначеної множини символів абетки Z існує певна кількість способів кодування $C_{N(u)}^{N(z)}$, для яких, за умови $N(U) > N(Z)$, множині можливих повідомлень Z відповідає множина сигналів U із більшою потужністю. Інакше кажучи, із множини сигналів U можна виділити дозволені сигнали, яким відповідають певні символи абетки Z, та заборонені сигнали, або заборонені кодові послідовності, на які символи абетки не відображаються. У цьому і полягає головна ідея завадостійкого кодування повідомлень.

За умови рівноймовірного вибору послідовностей елементарних сигналів із множини U для будь-якої підмножини дозволених послідовностей $U_{\text{д}}$ імовірність p того, що конкретна послідовність буде віднесена до дозволених, становить [5, 64, 65]:

$$p = \frac{N(z)}{N(u)} = \frac{2^{\frac{T}{\tau_i}H(Z)}}{2^{\frac{T}{\tau_k}\log_2 m}}. \qquad (2.6)$$

Результатом дії завад під час отримання на виході каналу зв'язку

множини вихідних сигналів *v* залишається деяка невизначеність щодо переданих сигналів *u*. Кількісно ця помилка розпізнавання сигналу визначається умовною ентропією $H_V(U)$, а відповідна кількість послідовностей вихідних символів, згідно із співвідношеннями (2.5), (2.6), становить:

$$N_V(U) = 2^{\frac{T}{\tau_k} H_V(U)}. \qquad (2.7)$$

Із співвідношень (2.6), (2.7), безпосередньо випливає, що конкретна вихідна послідовність може бути ідентифікована із будь-якою малою імовірністю помилки тоді і лише тоді, коли серед послідовностей $N_V(U)$ є лише одна дозволена. Це зайвий раз підтверджує важливий висновок про те, що для компенсації втрат інформації у каналі зв'язку через наявність завад принципово необхідно вводити надлишкову інформацію у вхідні повідомлення. Згідно із отриманими результатами та наведеними теоретичними міркуваннями для визначення кількісних параметрів завадостійкого коду необхідно насамперед розрахувати середню за усіма можливими кодовими комбінаціями імовірність $\overline{P}$ того, що із $N_V(U)$ послідовностей лише одна є дозволеною, а інші $N_V(U)-1$ – відповідно, забороненими. З урахуванням співвідношення (2.6), маємо [5, 64, 65]:

$$\overline{P} = (1-p)^{N_V(U)-1}. \qquad (2.8)$$

Для спрощення подальших аналітичних розрахунків перепишемо співвідношення (2.8). Насамперед, якщо перейти від рівності до нерівності, можна ігнорувати одиницю у показнику степені, тобто [5, 64, 65]:

$$\overline{P} > (1-p)^{N_V(U)}. \qquad (2.9)$$

Тоді, розкладаючи праву частину нерівності (2.9) до степеневого ряду, маємо [5, 77]:

$$\overline{P} > 1 - pN_V(U) + \frac{p^2 N_V(U)(N_V(U)-1)}{2} - \cdots. \qquad (2.10)$$

Щоб довести можливість отримання будь-якої малої величини

середньої імовірності помилкового приймання повідомлення, необхідно показати, що степеневий ряд (2.10) збігається, тобто, що члени ряду P_n зменшуються за абсолютним значенням із збільшенням номера члена n. З урахуванням співвідношення (2.4) можна записати:

$$C_{\text{д}} - \overline{I}(Z) = \eta,$$

або

$$C_{\text{д}} = \frac{\log_2 m}{\tau_k} - \frac{H(Z)}{\tau_i} = \frac{H_V(U)}{\tau_k} + \eta, \tag{2.11}$$

де $\eta > 0$. З іншого боку, співвідношення (2.6) можна переписати у вигляді:

$$p = \frac{1}{2^{\frac{T}{\tau_k} H_V(U) + T\eta}} = \frac{1}{H_V(U) 2^{T\eta}}. \tag{2.12}$$

Згідно із відомою із математичного аналізу ознакою Лейбніца, залишок знакозмінного ряду із членами, що спадають за абсолютним значенням, має знак старшого члена, який відкидається, та є меншим, ніж цей член, за абсолютним значенням [77]. Тоді, згідно із ознакою Лейбніца, можна відкинути із степеневого ряду (2.10) члени, які мають степінь, більшу за 2. За такої умови права частина нерівності зменшується, тобто, нерівність підсилюється. Проводячи відповідні математичні перетворення, отримуємо наступний вираз:

$$\overline{P} > 1 - pN_V(U) = 1 - \frac{1}{2^{T\eta}}. \tag{2.13}$$

Вираз (2.13) дає середнє значення імовірності правильного приймання повідомлення. Відповідно, для імовірності помилки, можна записати:

$$P_{\text{пом}} = 1 - \overline{P} < 2^{-T\eta}. \tag{2.14}$$

Проаналізуємо отримане співвідношення (2.14). Зрозуміло, що ймовірність помилки $P_{\text{пом}}$ прямує до нуля за умови необмеженого збільшення тривалості повідомлення T, тобто, у разі $T \to \infty$. З іншого боку, у теорії інформації також існує теорема про те, що у разі $T \to \infty$ імовірність

появи нетипових послідовностей також наближається до нуля [5, 64, 65]. Тобто, можна остаточно стверджувати, що для будь-якого заданого значення $\eta > 0$ можна обрати таке значення T, для якого значення середньої імовірності помилкового передавання інформації буде меншим за будь-яке мале, наперед задане додатне число ε. Тому можна вважати теорему 2.1 доведеною. Доведення теореми 2.2, зворотної до теореми 2.1, можна знайти у підручнику [5] та у монографії [7].

Як вже було відмічено, теорема Шеннона є важливим теоретичним підґрунтям для визначення способів побудови систематичних та групових завадостійких кодів, які розглядатимуться у другому та третьому розділах цієї частини посібника. Відповідні параметри завадостійких кодів були описані у підрозділі 5.2 другої частини посібника. Способи розрахунку цих параметрів та взаємозв'язок між ними будуть розглянуті у підрозділі 2.1.3.

2.1.2 Цифрове кодування сигналів та обґрунтування теореми Котельникова – Найквіста

Перед вивченням цього підрозділу необхідно повторити розділи 3 та 6 першої частини посібника

У розділі 6 першої частини посібника була, без доведення, розглянута теорема Котельникова – Найквіста [1]. Крім цього, були наведені важливі приклади використання цієї теореми для аналізу можливостей дискретизації сигналів та для визначення частоти їхньої дискретизації. Нагадаємо, що сутність теореми Котельникова – Найквіста (теорема 6.1 першої частини посібника) [1] полягає у тому, що будь-яку неперервну функцію $f(t)$, частотний спектр якої обмежений частотою F_c, можна інтерполювати послідовністю відліків у моменти часу $\frac{1}{2F_c}$ у разі, якщо функціями апроксимації є функції Найквіста, задані співвідношенням (6.2) першої частини посібника [1]. Оскільки теорема Котельникова – Найквіста має важливе значення у теорії

дискретизації сигналів та широко використовується для формування завадостійких двійкових кодів, розглянемо доведення цієї теореми [5], та тим самим покажемо, яким чином вона тісно пов'язана із теорією спектрального аналізу сигналів, яка була розглянута у розділі 3 першої частини посібника [1].

Припустимо, що функція $f(t)$, яка розглядається, може бути описаною у частотній формі своєю спектральною характеристикою $S(\omega)$ так, що:

$$S(j\omega) = 0,\ |\omega| > \omega_c,\ F_c = 2\pi\omega_c. \tag{2.15}$$

Використовуючи зворотне перетворення Фур'є, яке для неперіодичних функцій визначається співвідношенням (3.27), наведеним у першій частині посібника, можна записати [1]:

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} S(j\omega)\exp(j\omega t)d\omega. \tag{2.16}$$

Тоді для моментів часу $t_n = n\Delta t = n/\pi\omega_c$ функція $f(t)$, яка розглядається, з урахуванням співвідношення (2.16), приймає наступні значення [2 – 7]:

$$f(t_n) = f\left(\frac{n\pi}{\omega_c}\right) = \frac{1}{2\pi}\int_{-\omega_c}^{\omega_c} S(j\omega)\exp\left(\frac{jn\pi\omega}{\omega_c}\right)d\omega. \tag{2.17}$$

З іншого боку, у першій частині посібника у підрозділі 3.2.2 було показано, що спектральну функцію $S(j\omega)$ на інтервалі існування $[-\omega_c, +\omega_c]$ можна розкласти в ряд Фур'є за частотами, рівними $2\omega_c$ [2 – 7]:

$$S(j\omega) = \frac{1}{2}\sum_{n=-\infty}^{\infty} \overline{A}\exp\left(\frac{j\pi\omega}{\omega_c}\right), \quad \overline{A} = \frac{1}{\omega_c}\int_{-\omega_c}^{\omega_c} S(j\omega)\exp\left(\frac{-jn\pi\omega}{\omega_c}\right)d\omega. \tag{2.18}$$

Порівнюючи співвідношення (2.17) та (2.18), після відповідних математичних перетворень отримуємо наступний вираз [2 – 7]:

$$\overline{A} = \frac{2\pi}{\omega_c} f\left(-\frac{n\pi}{\omega_c}\right) = \frac{2\pi}{\omega_c} f(-n\Delta t). \tag{2.19}$$

З урахуванням отриманого співвідношення (2.19) перепишемо перше рівняння системи (2.18) через відліки функції $f(t)$, яка інтерполюється, наступним чином [2 – 7]:

$$S(j\omega)=\frac{\pi}{\omega_c}\sum_{n=-\infty}^{\infty}f(n\Delta t)\exp(j\omega n\Delta t). \qquad (2.20)$$

Слід відзначити, що знак перед $-n$ у співвідношенні (2.20) був змінений на протилежний, але, враховуючи те, що сумування проводиться як за негативними, так і за позитивними цілими числами, така математична операція у даному випадку є цілком коректною.

Тепер, підставив значення спектру функції $f(t)$, задані співвідношенням (2.20), до початкового виразу (2.16), отримуємо в інтегральній формі значення цієї функції у будь-який момент часу [2 – 7]:

$$f(t)=\frac{1}{2\omega_c}\int_{-\omega_c}^{\omega_c}\left(\sum_{n=-\infty}^{\infty}f(n\Delta t)\exp\left(\frac{-jn\pi\omega}{\omega_c}\right)\right)\exp(j\omega t)d\omega. \qquad (2.21)$$

Згідно із теоремою 3.1 про суперпозицію спектрів сигналів, яка була розглянуті у підрозділі 3.2.5 першої частини посібника [1], в отриманому співвідношенні (2.21) постійний множник $\sum_{n=-\infty}^{\infty}f(n\Delta t)$, який не залежить від змінної інтегрування ω, можна винести за знак інтеграла. Тоді вираз (2.21) переписується наступним чином [2 – 7]:

$$f(t)=\frac{1}{2\omega_c}\sum_{k=-\infty}^{\infty}f(n\Delta t)\int_{-\omega_c}^{\omega_c}\exp(j\omega(t-n\Delta t))d\omega. \qquad (2.22)$$

В отриманому виразі (2.22) необхідно обчислити інтеграл $\int_{-\omega_c}^{\omega_c}\exp(j\omega(t-n\Delta t))d\omega$, що зроблено у прикладі 3.1, наведеному у підрозділі 3.2.3 першої частини посібника [1]. Враховуючи отриманий результат, можна записати наступний вираз [2 – 7]:

$$\int_{-\omega_c}^{\omega_c}\exp(j\omega(t-n\Delta t))d\omega=\frac{1}{j(t-n\Delta t)}\exp(j\omega(t-n\Delta t))\Big|_{-\omega_c}^{\omega_c}=$$

$$=\frac{2\sin(\omega_c(t-n\Delta t))}{t-n\Delta t}. \qquad (2.23)$$

Підставляючи аналітичний вираз (2.23) до співвідношення (2.22),

отримуємо формулу для шуканого спектру функції $f(t)$ [2 – 7]:

$$f(t)=\sum_{k=-\infty}^{\infty} f(n\Delta t)\frac{\sin(\omega_c(t-n\Delta t))}{t-n\Delta t}. \qquad (2.24)$$

Отриману формулу (2.25) можна переписати у вигляді функціонального ряду, який збігається [2 – 7]:

$$f(t)=\sum_{k=-\infty}^{\infty} C_k\psi_k(t),\ C_k=f(n\Delta t),\ \psi_k(t)=\frac{\sin(\omega_c(t-n\Delta t))}{t-n\Delta t}, \qquad (2.25)$$

де C_k – точки відліку функції сигналу $f(t)$, $\psi_k(t)$ – функції Найквіста [1]. Тобто, можна вважати, що Котельникова – Найквіста (теорема 6.1 першої частини посібника) є доведеною.

Функції Найквіста $\psi_k(t)$ називаються також функціями відліків [2, 3, 5]. Слід відзначити, що функції $\psi_k(t)$ є ортогональними, тому ряд (2.25) збігається. Дійсно [5, 64, 65]:

$$\int_{-\infty}^{\infty}\frac{\sin(\omega_c(t-n\Delta t))}{t-n\Delta t}\cdot\frac{\sin(\omega_c(t-k\Delta t))}{t-k\Delta t}d\omega=\begin{cases}\dfrac{\pi}{\omega_c}, & \text{якщо } k=n;\\ 0, & \text{якщо } k\neq n.\end{cases} \qquad (2.26)$$

У прикладі 3.1 першої частини посібника [1] було показано, що функції Найквіста $\psi_k(t)$, задані співвідношеннями (2.25), описують спектр одиночних імпульсів.

2.2 Узагальнена класифікація завадостійких кодів та їх параметри

2.2.1 Класифікація завадостійких кодів

Перед вивченням цього підрозділу необхідно повторити підрозділ 1.1

Згідно із теоремою Шеннона 2.1, розглянутою у підрозділі 2.1.1, можливі такі способи кодування сигналів у каналах зв'язку із завадами, які за рахунок наявності відповідної кількості дозволених та заборонених кодових комбінацій дозволяють виправляти помилки у кодових послідовностях, які передаються. Кількість таких дозволених та недозволених комбінацій визначається з використанням методів комбінаторного аналізу, які розглядалися у підрозділі 1.3 другої частини посібника [48]. Головна ідея

формування завадостійких кодів полягає у тому, що сигнал на приймальній стороні після дії на нього у каналі зв'язку потужної завади має бути ближчим до сигналу переданої кодової послідовності, ніж до сигналів інших послідовностей, а степінь близькості між переданою та прийнятою кодовою комбінацією визначається через кількість бітів, які відрізняються. Саме такі коди, згідно із класифікацією, наведеною на рис. 1.2, називаються коректувальними, або завадостійкими.

Оскільки для переважної більшості завадостійких кодів можливість виправлення помилок обумовлена їх алгебраїчною структурою, такі коди називають також алгебраїчними. Математичні способи побудови таких кодів та методи пошуку помилок пов'язані із поняттям векторних просторів та матричних перетворень та розглядалися у п'ятому розділі другої частини посібника [48]. Згідно із рис. 1.2, алгебраїчні коди розділяються на дві групи – блокові та неперервні. Для блокових кодів процедура кодування являє собою зіставлення кожному із k символів абетки відповідної групи із n символів. У разі використання такого способу кодування аналізується лише поточний символ, і код, який формується, не залежить від попередньої послідовності символів. Необхідною умовою формування завадостійкого коду є виконання співвідношення $n > k$ [2 – 6]. Блоковий код називається рівномірним, якщо кількість символів коду n є однаковою для всіх символів повідомлення.

Згідно із класифікацією, наведеною на рис. 1.2, розрізняють роздільні та нероздільні блокові коди. У разі використання роздільних кодів роль всіх символів коду чітко розмежовується. У загальному випадку розрізняють інформаційні символи, які відповідають повідомленню, що передається, та контрольні символи, які навмисно вводяться до кодової послідовності для забезпечення функцій знаходження та виправлення помилок. Математичні способи та алгоритми обчислення контрольних символів та формування відповідних контрольних сум для завадостійких кодів також розглядалися у п'ятому підрозділі першої частини посібника [48]. У разі кодування нероздільними кодами інформаційні та контрольні символи повідомлення не розрізняються, проте способи формування таких кодів є значно складнішими. До блокових відносяться також лінійні та циклічні коди. Алгебраїчні основи

побудови цих кодів розглядалися у розділі 5 другої частини посібника [48]. Технічні способи побудови лінійних кодів, а також відповідні алгоритми, розглядатимуться у підрозділі 2.3 та 2.4, а способи побудови циклічних кодів – у підрозділі 2.5. Окремо виділяють групові блокові коди, принципи побудови яких базуються на теорії скінченних полів Галуа та на алгебраїчних операціях над групами. Відповідний математичний апарат був розглянутий у другому та третьому розділах другої частини посібника [48], а способи побудови таких кодів розглядатимуться у розділі 3. Серед групових кодів найбільш поширеними є коди Боузера – Чоудхурі – Хоквінгема (БЧХ) та коди Ріда – Соломона [33, 40, 41].

Згідно із класифікацією, наведеною на рис. 1.2, розрізняють також неперервні, або деревоподібні коди. Особливість побудови таких кодів полягає у тому, що в них введення контрольних символів здійснюється неперервно, без ділення інформації на незалежні блоки. Неперервні коди також бувають роздільними та нероздільними. Серед неперервних кодів найчастіше використовуються згорткові, або рекурентні коди. Для таких кодів є характерним постійне здійснення процесу кодування на передавальній та деконування на приймальній стороні та передбачення найбільш імовірних послідовностей символів, які мають бути передані, з урахуванням переданої інформації. У разі невірного передбачення можуть виникати помилки декодувального пристрою, які виправляються в процесі його роботи. Згорткові коди є найпростішими серед неперервних кодів з точки зору апаратної реалізації та також будуть розглянуті у третьому розділі цієї частини посібника.

2.2.2 Параметри систематичних блокових завадостійких кодів

Перед вивченням цього підрозділу необхідно повторити підрозділ 5.2.4 другої частини посібника

Узагальнений аналіз параметрів завадостійких блокових кодів та їхня класифікація були розглянуті у підрозділі 5.2 другої частини посібника, а базуються ці параметри на методах комбінаторного аналізу, розглянутих у підрозділі 1.3 другої частини посібника [48]. Розглянемо тепер параметри блокових кодів з практичної точки зору. Згідно із міркуваннями, наведеними

у попередньому підрозділі, будемо вважати, що вхідні символи абетки кодується k бітами та, за умови $n > k$, їм відповідають n бітів вихідного коду. Оскільки кількість можливих кодових комбінацій визначається як кількість розміщень із повтореннями, згідно із теоретичними відомостями, наведеними у другій частині посібника, можна сказати, що у такому випадку кількість вхідних кодових комбінацій складає 2^k, а вихідних, відповідно, 2^n. Тоді із загальної кількості 2^n вихідних комбінацій лише 2^k є дозволеними, а інші $2^n - 2^k$ – забороненими. За таких умов спотворення інформації у каналі зв'язку можна розглядати як перехід від дозволених кодових послідовностей до інших, які можуть бути або також дозволеними, або забороненими. Згідно із правилом множення, відомим із основ комбінаторики, які розглядалися у підрозділі 1.3 першої частини посібника [48], загальна кількість можливих переходів між усіма кодовими комбінаціями складає

$$C_{\text{з}} = 2^n \cdot 2^k . \tag{2.27}$$

Серед таки переходів розрізняють наступні три випадки [2 – 5, 64, 65].

1. Безпомилкове передавання інформації. Таких варіантів може бути

$$C_{\text{б}} = 2^k. \tag{2.28}$$

2. Перехід до інших дозволених кодових комбінацій. Таким переходам відповідають помилки передавання сигналу, які неможливо виявити з використанням даного коду. Таких варіантів може бути

$$C_{\text{п}} = 2^k \cdot \left(2^k - 1\right). \tag{2.29}$$

3. Перехід до заборонених кодових комбінацій, який відповідає виявленню помилки передавання сигналу. Кількість таких варіантів складає:

$$C_{\text{в}} = 2^k \cdot \left(2^n - 2^k\right). \tag{2.30}$$

Із співвідношень (2.27) – (2.30) зрозуміло, що [2 – 5, 64, 65]:

$$C_{\text{б}} + C_{\text{п}} + C_{\text{в}} = 2^k + 2^k \cdot \left(2^k - 1\right) + 2^k \cdot \left(2^n - 2^k\right) =$$

$$= 2^k \cdot \left(1 + 2^k - 1 + 2^n - 2^k\right) = 2^k \cdot 2^n ,$$

тобто:

$$C_{\text{б}} + C_{\text{п}} + C_{\text{в}} = C_{\text{з}} . \tag{2.31}$$

Наочна ілюстрація описаних трьох типів переходів між кодовими комбінаціями показана на рис. 2.1, де жирними лініями показане безпомилкове передавання інформації, тонкими лініями – помилки, які

можуть бути виявлені, а пунктирними – помилки, які не можуть бути виявлені [5]. На рис. 2.1 вхідні кодові комбінації позначені символами $A_1 – A_i – A_j – A_k$, вихідні – символами $B_1 – B_i – B_j – B_k$, а підмножини $M_1 – M_i – M_j – M_k$ являють собою кодові комбінації, які відповідають вхідним комбінаціям $A_1 – A_i – A_j – A_k$. Розділення загальної множини заборонених кодових комбінацій згідно із їх відповідністю правильним комбінаціям проводиться з використанням методів теорії груп та з розбиттям груп на підгрупи, відповідний математичний апарат був розглянутий у розділі 2 другої частини посібника [48]. Також, в кінці другого розділу другої частини посібника, розглядався показовий приклад виправлення помилок у двійкових кодових послідовностях з використанням розбиття загальної групи всіх можливих кодових комбінацій на відповідні підгрупи. Слід відзначити, що у разі, якщо прийнята кодова комбінації B_j належить до групи M_i, яка відповідає переданій комбінації A_i, така помилка може бути виправленою [2 – 5, 64, 65].

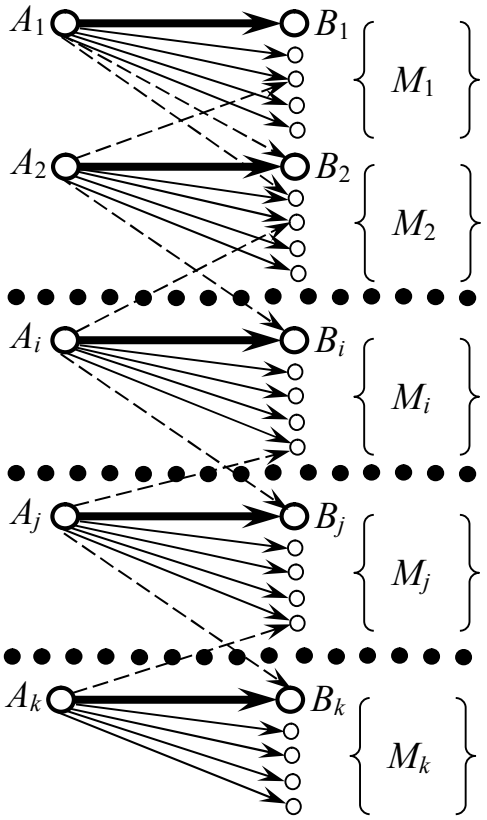


Рис. 2.1 Відповідність між вхідними, вихідними та забороненими кодовими комбінаціями для кодових послідовностей завадостійких кодів

Із рис. 2.1 зрозуміло, що загальна кількість переходів до заборонених комбінацій, згідно із співвідношенням (2.30), складає $C_{\text{в}}$, але помилка може бути виправлена лише тоді, коли перехід здійснений до кодової послідовності, яка належить до групи M_i, що відповідає переданому символу A_i. Загальна кількість таких випадків є значно меншою і становить [2 – 5, 64, 65]:

$$C_{\text{вип}} = 2^n - 2^k . \tag{2.32}$$

Таким чином, згідно із формулами (2.30), (2.32), співвідношення загальної кількості помилок, що виявляються, до кількості помилок, які можуть бути виправлені, складає [2 – 5, 64, 65]:

$$\frac{C_{\text{вип}}}{C_{\text{в}}} = \frac{2^n - 2^k}{2^k \cdot \left(2^n - 2^k\right)} = \frac{1}{2^k} . \tag{2.33}$$

Зрозуміло, що найбільш важливим під час формування завадостійких кодів є спосіб розбиття вихідних кодових комбінацій на групи. Він у загальному випадку залежить від того, які саме помилки має виявляти та виправляти відповідний код. Більшість завадостійких кодів, які сьогодні розроблені та використовуються у системах зв'язку, призначені або для виправлення взаємно незалежних помилок певної кратності, або для виправлення пачок помилок, які також іноді називаються пакетами помилок (англійський термін – error burst). Циклічні та ітераційні коди, які дозволяють виправляти пачки помилок, розглядатимуться у третьому розділі цієї частини посібника. Надамо визначення незалежної помилки (англійський термін – independent error) та кратності помилки (англійський термін – multiplication of errors) [2 – 5, 64, 65].

Визначення 2.1. Незалежною помилкою називається така помилка, для якої імовірність появи будь-якої комбінації спотворених символів залежить лише від кількості спотворених символів r та імовірності спотворення одного символу p.

Визначення 2.2. Кількість спотворених символів у кодовій комбінації

називається кратністю помилки.

Зрозуміло, що введене поняття незалежної помилки цілком відповідає визначенню незалежної події у теорії ймовірностей, яке розглядалося у розділі 6 другої частини посібника [49]. Для обчислення імовірності великої кількості таких подій використовується схема повторних випробувань Бернуллі, яка розглядалась у підрозділі 6.1 другої частини посібника [49]. Зокрема, у підрозділі 6.1.4 другої частини посібника були проведені оцінки правильного та хибного приймання двійкових повідомлень з урахуванням кратності помилок [49]. Із теоретичних відомостей, наведених у розділі 6 другої частини посібника, легко зрозуміти, що імовірність p_r кількості помилок r у кодовій комбінації довжини n обчислюється за схемою Бернуллі наступним чином [2 – 7, 49]:

$$p_r = C_n^r p^r (1-p)^{n-r}, \qquad C_n^r = \frac{n!}{r!\cdot(n-r)!}, \tag{2.34}$$

де C_n^r – кількість сполучень з n по r.

У підрозділі 6.1.4 другої частини посібника також було показано, що за умови $p << 1$ найбільш імовірними є помилки малої кратності [49], проте у багаторозрядних кодах, навіть за цієї умови, загальна ймовірність помилок малої кратності, яка визначається співвідношенням (2.34), може бути досить високою і більш ніж на порядок перевищувати імовірність спотворення одного символу p [49]. Саме тому більшість завадостійких кодів, які сьогодні розроблені, орієнтовані на виправлення помилок малої кратності.

У п’ятому підрозділі другої частини посібника було показано, що коректувальна здатність завадостійких кодів тісно пов’язана із кодовою відстанню, або із відстанню Хеммінга [48]. Нагадаємо, що у загальному випадку кодова відстань визначається як сума за модулем два правильної та спотвореної кодових комбінацій. Також у розділі 5 другої частини посібника було наведено поняття мінімальної кодової відстані, яка визначається як мінімальна величина кодової відстані за усіма дозволеними комбінаціями

[48]. Тоді, у разі прийняття хибної кодової комбінації під час декодування, зазвичай вважається, що була передана правильна кодова комбінація, яка розташована на найменший кодовій відстані до прийнятої. Такий спосіб декодування сигналів також відповідає принципу максимальної правдоподібності, який був розглянутий у підрозділі 6.4 другої частини посібника [49].

У п’ятому розділі другої частини посібника були наведені та обґрунтовані наступні співвідношення між параметрами завадостійких кодів [48].

1. Для завадостійкого коду із виявленням помилок повинна виконуватись умова:

$$d_{\min} \geq t + 1, \tag{2.35}$$

де t – кратність помилок, які виявляються.

2. Для завадостійкого коду із виправленням помилок повинна виконуватись умова:

$$d_{\min} \geq 2 \cdot \sigma + 1, \tag{2.36}$$

де σ – кратність помилок, які виправляються.

3. Для завадостійкого коду із виправленням та виявленням повинна виконуватись умова:

$$d_{\min} \geq t + \sigma + 1. \tag{2.37}$$

Розглянемо наочну ілюстрацію наведених співвідношень (2.35) – (2.37) [5, 64, 65]. Припустимо, що правильним кодовим комбінаціям B_i та B_j відповідають заборонені кодові комбінації із множин M_i та M_j, і ці множини будуються наступним чином [5, 64, 65].

1. Одиночним помилкам, кількість яких для коду розрядності n становить C_n^1, відповідає перша внутрішня сфера із діаметром $d = 1$.

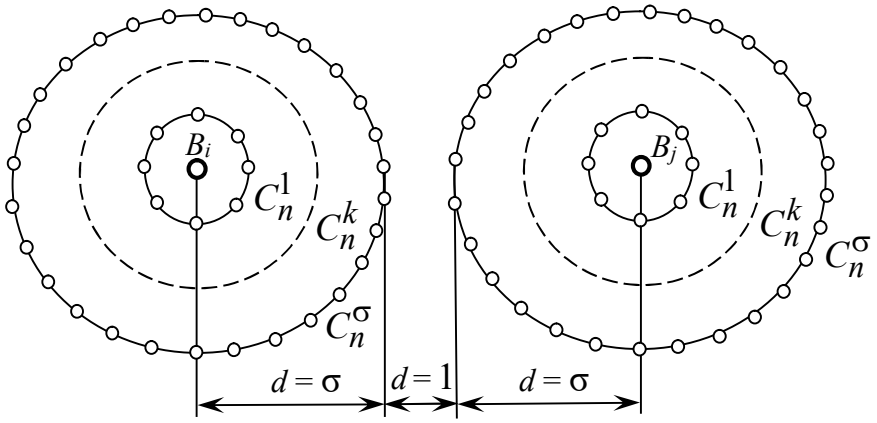
2. Подвійним помилкам, кількість яких для коду розрядності n становить C_n^2, відповідає друга внутрішня сфера із діаметром $d = 2$.

3. Із аналогічних міркувань, помилкам кратності σ, кількість яких складає C_n^{σ}, за умови, що σ – це максимальна кількість помилок, які виправляються, відповідає зовнішня сфера.

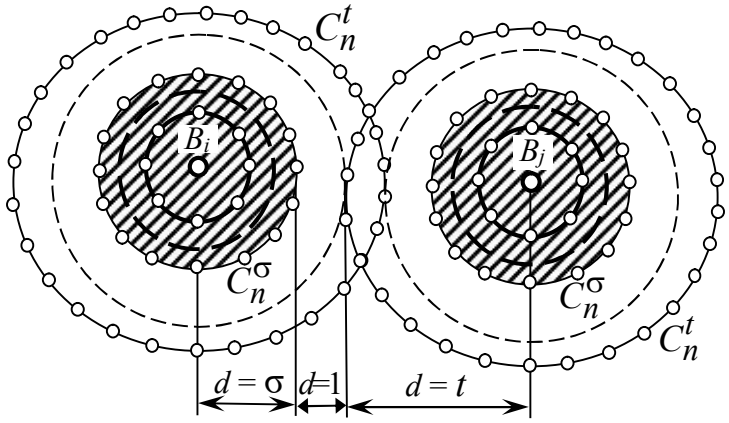
Наочна ілюстрація способів формування завадостійких кодів із

виправленням помилок наведена на рис. 2.2, а, а способів формування кодів із виправленням помилок кратності σ та виявленням помилок кратності t за умови $t > \sigma$ – на рис. 2.2, б. Зрозуміло, що рис. 2.2, а, відповідає співвідношенню (2.36), а рис. 2.2, б – співвідношенню (2.37).

Слід відзначити, що оцінки за формулами (2.35) – (2.37) часто дають занижене значення кодової відстані для реальних умов зв'язку. Це пов'язано з тим, що зазвичай час дії завади є більшим, ніж час дії сигналу, а за таких умов є можливим виникнення пачок помилок [5, 64, 65]. У цьому разі вибір кодової відстані базується на статистичних даних про помилки та відповідних статистичних оцінках параметрів сигналів, які розглядалися у підрозділі 6.5 другої частини посібника [49].



а)

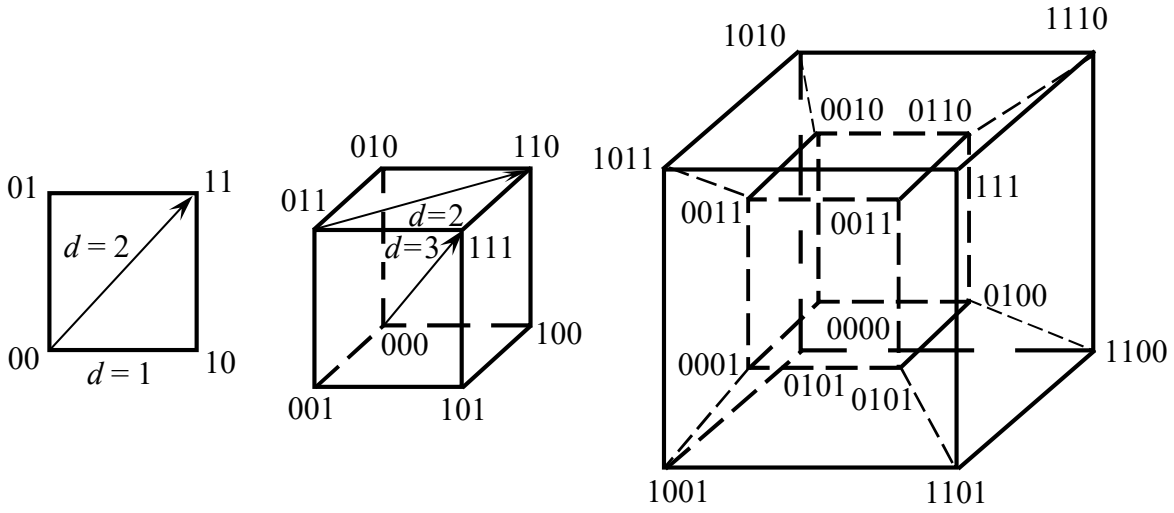


б)

Рис. 2.2 Наочна ілюстрація принципів роботи завадостійких кодів із виправленням помилок (а) та із виявленням та виправленням помилок (б)

Наприклад, якщо кодова комбінація із 18 нулів 000000000000000000 перетворюється до комбінації 01001000010101000 і вважається, що мінімальна кодова відстань дорівнює 3, тоді початкову вхідну послідовність можна розділити на два пакети по 4 і по 5 символів [5]. Зрозуміло, що за таких умов використання блокових завадостійких кодів не є ефективним. У цьому випадку більш ефективними є ітераційні згорткові коди, спосіб побудови яких орієнтований на пошук пачок помилок. Алгоритми побудови згорткових завадостійких кодів розглядатимуться у третій частині посібника у підрозділі 3.3.

Будь-яка двійкова кодова комбінація із кількістю розрядів n також може бути подана ілюстративно у вигляді куба, який має розмірність n, довжина ребра якого дорівнює одиниці. Така графічна інтерпретація тісно пов'язана із поняттям кодової відстані, яке було розглянуто у п'ятому розділі другої частини посібника, там же були наведені відповідні графічні інтерпретації для випадків $n = 3$ та $n = 4$. Більш наочні графічні інтерпретації гіперкубів для випадків $n = 2$, $n = 3$ та $n = 4$ наведені на рис. 2.3 [5, 64, 65].



а) б) в)

Рис. 2.3 Наочна ілюстрація кодових відстаней між двійковими сигналами у вигляді гіперкубів для випадків двохрозрядного (а), трьохрозрядного (б) та чотирьохрозрядного (в) коду

У загальному випадку одиничний куб розмірності n має 2^n вершин і це число дорівнює найбільшій можливій кількості кодових комбінацій. Така геометрична модель дає також просту геометричну інтерпретацію кодової відстані. Кодова відстань у даному випадку відповідає найменшій кількості ребер одиничного куба, які необхідно пройти від однієї вершини до іншої. За таких умов процес декодування із виправленням одиночних помилок можна наявно уявити наступним чином. До підмножини кожної дозволеної комбінації відносять ті вершини, які лежать у сфері із радіусом $\frac{d-1}{2}$, якщо центр цієї сфери відповідає базовій вершині. Тоді, якщо в результаті дії завади кодова комбінація переходить із точки B_i до точки B_j, а відстань між цими точками складає $\left|B_iB_j\right| \le \frac{d-1}{2}$, можна вважати, що B_i є правильною кодовою комбінацією.

Важливими є також показники якості коректувального коду, які також були розглянуті у п'ятому розділі другої частини посібника у підрозділі 5.2.4 [48]. Серед показників якості найважливішим є надлишковість коду (англійський термін – code redundancy). Якщо завадостійкий код за умови загальної кількості розрядів n містить k інформаційних розрядів, тоді надлишковість коду визначається одним із двох можливих способів [5, 64, 65]:

$$R_n = \frac{n-k}{n}, \tag{2.38}$$

$$R_k = \frac{n-k}{k}. \tag{2.39}$$

Параметр надлишковості R_k краще відповідає цьому поняттю, проте в літературі часто використовується і коефіцієнт R_n, який іноді також позначається символом ρ. [2 – 5, 33, 40, 41, 48]. Значення R_k змінюються в діапазоні [0, ∞[, а значення R_n – в діапазоні [0,1] [5, 64, 65]. Для натуральних кодів $n = k$, $R_n = 0$, $R_k = 0$.

Серед завадостійких кодів важливу роль відіграють оптимальні коди.

Надамо відповідне визначення.

Визначення 2.3. Оптимальним кодом (англійський термін – optimum code) називається завадостійкий код, який забезпечує необхідну коректувальну здатність за умови мінімальної надлишковості.

Із наведених вище міркувань можна оцінити найбільшу кількість Q дозволених комбінацій завадостійкого коду розрядності n, за умови, що припустима кратність помилок складає σ. Згідно із співвідношенням (2.36) у цьому випадку мінімальна кодова відстань становить $d_{\min} \geq 2 \cdot \sigma + 1$, а загальна кількість помилок, які виправляються для всіх кодових комбінацій, згідно із рис. 2.2, б, складає [5, 64, 65]:

$$S = \sum_{i=1}^{n} C_n^i . \tag{2.40}$$

Зрозуміло, що кожна із таких помилок призводить до переходу до забороненої кодової комбінації, яка відноситься до підмножини даної дозволеної комбінації. Тобто, разом із дозволеною комбінацією, повна підмножина M_i, яка відповідає вхідній кодовій комбінації B_i (рис. 2.1), містить

$$S_{\text{з}} = 1 + \sum_{i=1}^{n} C_n^i \tag{2.41}$$

вихідних комбінацій.

Як було відмічено раніше, і як видно із рис. 2.1 та рис. 2.2, множини M_i за будь-яких значень i не повинні перетинатися. Оскільки загальна кількість комбінацій для коду із розрядністю n складає 2^n, кількість дозволених комбінацій Q не повинна перевищувати максимального значення q [2 – 5, 64, 65]:

$$q = \frac{2^n}{1 + \sum_{i=1}^{n} C_n^i},$$

тобто

$$Q \le \frac{2^n}{\sum\limits_{i=1}^{n} C_n^i}. \qquad (2.42)$$

Ця верхня границя q для кількість дозволених комбінацій Q була знайдена Р.В. Хеммінгом. Для значень мінімальної кодової відстані $d_{\min}$ від 1 до 5 значення q наведені у таблиці 2.1 [5, 64, 65].

Таблиця 2.1 – Способи обчислення мінімальної кількості дозволених кодових комбінацій q для різних значень мінімальної кодової відстані $d_{\min}$

$d_{\min}$	q	$d_{\min}$	q
1	$q = 2^n$	5	$q = \frac{2^{n+1}}{n^2 + n + 2}$
2	$q = 2^{n-1}$	4	$q = \frac{2^{n-1}}{n}$
3	$q = \frac{2^n}{n+1}$	$2k+1$	$q = \frac{2^n}{1 + C_n^1 + C_n^2 + \ldots + C_n^k}$

Іншим важливим параметром завадостійких кодів, який використовується у теорії кодування, є кількість контрольних символів ρ.

У теорії кодування вельми важливе значення мають щільноупаковані коди. Надамо відповідне визначення.

Визначення 2.4. Щільноупакованим кодом (англійський термін – close-packed code) називається завадостійкий код, для якого досягається мінімальна кількість кодових комбінацій, визначена співвідношенням (2.42).

Проте слід відзначити, що не завжди доцільно прагнути використовувати завадостійкі коди, які є близькими до оптимальних. Іншим важливим показником якості завадостійких кодів є технічна складність реалізації процесів їхнього кодування та декодування. Наведемо конкретні приклади, пов'язані із проектуванням систем зв'язку. Якщо необхідно передавати інформацію із

невисокою швидкістю з використанням надійного та коштовного каналу зв'язку, а кодувальні та декодувальні пристрої виконуються на швидкодійних та високонадійних електронних компонентах, тоді технічна складність реалізації такої системи зв'язку не грає особливої ролі. У такому випадку вирішальним фактором є підвищення ефективності використання лінії зв'язку, тому тут бажано використовувати оптимальні завадостійкі коди із мінімальним параметром надлишковості R_k. В іншому випадку, коли завадостійкий код застосовується в електронній системі, що виконана на компонентах, надійність та швидкодія роботи яких є близькою до відповідних параметрів кодувальної апаратури, тоді одним із критеріїв ефективності такого коду є надійність всієї системи, яка складається із виконавчого, кодувального та декодувального блоків. Тоді часто доцільним є використання кодів із більшою надлишковістю, які є більш простими з точки зору технічної реалізації. Саме така ситуація часто виникає у системах автоматичного керування технологічним обладнанням та технологічними процесами, реалізованими на основі обчислювальної електронної апаратури [31]. Теорія надійності зосереджених та розподілених електронних систем та відповідні способи оцінки цього параметру розглядалися у підрозділі 6.1.6 [49]. Важливими є також параметри завадостійких кодів, пов'язані із визначенням імовірності помилки заданої кратності за визначених умов формування сигналів. Тут найбільш важливими є принцип багаторазових випробувань Бернуллі та закони статистичної радіотехніки. З теоретичної точки зору імовірнісні параметри лінійних завадостійких кодів розглядалися у підрозділі 6.1.5 другої частини посібника. Способи оцінки імовірності заданої кількості помилок у кодовій комбінації з використанням схеми випробувань Бернуллі розглядалися у підрозділі 6.1.4, а способи оцінки імовірності спотворення цифрових сигналів з використанням законів статистичної радіотехніки – у підрозділі 6.5.2 другої частини посібника.

Враховуючи вищесказане, виділяють наступні стаціонарні параметри завадостійких кодів [2].

1. Кількість символів у коді, або довжина коду n.

2. Основа коду m, яка визначається кількістю символів абетки, які

використовуються для формування кодової комбінації. У цифрових сигналах, які використовуються в електронних системах, зазвичай використовуються двійкові коди, тобто, коди з основою $m = 2$.

3. Кількість інформаційних символів k.

4. Кількість контрольних символів r.

5. Повна кількість всіх можливих кодових комбінацій $N_0 = m^n$.

6. Кількість дозволених кодових комбінацій $N = m^k$. Цей параметр називають також потужністю коду.

7. Мінімальна кодова відстань $d_{\min}$ між двома дозволеними кодовими комбінаціями, яка визначається з використанням співвідношень (2.35) – (2.37).

Слід відзначити, що мінімальна кодова відстань $d_{\min}$ пов'язана із кодовою відстанню d, яка розглядається як параметр окремих кодових комбінацій, але мінімальне значення кодової відстані є узагальненим, або стаціонарним параметром систематичного блочного завадостійкого коду.

Розрізняють також відповідні параметри окремих кодових комбінацій.

1. Вага кодової комбінації w, яка відповідає кількості одиниць в ній.

2. Вага вектора помилки w_l, який визначається між двома заданими кодовими комбінаціями через їх сумування за модулем 2.

3. Кодова відстань d між двома заданими кодовими комбінаціями, яка визначається через їх сумування за модулем 2.

Із стаціонарними параметрами пов'язані параметри надлишковості коду, які визначаються співвідношеннями (2.38), (2.39).

1. Коефіцієнт надлишковості коду за параметром n R_n, який визначається співвідношенням (2.38).

2. Коефіцієнт надлишковості коду за параметром k R_k, який визначається співвідношенням (2.39).

Серед імовірнісних параметрів завадостійких кодів, які характерізують їх коректувальну здатність, виділяють наступні [2, 49].

1. Імовірність спотворення одного біту $p_б$.

2. Імовірність правильного приймання кодових комбінацій $P_п$.

3. Імовірність помилкового, або хибного приймання кодових комбінацій $P_{\text{х}}$.

Зрозуміло, що зв’язок між цими двома параметрами визначаєстья через умову нормування:

$$P_{\text{п}} = 1 - P_{\text{х}}. \tag{2.43}$$

4. Завадостійкість кода, яка залежить від імовірності правильного та хибного приймання кодових комбінацій та визначається наступним чином [49]:

$$S = \lg\left(\frac{1}{P_{\text{п}}}\right) = \lg\left(\frac{1}{1 - P_{\text{х}}}\right). \tag{2.44}$$

5. Коефіцієнт виявлення помилок, який визначається як співвідношення ймовірності виявленої помилки до ймовірності невиявленої помилки [49]:

$$K_{\text{вп}} = \frac{p_{\text{вп}}}{p_{\text{нп}}}, \qquad p_{\text{вп}} = 1 - p_{\text{нп}}, \tag{2.45}$$

де $p_{\text{вп}}$ – ймовірність виявленої помилки, $p_{\text{нп}}$ – ймовірність невиявленої помилки.

6. Коефіцієнт виправлення помилок, який визначається як співвідношення ймовірності виправленої помилки до ймовірності невиправленої помилки [49]:

$$K_{\text{впп}} = \frac{p_{\text{впп}}}{p_{\text{нвп}}}, \qquad p_{\text{нвп}} = 1 - p_{\text{впп}}, \tag{2.46}$$

де $p_{\text{впп}}$ – імовірність виправлення помилки, $p_{\text{нвп}}$ – імовірність не виправлення помилки.

Зв’язок між імовірністю бітовою помилки та узагальненими імовірнісними параметрами кодової комбінації визначається через схему повторних випробувань Бернуллі та був розглянутий у підрозділі 6.1.4 другої частини посібника [49].

Із основ теорії інформації, які були описані у першому та четвертому розділах першої частини посібника [48], зрозуміло, що у разі використання завадостійких кодів через канал зв’язку передається надлишкова інформація. Тобто, порівняно із натуральними та ефективними кодами, швидкість

передавання інформації зменшується. У зв'язку з цим розглядають також інформаційні параметри завадостійких кодів [5, 55, 62 – 64].

1. Відносна швидкість передавання інформації:

$$w_{n,k} = R_n = 1 - \frac{k}{n}. \tag{2.47}$$

2. Ефективність дії коректувального коду, яка визначається як:

$$K_{\text{кк}} = 10\lg \frac{\left(\frac{E_1}{N_v^*}\right)}{\left(\frac{E_1}{N_v}\right)}, \tag{2.48}$$

де E_1 – енергія одиничного сигналу на 1 біт, N_v – потужність шуму на 1 Гц смуги частот каналу зв'язку.

3. Нижня границя Варшамова – Гільберта, яка задається наступним співвідношенням [55, 62 – 64]:

$$R_n \geq H\left(\left|d_{\min} - 2\right|\right) = \left|d_{\min} - 2\right| \cdot \log_2\left(\left|d_{\min} - 2\right|\right), \tag{2.49}$$

або

$$r \geq \frac{n \cdot H\left(\left|d_{\min} - 2\right|\right)}{n-1}, \tag{2.50}$$

$$\frac{k}{n} \leq \frac{1 - H\left(\left|d_{\min} - 2\right|\right)}{n-1}. \tag{2.51}$$

Слід відзначити, що наведені співвідношення (2.49) – (2.51) для границі Варшамова – Гільберта у теорії кодування пов'язані із границею Хеммінга, яка визначається співвідношенням (2.42).

Узагальнена класифікаційна діаграма параметрів блокових систематичних завадостійких кодів наведена на рис. 2.4.

Розглянуті у цьому підрозділі параметри завадостійких кодів та співвідношення (2.38) – (2.51) широко використовуються на практиці для визначення інформаційних характеристик кодувальних електронних систем [2, 5, 33, 52].

Рис. 2.4 Узагальнена класифікація параметрів систематичних блокових завадостійких кодів

2.3 Завадостійкі коди із виявленням помилок

2.3.1 Коди із перевіркою на парність

Найпростішим завадостійким кодом, який дозволяє виявляти у кодовій послідовності одиночні помилки, є код із перевіркою на парність (англійський термін – parity-check code) [2 – 5, 64, 65]. Сутність цього способу кодування є загальновідомою та полягає у тому, що до інформаційних бітів, які передаються, додається один контрольний біт, значення якого обирається таким, щоб сума всіх бітів за модулем 2 дорівнювала 0. Тобто, кількість одиниць у кодовій послідовності має бути парною. Надлишковість коду із перевіркою на парність згідно із співвідношенням (2.39) становить $R_k = \frac{1}{k}$, а мінімальна кодова відстань між дозволеними комбінаціями складає 2. Це можна обґрунтувати на основі логічних міркувань наступним чином. Якщо дві кодові послідовності, які кодуються кодом із перевіркою на парність, відрізняються одна від одної одним бітом, тоді одна з них містить парну, а друга – непарну кількість одиниць, тобто, контрольні біти у таких двох послідовностях також відрізняються. Коректувальна здатність коду із перевіркою на парність полягає у виявленні одиночних помилок або помилок непарної кратності. Розглянемо простий приклад формування коду із перевіркою на парність.

Приклад 2.1. Побудувати код із перевіркою на парність для бітових послідовностей 1010 та 1011.

Бітові послідовності, які кодуються, мають чотири розряди. Для формування коду із перевіркою на парність в них необхідно додати один розряд, тоді загальна кількість розрядів буде складати 5. Оскільки число 1010 містить парну кількість одиниць, код із перевіркою на парність для цього числа буде 01010. Число 1011 містить непарну кількість одиниць, для нього код із перевіркою на парність становить 11011.

Незважаючи на те, що код із перевіркою на парність є досить простим, узагальнений метод, пов’язаний із перевіркою на парність контрольних сум, є вельми універсальним і широко використовується для формування контрольних сум в алгоритмах побудови лінійних кодів. Відповідний математичний апарат був розглянутий у підрозділі 5.4 другої частини посібника. Слід відзначити, що способи формування лінійних кодів розрізняються лише кількістю контрольних сум та методами їхнього формування, а розраховуються контрольні суми за однаковим алгоритмом – через сумування відповідних розрядів числа за модулем 2. Особливості виконання операції сумування двійкових чисел за модулем 2 для випадку, коли кількість бітів, які сумуються, є більшою за 2, були розглянуті у підрозділі 1.3.6 цієї частини посібника. Способи формування контрольних сум для кодів Хеммінга та інших типів лінійних кодів розглядатимуться у підрозділі 2.4.

2.3.2 Коди із повторенням елементів

Іншим кодом із виявленням помилок є код із удвоєнням елементів [2 – 5, 64, 65]. Сутність побудови цього коду полягає у тому, що під час його формування кожний біт послідовності, яка передається, повторюється двічі. Тобто, замість 0 передається послідовність бітів 00, а замість 1 – послідовність бітів 11. Розглянемо приклад побудови коду із удвоєнням елементів.

Приклад 2.2. Побудувати код із удвоєнням елементів для бітових послідовностей 1010 та 1011 та розрахувати параметр надлишковості цього коду.

Для даного прикладу зрозуміло, що код із удвоєнням елементів для числа 1010 становить 11001100, а для числа 1011 – 11001111. Для коду із удвоєнням елементів згідно із формулами (2.38), (2.39), коефіцієнт

надлишковості R_k становить 2, а коефіцієнт надлишковості R_n складає 0,5.

Коректувальна здатність коду із удвоєнням елементів, як і коду із перевіркою на парність, полягає у виявленні одиночних помилок, проте надлишковість такого коду є значно вищою.

Існують різні модифікації коду із удвоєнням елементів. Наприклад, замість послідовностей 00 для коду нуля та 11 для коду одиниці можуть використовуватись інші пари символів. Наприклад, для манчестерського коду, розглянутого у підрозділі 1.2.4.1, символу 1 відповідає послідовність бітів 01, а символу 0 – послідовність 10, а для різницевого манчестерського коду, розглянутого у підрозділі 1.2.4.2, символу 1 відповідає послідовність біт 11, а символу 0 – послідовність 01. Взагалі всі різновиди манчестерських кодів, розглянуті у підрозділі 1.2.4, можна розглядати як завадостійкі коди із виявленням помилок, хоча за принципом свого формування вони більш схожі на натуральні коди. Це зайвий раз свідчить про умовність класифікації цифрових кодів, наведеної у підрозділі 1.1.2, та класифікаційної діаграми, наведеної на рис. 1.2.

Код із удвоєнням елементів є різновидом більш широкого класу кодів із повторенням елементів (англійський термін – repetition code). Такі коди відрізняються тим, що в них кількість повторень одиниць та нулів є більшою за 2 та розглядається як один із параметрів формування коду. Якщо кількість повторень елементів становить l, співвідношення (2.38) та (2.39) можна переписати наступним чином:

$$R_n = \frac{l \cdot k - k}{l \cdot k} = \frac{k \cdot (l-1)}{l \cdot k} = \frac{l-1}{l},\ R_k = \frac{n-k}{k} = \frac{l \cdot k - k}{k} = \frac{k \cdot (l-1)}{k} = l - 1. \quad (2.52)$$

Слід відзначити, що у кодах із повторенням елементів у разі, якщо кількість повторень елементів l є великою величиною, можна також виправляти помилки. У цьому випадку правильним вважається символ, частота появи якого у ланцюжку коду із довжиною l є більшою. Наприклад,

якщо передана послідовність символів із восьми бітів 00000110, можна вважати, що це код символу 0. Цікавим є те, що коефіцієнти надлишковості коду із повторенням елементів не залежать від кількості інформаційних розрядів k, а однозначно визначаються кількістю повторень l.

Код із повторенням елементів також є теоретичним підґрунтям для способу мажоритарного декодування, який буде розглянутий у підрозділі 2.4.3.

Розглянемо приклад розрахунку коефіцієнтів надлишковості коду із повторенням елементів.

Приклад 2.3. Розрахувати коефіцієнти надлишковості коду із повторенням елементів за умови $l = 10$.

Із співвідношень (2.52) маємо:

$$R_n = \frac{l-1}{l} = 0{,}9\,;\; R_k = l - 1 = 9\,.$$

Перевагою кодів із повторенням елементів є простота їхньої побудови, проте надлишковість цих кодів R_k є вкрай великою величиною, яка збільшується із зростанням кількості повторень l. Це призводить до збільшення часу передавання повідомлень та до розширення частотної смуги спектрів сигналів. Одним із прикладів розширення спектрів сигналу за умови використання кодів із удвоєнням елементів є манчестерські коди, спектри яких розглядалися у підрозділі 1.2.9. У зв'язку з цим, коди із повторенням елементів широко використовувались у середині XX століття на початковому етапі розвитку систем зв'язку та обчислювальної техніки, коли об'єми інформації, які передавалися, ще були незначними, і висока надлишковість кодів із повторенням елементів не вважалась критичним параметром. У сучасних системах зв'язку зазвичай використовують лінійні та групові коди із контрольними сумами, які перевіряються на парність, оскільки надлишковість таких кодів є значно меншою [2 – 10, 64, 65]. Крім того, ефективними з точки зору технічної реалізації є циклічні коди, які також мають невисокий коефіцієнт надлишковості R_k. Способи побудови лінійних кодів розглядатимуться у підрозділі 2.4, циклічних – у підрозділі 2.5, а алгоритми формування групових кодів будуть розглянуті у розділі 3.

Розглянемо тепер спектри завадостійких кодів із виявленням помилок, для побудови яких скористаємося комп'ютерною програмою, наведеною у додатку В. Будемо вважати, що кодується послідовність бітів 1011 з використанням натурального коду, коду із перевіркою на парність, коду із

удвоєнням елементів та коду із повторенням елементів з параметром $l = 5$. Вважатимемо також, що для формування сигналу у каналі зв'язку використовується код AMI. Відповідні залежності для спектрів сигналів наведені на рис. 2.5.

На ріс 2.5 видно, що спектри сигналів натурального коду та коду із перевіркою на парність майже не відрізняються, а для кодів із повторенням елементів спектри стають більш вузькими та мають більшу енергію на ділянках, які відповідають передаванню великої кількості одиниць. Проте такі спектри мають гребінчастий характер і для приймання таких сигналів без помилок необхідно використовувати гребінчасті фільтри [2 – 6], що у значній мірі ускладнює приймальну апаратуру. Крім цього, час передавання сигналів, закодованих кодом із повторенням елементів, збільшується пропорційно кількості повторень, що у значній мірі збільшує час зайнятості каналу зв'язку [1 – 6, 8].

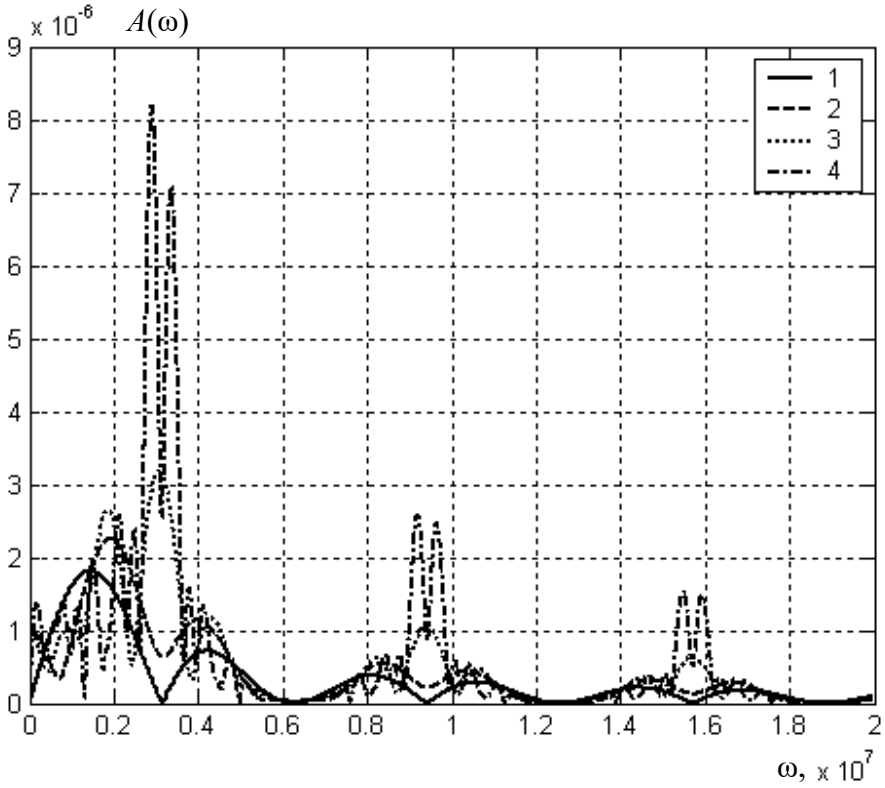


Рис. 2.5 Спектри сигналів для кодової послідовності 1011 за умови використання натурального коду (1), коду із перевіркою на парність (2), коду із удвоєнням елементів (3) та коду із повторенням елементів з параметром $l = 5$. Вважається, що для формування сигналу у каналі зв'язку застосований біполярний код AMI

2.3.3 Прямокутний код

Цікавим способом узагальнення методу кодування із перевіркою на парність є прямокутний код (англійський термін – rectangle code) [33]. Такий код дозволяє не лише визначати, але й виправляти одиночні помилки у інформаційних потоках досить великої потужності. Сутність прямокутного коду полягає у тому, що інформація, яка передається та перевіряється, подається у вигляді упорядкованої матриці, всі рядки та стовпчики якої перевіряються на парність. Для здійснення такої перевірки до матриці додаються один стовпчик та один рядок, де вписуються контрольні біти. Тоді, у разі спотворення одного біту такої матриці, одиниці в контрольних сумах вказують на номер стовпчика та номер рядка спотвореного елементу.

Якщо кількість рядків інформаційної матриці дорівнює m, а кількість стовпчиків n, тоді кількість елементів перевірочної матриці становить $N=(m+1)\cdot(n+1)$. Тоді, згідно із співвідношеннями (2.38), (2.39), коефіцієнти надлишковості прямокутного коду складають:

$$R_n=\frac{N-m\cdot n}{N}=\frac{(m+1)\cdot(n+1)-m\cdot n}{(m+1)\cdot(n+1)}=\frac{m+n+1}{(m+1)\cdot(n+1)}, \qquad (2.53)$$

$$R_k=\frac{(m+1)\cdot(n+1)-m\cdot n}{m\cdot n}=\frac{m+n+1}{m\cdot n}.$$

Розглянемо приклад формування прямокутного коду та обчислення його коефіцієнтів надлишковості.

Приклад 2.4. Каналом зв'язку з використанням прямокутного необхідно передати 8 байтів: 10010001, 10101101, 10101011, 11110000, 10101010, 10001000, 11111110, 10010011. Для таких умов сформувати матрицю прямокутного коду та обчислити його коефіцієнти надлишковості.

Зрозуміло, що матриця прямокутного коду має наступний вигляд:

$$\mathbf{M}=\begin{pmatrix}1&0&0&1&0&0&0&1&1\\1&0&1&0&1&1&0&1&1\\1&0&1&0&1&0&1&1&1\\1&1&1&1&0&0&0&0&0\\1&0&1&0&1&0&1&0&0\\1&0&0&0&1&0&0&0&0\\1&1&1&1&1&1&1&0&1\\1&0&0&1&0&0&1&1&0\\0&0&1&0&1&0&0&0&0\end{pmatrix}.$$

У визначеній матриці прямокутного коду **M** контрольні символи вписані в останній стовпчик та в останній рядок.

З використанням співвідношень (2.53) обчислимо коефіцієнти надлишковості отриманого прямокутного коду:

$$R_n = \frac{m+n+1}{(m+1)\cdot(n+1)} = \frac{8+8+1}{9\cdot 9} = \frac{17}{81} \approx 0{,}21.$$

$$R_k = \frac{m+n+1}{m\cdot n} = \frac{17}{8\cdot 8} = \frac{17}{64} = 0{,}265625.$$

Слід відзначити, що прямокутний код має досить невелику кількість контрольних розрядів відносно кількості інформаційних, тому його надлишковість є невисокою, особливо зважаючи на те, що він дозволяє виправляти одиночні помилки [33].

2.3.4 Інверсний код

Ще одним різновидом коду із удвоєнням елементів є інверсний код [2]. В основу побудови цього коду, як і коду з удвоєнням елементів, покладено принцип повторення інформаційного слова, яке кодується, проте спосіб формування інверсного коду залежить від кількості одиниць у ньому. Якщо кількість одиниць у вхідному інформаційному слові є парною, воно записується повторно без змін, а якщо парною – другий раз записуються інверсні значення бітів вхідного слова. Тобто, замість одиниць ставляться нулі, а замість нулів – одиниці. Наприклад, кодова комбінація 10010 записується у вигляді 1001010010, а кодова комбінація 10011 – у вигляді 1001101100. З точки зору описаного принципу побудови інверсного коду цікавим є те, що він завжди містить парну кількість одиниць.

Декодер інверсного коду працює наступним чином. Якщо перша прийнята кодова послідовність містить парну кількість одиниць, вона приймається повторно без інверсії бітів і прийняти послідовності порівнюються. А якщо перша послідовність має непарну кількість одиниць, друга послідовність приймається з інверсією бітів, і, знову ж таки, проводиться порівняння першої та другої послідовностей. Коректувальна здатність інверсного коду, як і коду із перевіркою на парність – виявлення

помилок непарної кратності. Схема цифрового кодера, який створює інверсний код, наведена на рис. 2.6, а, а схема декодера інверсного коду – на рис. 2.6, б.

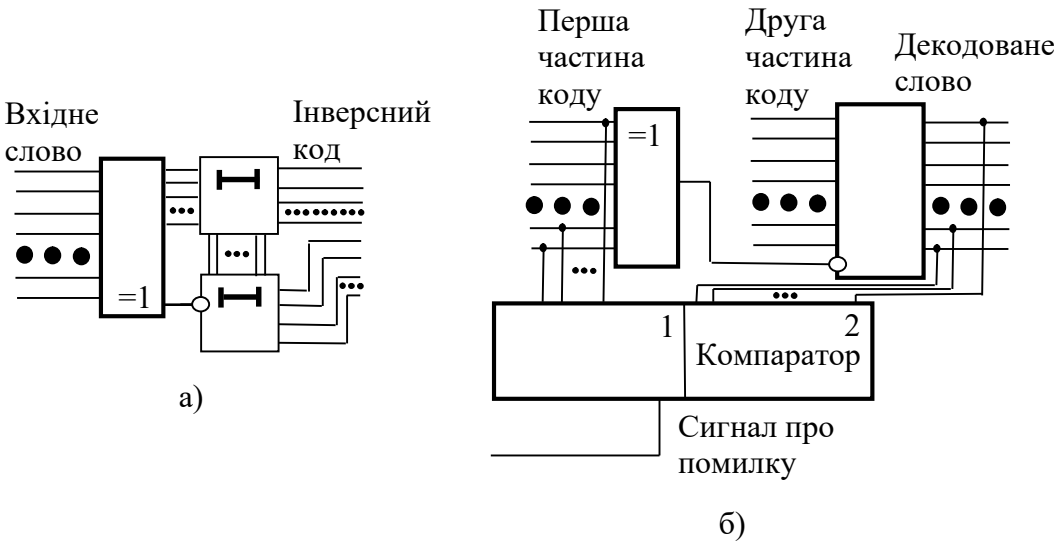


Рис. 2.6 Структурні схеми кодера (а) та декодера (б) інверсного коду

Розглянемо приклад побудови інверсного коду та проаналізуємо його частотний спектр.

Приклад 2.5. Побудувати код із перевіркою на парність та код з удвоєнням елементів для двійкової послідовності 10101. З використанням комп'ютерної програми, наведеної у додатку В, побудувати спектри для натурального коду цієї послідовності, її інверсного коду та коду із удвоєнням елементів. Розрахунки спектрів сигналів провести за умови, що для подання сигналу у каналі зв'язку використовується код AMI. Для коду із удвоєнням елементів та інверсного коду розрахувати коефіцієнти надлишковості.

Зрозуміло, що для кодової послідовності 10101 код із удвоєнням елементів записується у вигляді 1100110011, а інверсний код – у вигляді 1010101010.

Спектри отриманих кодових послідовностей наведені на рис. 2.7. аналіз спектрів показує, що спектр інверсного коду має найбільшу енергію на частоті, близькій до $0{,}1\cdot\omega$ та є найбільш вузькосмужним, тому з технічної точки зору найпростіше створити сприятливі умови саме для передавання без спотворень такого сигналу. В даного прикладу вузкосмужність спектру інверсного коду пов'язана з тим, що кодова послідовність 1010101010 є періодичною.

Щодо коефіцієнтів надлишковості, то вони для інверсного коду та для коду із удвоєнням елементів є однаковими та, згідно із співвідношеннями (2.52), становлять $R_n = 0{,}5$ та $R_k = 1$.

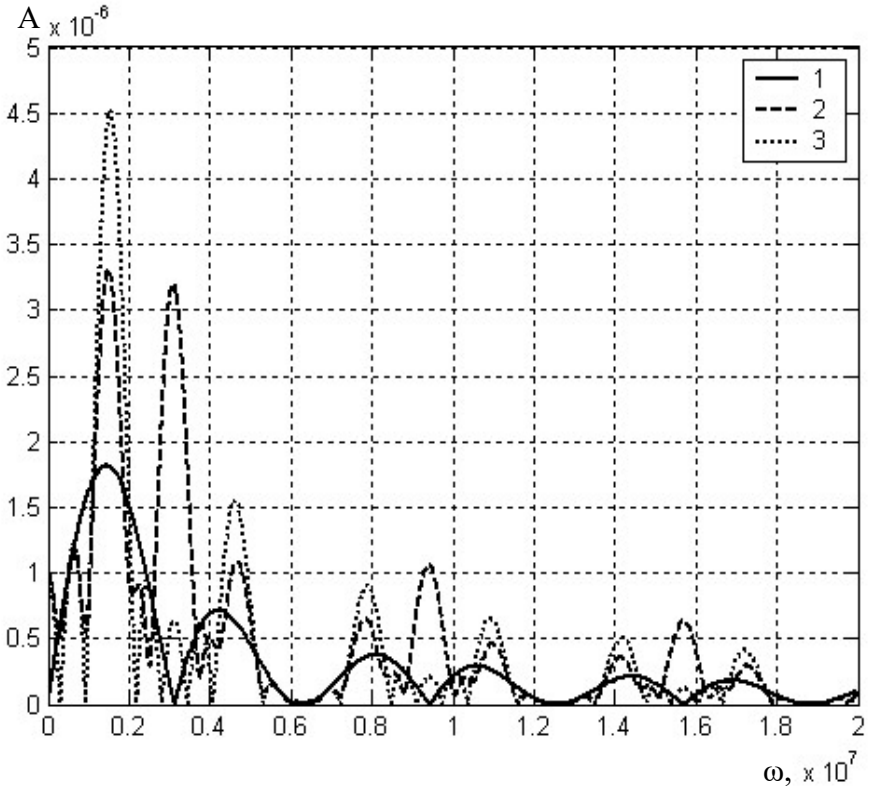


Рис. 2.7 Частотні спектри для двійкової послідовності 10101 за умови використання натурального коду (1), коду із удвоєнням елементів (2) та інверсного коду (3). Вважається, що для канального кодування сигналів використаний код AMI

Слід відзначити, що розглянуті у цьому підрозділі код із перевіркою на парність та код із повторенням елементів не лише широко використовуються на практиці в цифровій електронній апаратурі, але й мають також вельми велике теоретичне значення для формування базових принципів побудови лінійних та циклічних завадостійких кодів. Принцип перевірки на парність використовується для формування контрольних сум лінійних кодів, а правило повторення символів становить теоретичне підґрунтя для метода мажоритарного декодування, який широко застосовується у сучасній кодувальній електронній апаратурі. Відповідні способи формування завадостійких кодів та декодування їх кодових комбінацій розглядатимуться у підрозділі 2.4.

2.4 Лінійні коди та коди Хеммінга

2.4.1 Загальний принцип побудови коду Хеммінга із та приклади його формування

Перед вивченням цього підрозділу необхідно повторити перший, четвертий п'ятий та сьомий розділи другої частини посібника

Загалом способи побудови лінійних завадостійких кодів базуються на теорії лінійних векторних та матричних перетворень, яка розглядалася у п'ятому розділі другої частини посібника [48]. Основою цієї теорії є розробка алгоритмів побудови контрольних сум, в які входять інформаційні та контрольні розряди кодованого числа. Також для формування та для декодування блокових лінійних кодів ефективно використовуються математичний апарат матричних відносин, зокрема породжувальні та перевірочні матриці. Відповідні теоретичні відомості та способи формування лінійних кодів були розглянуті у підрозділі 5.4 другої частини посібника.

Розглянемо спочатку алгоритм формування лінійного коду Хеммінга у спрощеній формі, а саме у вигляді тезових формулювань. Точніше, будемо формулювати цей алгоритм через ставлення послідовних питань та формування відповідей на ці питання. Такий підхід до розгляду принципу побудови коду Хеммінга був використаний у навчальному посібнику [16]. Згідно із загальною структурою цього посібника будемо розглядати деякі відповіді на поставлені питання як властивості коду Хеммінга.

По-перше слід відзначити, що коди Хеммінга є одними із найбільш поширених. Вони широко використовують у сучасній кодувальній електронній апаратурі та існує багато їхніх різновидів [2 – 5, 33, 52 – 57, 64, 65]. Зазичай це коди або з виправленням одиночних, або з виправленням одиночних та виявленням подвійних помилок. Тобто, згідно із співвідношеннями (2.36), (2.37), для кодів Хеммінга $d_{\min} = 3$ або $d_{\min} = 4$.

Розрядність коду Хеммінга n обирається умови (2.42):

$$2^k \le \frac{2^n}{1+n},$$

або у логарифмічній формі:

$$k \le n - \log_2(1+n).$$

Залежність $k(n)$, задана співвідношенням (2.42), отримана з використанням графічних засобів системи науково-технічних розрахунків MatLab, показана на рис. 2.8.

Коди Хеммінга, порівняно із іншими лінійними завадостійкими кодами, мають наступні переваги.

1. Досить простий, наочний та ефективний алгоритм формування.

2. Малий коефіцієнт надлишковості за умови великої кількості інформаційних розрядів.

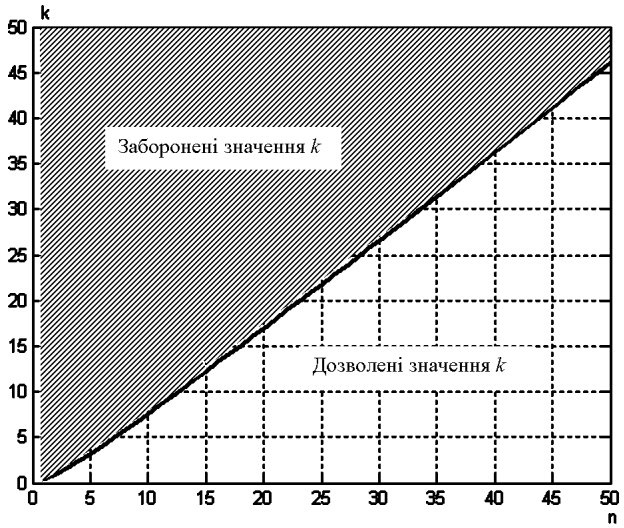


Рис. 2.8 Залежність загальної кількості розрядів коду Хеммінга від кількості інформаційних розрядів згідно із співвідношенням (2.42)

Завдяки вказаним перевагам саме коди Хеммінга знаходять сьогодні широке впровадження у різноманітних інформаційних системах та системах зв'язку.

Хоча загальні основи побудови коду Хеммінга є досить простими, проте алгоритм їх формування містить декілька важливих ітераційних кроків. Тому відразу будемо розглядати його не в абстрактному вигляді, а на прикладі побудови коду конкретного восьмирозрядного двійкового числа 10010011. Розглядаючи послідовність дій, виконання яких є необхідним для побудови коду Хеммінга, будемо почергово відповідати на прості та конкретні питання.

1. *Скільки необхідно взяти контрольних розрядів?*

Можна для відповіді на це питання скористатися співвідношенням (2.42) та графічною залежністю, наведеною на рис. 2.8, проте краще

запам’ятати дуже просте правило, яке сформулюємо як властивість коду Хеммінга.

Властивість 2.1. Кількість контрольних розрядів у коді Хеммінга повинна бути такою, щоб у них можна було записати номер будь-якої позиції отриманого коду у двійковому форматі, але і не більшою.

Наприклад, у нашому випадку, для числа 10010011, трьох контрольних розрядів буде замало. Дійсно, вже для двійкового запису числа 8, яке визначає кількість інформаційних розрядів k, необхідно 4 розряди, а нам слід ввести ще контрольні символи. Проте кількість контрольних розрядів $\rho = 4$ буде достатньою. Дійсно, за умови чотирьох контрольних символах довжина коду становитиме $n = k + \rho = 12$, і у такій кількості розрядів можна записати номер будь-якої позиції отриманого коду у двійковому форматі.

2. *Як розташувати контрольні та інформаційні розряди у запису коду?*

Тут також достатньо запам’ятати одне дуже просте правило.

Властивість 2.2. Контрольні розряди у коді Хеммінга розташовуються на позиціях, номера яких відповідають степені числа 2, тобто на першій, другий, четвертій, восьмій і т.д. позиціях. Інформаційні розряди числа ставляться на вільних позиціях між контрольними, а порядок їх слідування не змінюється відносно натурального коду числа.

Взявши до уваги дві вище сформульованих властивості, можна зробити дуже простий шаблон для побудови коду Хеммінга, який наведений у таблиці 2.2.

Таблиця 2.2 – Ілюстрація принципу формування коду Хеммінга

Номера позицій	n_{12}	n_{11}	n_{10}	n_9	n_8	n_7	n_6	n_5	n_4	n_3	n_2	n_1
Символи	k_8	k_7	k_6	k_5	r_4	k_4	k_3	k_2	r_3	k_1	r_2	r_1
Значення	1	0	0	1	r_4	0	0	1	r_3	1	r_2	r_1

У наведеній таблиці через r позначені контрольні розряди числа, а через k – інформаційні, які, до речі, вже є відомими і вписані в останньому рядку таблиці. Нижні індекси визначають номер розряду. Залишається

визначити контрольні розряди коду, який формується, через контрольні суми.

3. *Яким чином формуються контрольні суми?*

Спосіб формування чотирьох контрольних сум коду Хеммінга, S_1, S_2, S_3 та S_4, легко зрозуміти із таблиці 2.3. Тут записані двійкові коди чисел від 1 до 12, які відповідають номерам розрядів коду Хеммінга, що створюється. Для того, щоб сформувати контрольну суму S_1, необхідно проаналізувати останній стовпчик таблиці 2.3 і вписати до контрольної суми ті номери розрядів, двійковий запис яких у останній колонці має 1. Тобто, для першої контрольної суми S_1 маємо:

$$S_1 = n_1 \oplus n_3 \oplus n_5 \oplus n_7 \oplus n_9 \oplus n_{11}. \qquad (2.54)$$

Таблиця 2.3 – Ілюстрація принципу обчислення контрольних сум коду Хеммінга

Номер розряду	S_4	S_3	S_2	S_1
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0

Аналогічним чином, через другу колонку справа, формуємо другу

контрольну суму, включаючи до неї тільки ті розряди коду, у яких двійковий запис номеру містить 1 на другій позиції.

$$S_2 = n_2 \oplus n_3 \oplus n_6 \oplus n_7 \oplus n_{10} \oplus n_{11}. \quad (2.55)$$

Аналогічним способом формуємо третю та четверту контрольні суми:

$$S_3 = n_4 \oplus n_5 \oplus n_6 \oplus n_7 \oplus n_{12}, \quad (2.56)$$

$$S_4 = n_8 \oplus n_9 \oplus n_{10} \oplus n_{11} \oplus n_{12}. \quad (2.57)$$

4. *Як через контрольні суми визначаються контрольні розряди?*

Перепишемо контрольні суми (2.54) – (2.57), підставивши до цих виразів відомі значення інформаційних розрядів та номери контрольних розрядів, які подані у таблиці 2.2, і прирівняємо контрольні суми до нуля. Це означає, що всі чотири контрольні суми повинні містити парну кількість одиниць. Відповідно, можна записати наступні вирази:

$S_1 = n_1 \oplus n_3 \oplus n_5 \oplus n_7 \oplus n_9 \oplus n_{11} = r_1 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0$. Тобто, $r_1 = 1$.

$S_2 = n_2 \oplus n_3 \oplus n_6 \oplus n_7 \oplus n_{10} \oplus n_{11} = r_2 \oplus 1 \oplus 0 \oplus 0 \oplus 0 \oplus 0$. Тобто, $r_2 = 1$.

$S_3 = n_4 \oplus n_5 \oplus n_6 \oplus n_7 \oplus n_{12} = r_3 \oplus 1 \oplus 0 \oplus 0 \oplus 1$. Тобто, $r_3 = 0$.

$S_4 = n_8 \oplus n_9 \oplus n_{10} \oplus n_{11} \oplus n_{12} = r_4 \oplus 1 \oplus 0 \oplus 0 \oplus 1$. Тобто, $r_4 = 0$.

Таким чином, контрольні розряди визначені, код Хеммінга побудований, і таблицю 2.2, з урахуванням отриманих значень контрольних розрядів, можна переписати. Відповідні числові дані систематизовані у таблиці 2.4.

Таблиця 2.4 – Значення інформаційних та контрольних розрядів коду Хеммінга для числа 10010011

Номера позицій	n_{12}	n_{11}	n_{10}	n_9	n_8	n_7	n_6	n_5	n_4	n_3	n_2	n_1
Символи	k_8	k_7	k_6	k_5	r_4	k_4	k_3	k_2	r_3	k_1	r_2	r_1
Значення	1	0	0	1	0	0	0	1	0	1	1	1

Тобто, код Хеммінга для числа 10010011 становить 100100010111.

5. *Яким чином у коді Хеммінга виправляються помилки, що виникають*

при передаванні даних?

Проаналізуємо коректувальну здатність отриманого коду Хеммінга. Припустимо, що має місце одиночна помилка у сьомому розряді коду: замість 0 там стоїть 1, тобто, замість послідовності 100100010111 прийнято хибне повідомлення 100101010111. Необхідно перерахувати контрольні суми S_1, S_2, S_3 та S_4. Із співвідношень (2.54) – (2.57) маємо:

$S_1 = n_1 \oplus n_3 \oplus n_5 \oplus \overline{n_7} \oplus n_9 \oplus n_{11} = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0$. Тобто, $S_1 = 1$.

$S_2 = n_2 \oplus n_3 \oplus n_6 \oplus \overline{n_7} \oplus n_{10} \oplus n_{11} = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 0$. Тобто, $S_2 = 1$.

$S_3 = n_4 \oplus n_5 \oplus n_6 \oplus \overline{n_7} \oplus n_{12} = 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1$. Тобто, $S_3 = 1$.

$S_4 = n_8 \oplus n_9 \oplus n_{10} \oplus n_{11} \oplus n_{12} = 0 \oplus 1 \oplus 0 \oplus 0 \oplus 1$. Тобто, $S_4 = 0$.

Якщо прочитати отриманий двійковий код, починаючи із першої контрольної суми і закінчуючи четвертою, одержимо число $0111_2 = 7_{10}$. Тобто, через аналіз контрольних сум коду Хеммінга ми обчислили номер розряду, у якому виникла помилка.

Важливою особливістю коду Хеммінга є те, що одноразові помилки однаково виправляються як у інформаційних, так і у контрольних розрядах коду.

Наприклад, припустимо, що помилка виникла у четвертому розряді числа. Тоді контрольні суми $S_1 – S_4$ переписуються наступним чином:

$S_1 = n_1 \oplus n_3 \oplus n_5 \oplus n_7 \oplus n_9 \oplus n_{11} = 1 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0$. Тобто, $S_1 = 0$.

$S_2 = n_2 \oplus n_3 \oplus n_6 \oplus n_7 \oplus n_{10} \oplus n_{11} = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 \oplus 0$. Тобто, $S_2 = 0$.

$S_3 = \overline{n_4} \oplus n_5 \oplus n_6 \oplus n_7 \oplus n_{12} = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 1$. Тобто, $S_3 = 1$.

$S_4 = n_8 \oplus n_9 \oplus n_{10} \oplus n_{11} \oplus n_{12} = 0 \oplus 1 \oplus 0 \oplus 0 \oplus 1$. Тобто, $S_4 = 0$.

Надамо відповідне визначення.

Визначення 2.5. Завадостійкий код, у якому виправляються помилки як в інформаційних, так і в контрольних розрядах, називається рівномірно захищеним [2 – 5, 64, 65].

За значеннями контрольних сум отримуємо двійковий код $0100_2 = 4_{10}$, тобто, значення розряду, у якому виникла помилка, і в цьому випадку

знайдене правильно.

Слід відзначити, що класичний код Хеммінга має кодову відстань $d_{\min} = 3$ та не визначає подвійні помилки.

Припустимо, наприклад, що за умови активної зовнішньої завади виникла помилка у шостому та сьомому розрядах коду, тобто замість послідовності 100100010111 прийнята послідовність 100101110111. Обчислення контрольних сум (2.54) – (2.57) дає у даному випадку наступний результат:

$S_1 = n_1 \oplus n_3 \oplus n_5 \oplus \overline{n_7} \oplus n_9 \oplus n_{11} = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0$. Тобто, $S_1 = 1$.

$S_2 = n_2 \oplus n_3 \oplus \overline{n_6} \oplus \overline{n_7} \oplus n_{10} \oplus n_{11} = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 \oplus 0$. Тобто, $S_2 = 0$.

$S_3 = n_4 \oplus n_5 \oplus \overline{n_6} \oplus \overline{n_7} \oplus n_{12} = 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1$. Тобто, $S_3 = 0$.

$S_4 = n_8 \oplus n_9 \oplus n_{10} \oplus n_{11} \oplus n_{12} = 0 \oplus 1 \oplus 0 \oplus 0 \oplus 1$. Тобто, $S_4 = 0$.

Тобто, у даному випадку перевірка контрольних сум дає значення $1000_2 = 8_{10}$. Тоді, змінюючи восьмий розряд числа з 0 на 1, переходимо до дозволеної, але хибної кодової комбінації 100111110111.

Для закріплення наведених у цьому підрозділі теоретичних положень розглянемо приклад формування коду Хеммінга для послідовності із 12 біт.

Приклад 2.6. Побудувати код Хеммінга для послідовності бітів 110101100101. Проаналізувати роботу коду у випадку виникнення одиночної помилки у першому розряді сформованого коду, у десятому розряді, та у разі одночасного виникнення двох помилок, у четвертому та п’ятому розрядах коду.

Двійкове число 110101100101, яке кодується, має 12 інформаційних розрядів. Визначимо необхідну кількість контрольних розрядів згідно із властивістю 2.1. Видно, що чотирьох контрольних розрядів у даному випадку недостатньо. Дійсно, загальна кількість розрядів у коді за таких умов становитиме 12 + 4 = 16, а число 16 у двійковій системі містить не 4, а 5 розрядів. Тобто, за умови чотирьох контрольних розрядів та чотирьох контрольних сум остання, шістнадцята позиція коду не може бути пронумерованою. Тому мінімальна кількість контрольних розрядів для прикладу, який розглядається, є 5, і, згідно із властивістю 2.2, розташовуються вони на першій, другій, четвертій, восьмій та шістнадцятій позиціях коду. Порозрядна структура коду Хеммінга, який будується, без значень контрольних розрядів, наведена у таблиці 2.5.

Таблиця 2.5 – Узагальнена структура коду Хеммінга для прикладу 2.6

Номера позицій	n_9	n_8	n_7	n_6	n_5	n_4	n_3	n_2	n_1
Символи	k_5	r_4	k_4	k_3	k_2	r_3	k_1	r_2	r_1
Значення	0	–	0	1	0	–	1	–	–
Номера позицій	–	n_{17}	n_{16}	n_{15}	n_{14}	n_{13}	n_{12}	n_{11}	n_{10}
Символи	–	k_{12}	r_5	k_{11}	k_{10}	k_9	k_8	k_7	k_6
Значення	–	1	–	1	0	1	0	1	1

Принцип обчислення контрольних сум у даному випадку також буде трішки інакшим, його легко зрозуміти із таблиці 2.6. Згідно із таблицею 2.6 для обчислення контрольних сум можна записати наступні вирази.

Таблиця 2.6 – Ілюстрація принципу обчислення контрольних сум сімнадцятирозрядного коду Хеммінга для прикладу 2.6

Номер розряду	S_5	S_4	S_3	S_2	S_1
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	1	0	0
5	0	0	1	0	1
6	0	0	1	1	0
7	0	0	1	1	1
8	0	1	0	0	0
9	0	1	0	0	1
10	0	1	0	1	0
11	0	1	0	1	1
12	0	1	1	0	0
13	0	1	1	0	1
14	0	1	1	1	0
15	0	1	1	1	1
16	1	0	0	0	0
17	1	0	0	0	1

$$S_1 = n_1 \oplus n_3 \oplus n_5 \oplus n_7 \oplus n_9 \oplus n_{11} \oplus n_{13} \oplus n_{15} \oplus n_{17}. \quad (2.58)$$

$$S_2 = n_2 \oplus n_3 \oplus n_6 \oplus n_7 \oplus n_{10} \oplus n_{11} \oplus n_{14} \oplus n_{15}. \quad (2.59)$$

$$S_3 = n_4 \oplus n_5 \oplus n_6 \oplus n_7 \oplus n_{12} \oplus n_{13} \oplus n_{14} \oplus n_{15}. \quad (2.60)$$

$$S_4 = n_8 \oplus n_9 \oplus n_{10} \oplus n_{11} \oplus n_{12} \oplus n_{13} \oplus n_{14} \oplus n_{15}. \quad (2.61)$$

$$S_5 = n_{16} \oplus n_{17}. \quad (2.62)$$

Тоді, згідно із співвідношеннями (2.58) – (2.62) та із значеннями інформаційних розрядів, які наведені у таблиці 2.5, значення контрольних розрядів обчислюються наступним чином:

$S_1 = n_1 \oplus n_3 \oplus n_5 \oplus n_7 \oplus n_9 \oplus n_{11} \oplus n_{13} \oplus n_{15} \oplus n_{17} =$

$= r_1 \oplus k_1 \oplus k_2 \oplus k_4 \oplus k_5 \oplus k_7 \oplus k_9 \oplus k_{11} \oplus k_{12} =$

$= r_1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1$. Тобто, $r_1 = 1$.

$S_2 = n_2 \oplus n_3 \oplus n_6 \oplus n_7 \oplus n_{10} \oplus n_{11} \oplus n_{14} \oplus n_{15} =$

$= r_2 \oplus k_1 \oplus k_3 \oplus k_4 \oplus k_6 \oplus k_7 \oplus k_{10} \oplus k_{11} =$

$= r_2 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1$. Тобто, $r_2 = 1$.

$S_3 = n_4 \oplus n_5 \oplus n_6 \oplus n_7 \oplus n_{12} \oplus n_{13} \oplus n_{14} \oplus n_{15} =$

$= r_3 \oplus k_2 \oplus k_3 \oplus k_4 \oplus k_8 \oplus k_9 \oplus k_{10} \oplus k_{11} =$

$= r_3 \oplus 0 \oplus 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1$. Тобто, $r_3 = 1$.

$S_4 = n_8 \oplus n_9 \oplus n_{10} \oplus n_{11} \oplus n_{12} \oplus n_{13} \oplus n_{14} \oplus n_{15} =$

$= r_4 \oplus k_5 \oplus k_6 \oplus k_7 \oplus k_8 \oplus k_9 \oplus k_{10} \oplus k_{11} =$

$= r_4 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1$. Тобто, $r_4 = 0$.

$S_5 = n_{16} \oplus n_{17} = r_5 \oplus k_{12} = r_5 \oplus 1$. Тобто, $r_5 = 1$.

Обчисленні значення всіх інформаційних та контрольних розрядів сімнадцятирозрядного коду Хеммінга наведені у таблиці 2.7. Із результатів, наведених у таблиці, видно, що код Хеммінга для дванадцятирозрядного числа 110101100101 становить 11101011000101111.

Таблиця 2.7 – Розташування інформаційних та контрольних розрядів коду Хеммінга для прикладу 2.6

Номера позицій	n_9	n_8	n_7	n_6	n_5	n_4	n_3	n_2	n_1
Символи	k_5	r_4	k_4	k_3	k_2	r_3	k_1	r_2	r_1
Значення	0	0	0	1	0	1	1	1	1
Номера позицій	–	n_{17}	n_{16}	n_{15}	n_{14}	n_{13}	n_{12}	n_{11}	n_{10}
Символи	–	k_{12}	r_5	k_{11}	k_{10}	k_9	k_8	k_7	k_6
Значення	–	1	1	1	0	1	0	1	1

Розглянемо тепер коректувальну здатність отриманого коду. Припустимо, що помилка виникла у першому розряді сформованого коду. Згідно із таблицею 2.7 це контрольний розряд r_1. Для цього випадку контрольні суми (2.58) – (2.62) перепишуться наступним чином:

$$S_1 = \overline{n_1} \oplus n_3 \oplus n_5 \oplus n_7 \oplus n_9 \oplus n_{11} \oplus n_{13} \oplus n_{15} \oplus n_{17} =$$
$$= \overline{r_1} \oplus 1 \oplus 0 \oplus 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 =$$
$$= 0 \oplus 1 \oplus 0 \oplus 0 \oplus 0_5 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 1.$$

$$S_2 = n_2 \oplus n_3 \oplus n_6 \oplus n_7 \oplus n_{10} \oplus n_{11} \oplus n_{14} \oplus n_{15} =$$
$$= r_2 \oplus k_1 \oplus k_3 \oplus k_4 \oplus k_6 \oplus k_7 \oplus k_{10} \oplus k_{11} = 0.$$

$$S_3 = n_4 \oplus n_5 \oplus n_6 \oplus n_7 \oplus n_{12} \oplus n_{13} \oplus n_{14} \oplus n_{15} =$$
$$= r_3 \oplus k_2 \oplus k_3 \oplus k_4 \oplus k_8 \oplus k_9 \oplus k_{10} \oplus k_{11} = 0.$$

$$S_4 = n_8 \oplus n_9 \oplus n_{10} \oplus n_{11} \oplus n_{12} \oplus n_{13} \oplus n_{14} \oplus n_{15} =$$
$$= r_4 \oplus k_5 \oplus k_6 \oplus k_7 \oplus k_8 \oplus k_9 \oplus k_{10} \oplus k_{11} = 0.$$

$$S_5 = n_{16} \oplus n_{17} = r_5 \oplus k_{12} = 0.$$

Тобто, за контрольними сумами отримуємо двійковий код 00001, який відповідає першому розряду кодової комбінації. Можна зробити висновок, що для даного випадку код працює правильно.

Припустимо тепер, що помилка виникла у десятому розряді коду, у якому розташований інформаційний символ k_6. Для цього випадку, відповідно, маємо такі контрольні суми:

$$S_1 = n_1 \oplus n_3 \oplus n_5 \oplus n_7 \oplus n_9 \oplus n_{11} \oplus n_{13} \oplus n_{15} \oplus n_{17} =$$
$$= r_1 \oplus k_1 \oplus k_2 \oplus k_4 \oplus k_5 \oplus k_7 \oplus k_9 \oplus k_{11} \oplus k_{12} =$$
$$= 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 0.$$

$$S_2 = n_2 \oplus n_3 \oplus n_6 \oplus n_7 \oplus \overline{n_{10}} \oplus n_{11} \oplus n_{14} \oplus n_{15} =$$
$$= r_2 \oplus k_1 \oplus k_3 \oplus k_4 \oplus \overline{k_6} \oplus k_7 \oplus k_{10} \oplus k_{11} =$$
$$= 1 \oplus 1 \oplus 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1 = 1.$$

$$S_3 = n_4 \oplus n_5 \oplus n_6 \oplus n_7 \oplus n_{12} \oplus n_{13} \oplus n_{14} \oplus n_{15} =$$
$$= r_3 \oplus k_2 \oplus k_3 \oplus k_4 \oplus k_8 \oplus k_9 \oplus k_{10} \oplus k_{11} =$$
$$= 1 \oplus 0 \oplus 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1 = 0.$$

$$S_4 = n_8 \oplus n_9 \oplus \overline{n_{10}} \oplus n_{11} \oplus n_{12} \oplus n_{13} \oplus n_{14} \oplus n_{15} =$$
$$= r_4 \oplus k_5 \oplus \overline{k_6} \oplus k_7 \oplus k_8 \oplus k_9 \oplus k_{10} \oplus k_{11} =$$
$$= 0 \oplus 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 = 1.$$
$$S_5 = n_{16} \oplus n_{17} = r_5 \oplus k_{12} = 1 \oplus 1 = 0.$$

Тобто, за контрольними сумами отримуємо двійковий код 01010, який відповідає десятому розряду кодової комбінації. Можна зробити висновок, що і у цьому випадку код працює правильно.

Тепер припустимо, що виникла подвійна помилка у четвертому та п'ятому розрядах кодової комбінації, яким відповідають символи r_3 та k_2. Через обчислення контрольних сум, заданих співвідношеннями (2.58) – (2.62), маємо такий результат:

$$S_1 = n_1 \oplus n_3 \oplus \overline{n_5} \oplus n_7 \oplus n_9 \oplus n_{11} \oplus n_{13} \oplus n_{15} \oplus n_{17} =$$
$$= r_1 \oplus k_1 \oplus \overline{k_2} \oplus k_4 \oplus k_5 \oplus k_7 \oplus k_9 \oplus k_{11} \oplus k_{12} =$$
$$= 1 \oplus 1 \oplus 1 \oplus 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 1.$$
$$S_2 = n_2 \oplus n_3 \oplus n_6 \oplus n_7 \oplus n_{10} \oplus n_{11} \oplus n_{14} \oplus n_{15} =$$
$$= r_2 \oplus k_1 \oplus k_3 \oplus k_4 \oplus k_6 \oplus k_7 \oplus k_{10} \oplus k_{11} =$$
$$= 1 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 0.$$
$$S_3 = \overline{n_4} \oplus \overline{n_5} \oplus n_6 \oplus n_7 \oplus n_{12} \oplus n_{13} \oplus n_{14} \oplus n_{15} =$$
$$= \overline{r_3} \oplus \overline{k_2} \oplus k_3 \oplus k_4 \oplus k_8 \oplus k_9 \oplus k_{10} \oplus k_{11} =$$
$$= 0 \oplus 1 \oplus 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1 = 0.$$
$$S_4 = n_8 \oplus n_9 \oplus n_{10} \oplus n_{11} \oplus n_{12} \oplus n_{13} \oplus n_{14} \oplus n_{15} =$$
$$= r_4 \oplus k_5 \oplus k_6 \oplus k_7 \oplus k_8 \oplus k_9 \oplus k_{10} \oplus k_{11} =$$
$$= 0 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 = 0.$$
$$S_5 = n_{16} \oplus n_{17} = r_5 \oplus k_{12} = 1 \oplus 1 = 0.$$

Тобто, у цьому випадку аналіз коду вказує на помилку у першому розряді, що відповідає хибному переходу до дозволеної кодової комбінації, оскільки насправді, за умовою задачі, помилка виникла в четвертому та п'ятому розрядах кодової комбінації.

Приклад 2.7. Побудувати код Хеммінга для послідовності бітів 11010110 та з використанням програми, наведеної у додатку В, знайти спектри початкового та закодованого двійкових сигналів, вважаючи, що для подання сигналу використовується код AMI. Порівняти отримані спектри та пояснити отриманий результат.

Для пошуку коду Хеммінга восьмирозрядного числа скористаємося таблицею 2.2, в яку будемо заносити нові значення інформаційних та контрольних розрядів, та співвідношеннями (2.54) – (2.57).

Перепишемо таблицю 2.2 для формування коду восьмирозрядного двійкового числа 11010110, результат цього перетворення показаний у таблиці 2.8.

Таблиця 2.8 – Розташування інформаційних та контрольних розрядів коду Хеммінга для прикладу 2.7

Номера позицій	n_{12}	n_{11}	n_{10}	n_9	n_8	n_7	n_6	n_5	n_4	n_3	n_2	n_1
Символи	k_8	k_7	k_6	k_5	r_4	k_4	k_3	k_2	r_3	k_1	r_2	r_1
Значення	1	1	0	1	r_4	0	1	1	r_3	0	r_2	r_1

З урахуванням співвідношень (2.54) – (2.57), контрольні суми у даному випадку мають вигляд наступний.

$S_1 = n_1 \oplus n_3 \oplus n_5 \oplus n_7 \oplus n_9 \oplus n_{11} = r_1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 0$. Тобто, $r_1 = 0$.

$S_2 = n_2 \oplus n_3 \oplus n_6 \oplus n_7 \oplus n_{10} \oplus n_{11} = r_2 \oplus 0 \oplus 1 \oplus 0 \oplus 0 \oplus 1$. Тобто, $r_2 = 0$.

$S_3 = n_4 \oplus n_5 \oplus n_6 \oplus n_7 \oplus n_{12} = r_3 \oplus 1 \oplus 1 \oplus 0 \oplus 1$. Тобто, $r_3 = 1$.

$S_4 = n_8 \oplus n_9 \oplus n_{10} \oplus n_{11} \oplus n_{12} = r_4 \oplus 1 \oplus 0 \oplus 1 \oplus 1$. Тобто, $r_4 = 1$.

Тобто, згідно із таблицею 2.8, код Хеммінга для восьмирозрядного числа 11010110 становить 110110111000.

Тепер, за умовою задачі, з використанням програми, наведеної у додатку В, необхідно побудувати спектри сигналів двійкових послідовностей

11010110 та 110110111000, вважаючи, що для подання цих сигналів у цифровій формі використовується код AMI.

Для цього необхідно використати наступні командні рядки системи MatLab [13, 14].

```
>> e=[1,1,0,1,0,1,1,0];
>> digspectrum(e,3);
>> hold on;
>> f = [1,1,0,1,1,0,1,1,1,0,0,0];
>> digspectrum(f,3);
```

Графічні залежності, отримані як результат виконання цих командних рядків, наведені на рис. 2.9.

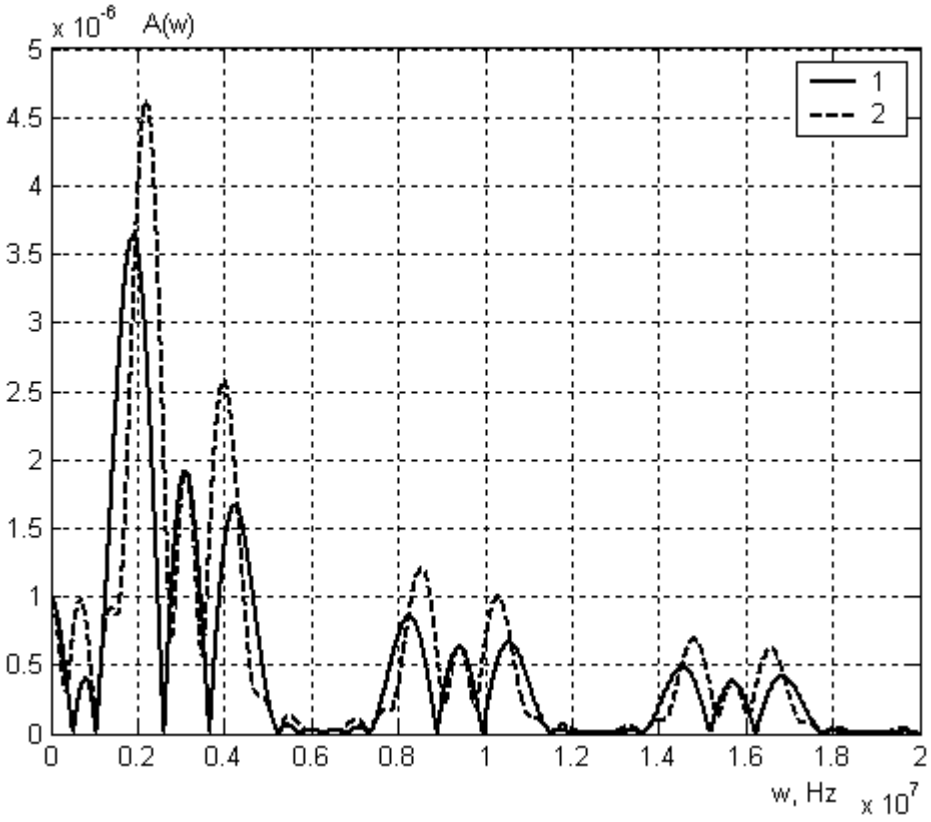


Рис. 2.9 Спектри сигналів двійкової послідовності 11010110 (1) та коду Хеммінга для цієї послідовності 110110111000 (2) за умови використання для подання цих сигналів коду AMI

Із рис. 2.9 зрозуміло, що спектр сигналу коду Хеммінга має більшу енергію, що обумовлено більшою кількість бітів, проте різниця між

енергетичними витратами не перевищує 20%, причиною чого є невисока надлишковість коду. Зважаючи на те, що код Хеммінга виправляє одиночні помилки, а імовірність подвійної помилки у дванадцяти бітах кодової послідовності є невисокою, у реальних системах витрати енергії на передавання закодованої інформації є меншими. Щодо частотного діапазону натурального коду та коду Хеммінга, то, як видно з рис. 2.9, за частотою ці два сигнали майже не відрізняються, причиною чого також є невисока надлишковість коду.

У науково-технічній літературі код Хеммінга заданої розрядності часто позначається як (n, k), де k – кількість інформаційних розрядів, n – загальна кількість розрядів [33, 56, 57, 62 – 65].

2.4.2 Лінійні коди із підвищеною коректувальною здатністю

У попередньому підрозділі був розглянутий та проілюстрований на конкретних прикладах спосіб побудови класичного варіанту кодів Хеммінга із мінімальною кодовою відстанню $d_{\min} = 3$, які дозволяють виправляти одиночні помилки. Зокрема, були розглянуті коди (12, 8) та (17, 12). Зрозуміло, що для підвищення коректувальної здатності лінійних кодів необхідно збільшувати кодову відстань $d_{\min}$ між дозволеними кодовими комбінаціями та змінювати відповідним чином алгоритми обчислення контрольних сум.

Розглянемо способи формування деяких із таких кодів.

Існує модифікований код Хеммінга, який загалом будується таким же чином, як і код із виправленням одиночних помилок, описаний у підрозділі 2.4.1. Єдина різниця полягає у тому, що до кодової комбінації додається ще один контрольний розряд, значення якого обчислюється як результат загальної перевірки на парність всіх розрядів коду [5, 33, 56, 57, 62 – 65]. Дешифрування такого коду проводиться у зворотному порядку. Спочатку необхідно зробити загальну перевірку на парність, а потім перевірити контрольні суми. Для модифікованого коду Хеммінга $d_{\min} = 4$.

Приклад 2.8. Записати контрольні суми для модифікованого коду Хеммінга (8, 4), який дозволяє виправляти одиночні та виявляти подвійні помилки.

Будемо розв'язувати поставлену задачу в три етапи. Спочатку за алгоритмом, описаним у підрозділі 2.4.1, побудуємо код Хеммінга із кодовою відстанню $d_{\min} = 3$. На другому етапі добавимо додатковий контрольний розряд та визначимо спосіб обчислення відповідної контрольної суми. А на третьому етапі визначимо спосіб виявлення та виправлення помилок відповідно до значень контрольних сум.

Згідно із алгоритмом, який був описаний у підрозділі 2.4.1, запишемо контрольні суми для коду (7, 4). Такий код дійсно є кодом Хеммінга, оскільки, згідно із властивістю 2.1, в трьох контрольних розрядах цього коду можна записати номер будь-якої позиції кодової комбінації, від 1 до 7. Зрозуміло також, що, згідно із властивістю 2.2, три контрольних розряди розташовуються на першій, другий та четвертій позиціях числа.

Використовуючи універсальний спосіб побудови контрольних сум, описаний у підрозділі 2.4.1, можна записати для їх обчислення наступні співвідношення:

$$S_1 = n_1 \oplus n_3 \oplus n_5 \oplus n_7; \quad S_2 = n_2 \oplus n_3 \oplus n_6 \oplus n_7; \quad (2.63)$$

$$S_3 = n_4 \oplus n_5 \oplus n_6 \oplus n_7; \quad S_{e1} = [S_3, S_2, S_1],$$

де S_{e1} – перша загальна контрольна сума, яка будується за значеннями контрольних сум S_1, S_2, S_3 та використовується у коді (7, 4) для визначення номера розряду, в якому виникла помилка.

Згідно із описаною вище методикою побудови модифікованого коду Хеммінга, будемо визначати другу загальну контрольну суму S_{e2}, як результат сумування всіх розрядів коду за модулем 2, тобто:

$$S_{e2} = n_1 \oplus n_2 \oplus n_3 \oplus n_4 \oplus n_5 \oplus n_6 \oplus n_7 \oplus n_8. \quad (2.64)$$

Тоді за значеннями загальних контрольних сум S_{e1} та S_{e2} виявлення та виправлення помилок у сформованому коді (8, 4) здійснюється з використанням синдромів помилки, які описані у таблиці 2.9.

Таблиця 2.9 – Синдроми помилки для коду Хеммінга (8, 4)

Значення S_{e1}	Значення S_{e2}	Синдром помилки
$S_{e1} = 0$	$S_{e2} = 0$	Помилка відсутня
$S_{e1} = 0$	$S_{e2} = 1$	Помилка у восьмому розряді
$S_{e1} \neq 0$	$S_{e2} = 0$	Подвійна помилка, в результаті якої корекція кодової комбінації блокується та надсилається запит на її повторне передавання
$S_{e1} \neq 0$	$S_{e2} = 1$	Одиночна помилка, яка виправляється згідно із значенням контрольної суми S_{e1}

Розглянемо приклад формування коду Хеммінга за алгоритмом, описаним у прикладі 2.8, та проаналізуємо його коректувальну здатність.

Приклад 2.9. За алгоритмом, описаним у прикладі 2.8, сформувати код Хеммінга для числа 1011 та проаналізувати роботу коду за умови одиночної помилки у третьому розряді кодової комбінації та подвійної помилки у п’ятому та шостому розрядах.

Зрозуміло, що, згідно із властивістю 2.2, шаблон для кодової комбінації, яка створюється, буде мати наступний вигляд:

$$r_4, 1, 0, 1, r_3, 1, r_2, r_1. \qquad (2.65)$$

Для формування контрольних сум та обчислення контрольних розрядів $r_1 - r_3$ скористаємося співвідношеннями (2.63), (2.64).

$S_1 = n_1 \oplus n_3 \oplus n_5 \oplus n_7 = r_1 \oplus 1 \oplus 1 \oplus 1; \Rightarrow r_1 = 1.$

$S_2 = n_2 \oplus n_3 \oplus n_6 \oplus n_7 = r_2 \oplus 1 \oplus 0 \oplus 1; \Rightarrow r_2 = 0.$

$S_3 = n_4 \oplus n_5 \oplus n_6 \oplus n_7 = r_3 \oplus 1 \oplus 0 \oplus 1; \Rightarrow r_3 = 0.$

Тепер для обчислення другої загальної контрольної суми S_{e2} скористаємось співвідношенням (2.64).

$$S_{e2} = n_1 \oplus n_2 \oplus n_3 \oplus n_4 \oplus n_5 \oplus n_6 \oplus n_7 \oplus n_8 = r_1 \oplus r_2 \oplus 1 \oplus r_3 \oplus 1 \oplus 0 \oplus 1 \oplus r_4 =$$
$$= 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus r_4; \Rightarrow r_4 = 0.$$

Тобто, згідно із визначеним шаблоном (2.65), модифікованому коду Хеммінга для числа 1011 відповідає кодова комбінація 01010101.

У разі помилки у третьому розряді кодової комбінації отримуємо спотворену комбінацію 01010001. Тоді обчислення першої загальної контрольної суми S_{e1} дає наступний результат:

$S_1 = n_1 \oplus n_3 \oplus n_5 \oplus n_7 = 1 \oplus 0 \oplus 1 \oplus 1; \quad \Rightarrow S_1 = 1.$

$S_2 = n_2 \oplus n_3 \oplus n_6 \oplus n_7 = 0 \oplus 0 \oplus 0 \oplus 1; \quad \Rightarrow S_2 = 1.$

$S_3 = n_4 \oplus n_5 \oplus n_6 \oplus n_7 = 0 \oplus 1 \oplus 0 \oplus 1; \quad \Rightarrow S_3 = 0.$

Друга загальна контрольна сума S_{e2}, відповідно, має значення:

$S_{e2} = n_1 \oplus n_2 \oplus n_3 \oplus n_4 \oplus n_5 \oplus n_6 \oplus n_7 \oplus n_8 = 1 \oplus 0 \oplus 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 1.$

Із таблиці 2.9 робимо висновок, що у даному разі виникла одиночна помилка, яку необхідно виправити, а значення розряду кодової комбінації, в якому виникла помилка, вказують контрольні суми $S_1 – S_3$. Це число $011_2 = 3$, тобто, третій розряд.

У випадку подвійної помилки у п’ятому та шостому розрядах спотворена кодова комбінація має вигляд 01100101. Тоді елементи першої загальної контрольної суми S_{e1}, $S_1 – S_3$, мають такі значення:

$S_1 = n_1 \oplus n_3 \oplus n_5 \oplus n_7 = 1 \oplus 1 \oplus 0 \oplus 1; \quad \Rightarrow S_1 = 1.$

$S_2 = n_2 \oplus n_3 \oplus n_6 \oplus n_7 = 0 \oplus 1 \oplus 1 \oplus 1; \quad \Rightarrow S_2 = 1.$

$S_3 = n_4 \oplus n_5 \oplus n_6 \oplus n_7 = 0 \oplus 0 \oplus 1 \oplus 1; \quad \Rightarrow S_3 = 0.$

Для другої загальної контрольної суми S_{e2}, відповідно, маємо:

$S_{e2} = n_1 \oplus n_2 \oplus n_3 \oplus n_4 \oplus n_5 \oplus n_6 \oplus n_7 \oplus n_8 = 1 \oplus 0 \oplus 1 \oplus 0 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 0.$

Згідно із таблицею 2.9 сукупність значень $S_{e1} \neq 0$ та $S_{e2} = 0$ свідчить про наявність подвійної помилки у кодовій комбінації, тобто, подвійна помилка виявлена.

Значно складнішим є завдання побудови лінійного коду, який забезпечує виправлення подвійних помилок. Згідно із співвідношенням (2.36) для таких кодів мінімальна кодова відстань між дозволеними комбінаціями повинна бути $d_{\min} = 5$. Сформулюємо загальні правила формування таких кодів як відповідні властивості лінійних кодів [5, 33].

Властивість 2.3. Синдром одиночної помилку відповідає або вектору помилки, або одній із невикористаних кодових комбінацій меншої

розрядності.

Властивість 2.4. Для формування синдрому подвійної помилки синдроми одиночних помилок сумуються за модулем два.

Властивість 2.5. Кожний із синдромів одиночних та подвійних помилок має бути унікальним, тобто, будь-які визначені синдроми помилок не повинні співпадати.

Властивість 2.6. До контрольних сум включаються ті розряди кодової комбінації, для яких відповідний біт синдрому помилки має значення 1.

Під час формування контрольних сум для завадостійких кодів із виправленням подвійних помилок необхідно чітко слідкувати за виконанням властивостей 2.3 – 2.6. Крім цього, вважається, що помилки у різних розрядах кодової комбінації є незалежними подіями. Тобто, більш складні випадки, коли одна із помилок може стати причиною іншої, під час формування лінійних завадостійких кодів не розглядаються. Виявлення та виправлення таких пакетних помилок є іншою задачею теорії кодування, для розв'язування якої використовують не лінійні, а згорткові коди. Цей тип кодів буде розглядатися у третьому розділі цієї частини посібника.

Розглянемо спосіб формування лінійних завадостійких кодів із виправленням подвійних помилок більш детально та сформуємо можливі синдроми одиночних помилок для номерів розрядів від 1 до 15. Зрозуміло, що під час розв'язування цієї задачі теоретичним підґрунтям для визначення синдромів помилок мають бути властивості 2.3 – 2.6.

Синдроми помилок для першого та другого розрядів кодової комбінації є простими та відповідають синдромам для коду Хеммінга із виправленням одиночної помилки. Тобто, для першого розряду синдром помилки складає 00000001, а для другого – 00000010. А далі починаються розбіжності. Так, для третього розряду синдром помилки 00000011 не може бути використаним. Такий принцип формування синдрому суперечить властивості 2.5, оскільки кодова комбінація 00000011, згідно із властивістю 2.4, є синдромом подвійної помилки у першому та другому розрядах. Дійсно,

$00000001 \oplus 00000010 = 00000011$.

Тоді, оскільки всі можливі двохрозрядні комбінації синдромів помилок вже використані, для формування наступних синдромів необхідно перейти до наступного, третього розряду. Наприклад, синдромом помилки у третьому розряді може бути кодова комбінація 00000100. За таких умов, згідно із властивістю 2.4, синдром 00000101 відповідає подвійній помилці у першому та третьому розрядах, а синдром 00000110 – подвійній помилці у другому та третьому розрядах. Із можливих трьохрозрядних комбінацій не використана комбінація 00000111, і на перший погляд здається, що саме вона може бути синдромом помилки у четвертому розряді. Проте тут необхідно впевнитись у тому, що у разі використання синдромів 00000111, 00000100, 00000010 та 00000001 синдроми всіх подвійних помилок будуть унікальними. Відповідні перевірки дають наступний результат:

$00000111 \oplus 00000001 = 00000110$;

$00000111 \oplus 00000010 = 00000101$;

$00000111 \oplus 00000100 = 00000011$.

Зрозуміло, що всі три отриманих результати вже використані раніше як синдроми подвійних помилок. Синдром 00000110 відповідає подвійній помилці у другому та третьому розрядах, синдром 00000101 – помилці у першому та третьому розрядах, а синдром 00000011 – помилці у першому та другому розрядах. Тому, згідно із властивістю 2.5, синдром 00000111 у даному випадку не є придатним для використання і його слід відкинути. А оскільки всі інші можливі трьохрозрядні синдроми помилок вже використані, необхідно для подальшого формування синдромів перейти до четвертого розряду. Тоді синдромом помилки у четвертому розряді може бути кодова комбінація 00001000. Для цього випадку перевірки синдромів подвійних помилок дають наступний результати.

Для четвертого та першого розрядів: $00001000 \oplus 00000001 = 00001001$.

Для четвертого та другого розрядів: $00001000 \oplus 00000010 = 00001010$.

Для четвертого та третього розрядів: $00001000 \oplus 00000100 = 00001100$.

Зрозуміло, що отримані кодові комбінації 00001001, 00001010 та 00001100 раніше не були використані, тому їх можна розглядати як синдроми відповідних подвійних помилок.

Із чотирьохрозрядних комбінацій не використаною залишилась лише комбінація 00001111. Розглянемо, чи може вона бути використана як синдром помилки у п’ятому розряді. Сумування визначених синдромів помилок за модулем 2 дає наступний результат.

Для п’ятого та першого розрядів: $00001111 \oplus 00000001 = 00001110$.

Для п’ятого та другого розрядів: $00001111 \oplus 00000010 = 00001101$.

Для п’ятого та третього розрядів: $00001111 \oplus 00000100 = 00001011$.

Для п’ятого та четвертого розрядів: $00001111 \oplus 00001000 = 00000111$.

Всі отримані синдроми подвійних помилок є унікальними, тому можна вважати, що розглянута кодова комбінація 00001111 є коректним синдромом помилки у п’ятому розряді. Результати подальшого аналізу, згідно із властивостями 2.3 – 2.5, дозволяють отримати наступну таблицю для синдромів одиночних помилок [5, 64, 65].

Таблиця 2.10 – Синдроми помилки для лінійного коду із виправленням подвійних помилок

Номер розряду	Синдром помилки	Номер розряду	Синдром помилки	Номер розряду	Синдром помилки
1.	00000001	6.	00010000	11.	01101010
2.	00000010	7.	00100000	12.	10000000
3.	00000100	8.	00110011	13.	10010110
4.	00001000	9.	01000000	14.	10110101
5.	00001111	10.	01010101	15.	11011011

Розглянемо приклад формування лінійного коду із виправленням подвійних помилок з використанням синдромів помилки, визначених у таблиці 2.9.

Приклад 2.10. Побудувати лінійний код (8, 2), який дозволяє

виправляти подвійні помилки.

Зрозуміло, що код, який формується, повинен мати 2 інформаційних розряди та 6 контрольних. Контрольні суми будемо записувати згідно із синдромами помилки для восьмирозрядного числа, які систематизовані у таблиці 2.10.

Спосіб формування контрольних сум визначається властивістю 2.6. Наприклад, перша контрольна сума включає перший, п'ятий та восьмий розряди, друга – другий, п'ятий та восьмий, третя – третій та п'ятий. Загалом для адресації восьми розрядів завадостійкого коду використовується 6 розрядів синдрому помилки. Тому необхідно сформувати 6 контрольних сум, які записуються наступним чином.

$$S_1 = n_1 \oplus n_5 \oplus n_8;\ S_2 = n_2 \oplus n_5 \oplus n_8;\ S_3 = n_3 \oplus n_5;$$
$$S_4 = n_4 \oplus n_5;\ S_5 = n_6 \oplus n_8;\ S_6 = n_7 \oplus n_8. \qquad (2.66)$$

Із визначених контрольних сум видно, що значення n_5 та n_8 входять до декількох контрольних сум, у той час як всі інші розряди – лише до однієї контрольної суми. Такий спосіб формування контрольних сум дійсно відповідає синдромам помилки, заданим у таблиці 2.10. Тому для спрощення розв'язування системи рівнянь (2.66) два інформаційних розряди розташовуються на п'ятій та восьмій позиціях кодової комбінації, а решта 6 позицій заповнюються контрольними розрядами. Тобто, структура лінійного коду (8, 2) із виправленням подвійних помилок має вигляд, наведений у таблиці 2.11.

Таблиця 2.11 – Структура лінійного коду (8, 2), який дозволяє виправляти подвійні помилки

Номер розряду	n_8	n_7	n_6	n_5	n_4	n_3	n_2	n_1
Структура коду	k_2	r_6	r_5	k_1	r_4	r_3	r_2	r_1

Приклад 2.11. Побудувати лінійний код (8, 2) описаний у прикладі 2.9, для кодової послідовності 10 та перевірити коректність роботи такого

коду, вважаючи, що в результаті дії потужної завади помилки виникли у п’ятому та шостому розрядах кодової комбінації.

Згідно із структурою коду, наведеною у таблиці 2.11, а також із співвідношеннями (2.66), обчислимо контрольні розряди та сформуємо відповідну кодову комбінацію. Оскільки значення інформаційних розрядів $k_1 = n_5 = 0$ та $k_2 = n_8 = 1$ є відомими, із співвідношень (2.66) маємо:

$$S_1 = n_1 \oplus n_5 \oplus n_8 = n_1 \oplus 0 \oplus 1 = r_1 \oplus 0 \oplus 1 \Rightarrow r_1 = 1;$$

$$S_2 = n_2 \oplus n_5 \oplus n_8 = n_2 \oplus 0 \oplus 1 = r_2 \oplus 0 \oplus 1 \Rightarrow r_2 = 1;$$

$$S_3 = n_3 \oplus n_5 = n_3 \oplus 0 = r_3 \oplus 0 \Rightarrow r_3 = 0;$$

$$S_4 = n_4 \oplus n_5 = n_4 \oplus 0 = r_4 \oplus 0 \Rightarrow r_4 = 0;$$

$$S_5 = n_6 \oplus n_8 = n_6 \oplus 1 = r_5 \oplus 1 \Rightarrow r_5 = 1;$$

$$S_6 = n_7 \oplus n_8 = n_7 \oplus 1 = r_6 \oplus 1 \Rightarrow r_6 = 1.$$

З урахуванням проведених розрахунків таблицю 2.11 можна переписати наступним чином:

Таблиця 2.12 – Розташування інформаційних та контрольних розрядів у лінійному коді (8, 2) та значення цих розрядів для двійкового числа 10

Номер розряду	n_8	n_7	n_6	n_5	n_4	n_3	n_2	n_1
Структура коду	k_2	r_6	r_5	k_1	r_4	r_3	r_2	r_1
Значення біту	1	1	1	0	0	0	1	1

Тобто, згідно із проведеними розрахунками, для визначеної

послідовності бітів 10 код (8, 2), який виправляє подвійні помилки, має вигляд 11100011.

Тепер припустимо, що, згідно із умовою задачі, у п’ятому та шостому розрядах кодової комбінації виникли помилки і прийнята спотворена кодова комбінація 11010011. Тоді підрахунки контрольних сум (2.66) дають наступні результати:

$$S_1 = n_1 \oplus n_5 \oplus n_8 = r_1 \oplus 1 \oplus 1 = 1 \oplus 1 \oplus 1 \Rightarrow S_1 = 1;$$

$$S_2 = n_2 \oplus n_5 \oplus n_8 = r_2 \oplus 1 \oplus 1 = 1 \oplus 1 \oplus 1 \Rightarrow S_2 = 1;$$

$$S_3 = n_3 \oplus n_5 = r_3 \oplus 1 = 0 \oplus 1 \Rightarrow S_3 = 1;$$

$$S_4 = n_4 \oplus n_5 = r_4 \oplus 1 = 0 \oplus 1 \Rightarrow S_4 = 1;$$

$$S_5 = n_6 \oplus n_8 = r_5 \oplus 1 = 0 \oplus 1 \Rightarrow S_5 = 1;$$

$$S_6 = n_7 \oplus n_8 = r_6 \oplus 1 = 1 \oplus 1 \Rightarrow S_6 = 0.$$

Тобто, помилка існує та її синдрому відповідає значення 011111. Звернемося до таблиці 2.10, із якої видно, що помилці у п’ятій позиції кодової комбінації відповідає синдром 1111, а помилці у шостій позиції – синдром 10000. Зважаючи на те, що $1111 \oplus 10000 = 11111$, можна зробити висновок, що помилка виникла у п’ятому та шостому розрядах числа.

Із розглянутих прикладів можна зробити висновок, що лінійні коди із підвищеною коректувальною здатністю є більш складними з точки зору алгоритмів їхньої побудови, ніж класичний код Хеммінга із виправленням одиночних помилок, який розглядався у підрозділі 2.4.1. Крім цього, такі коди мають більший коефіцієнт надлишковості, що необхідно для забезпечення досить високої кодової відстані $d_{\min} = 5$. Наприклад, для розглянутого коду (8, 2), згідно із співвідношеннями (2.38), (2.39), маємо:

$$R_n = \frac{n-k}{n} = \frac{8-2}{8} = 0{,}75\,, \qquad R_k = \frac{n-k}{k} = \frac{8-2}{2} = 3\,.$$

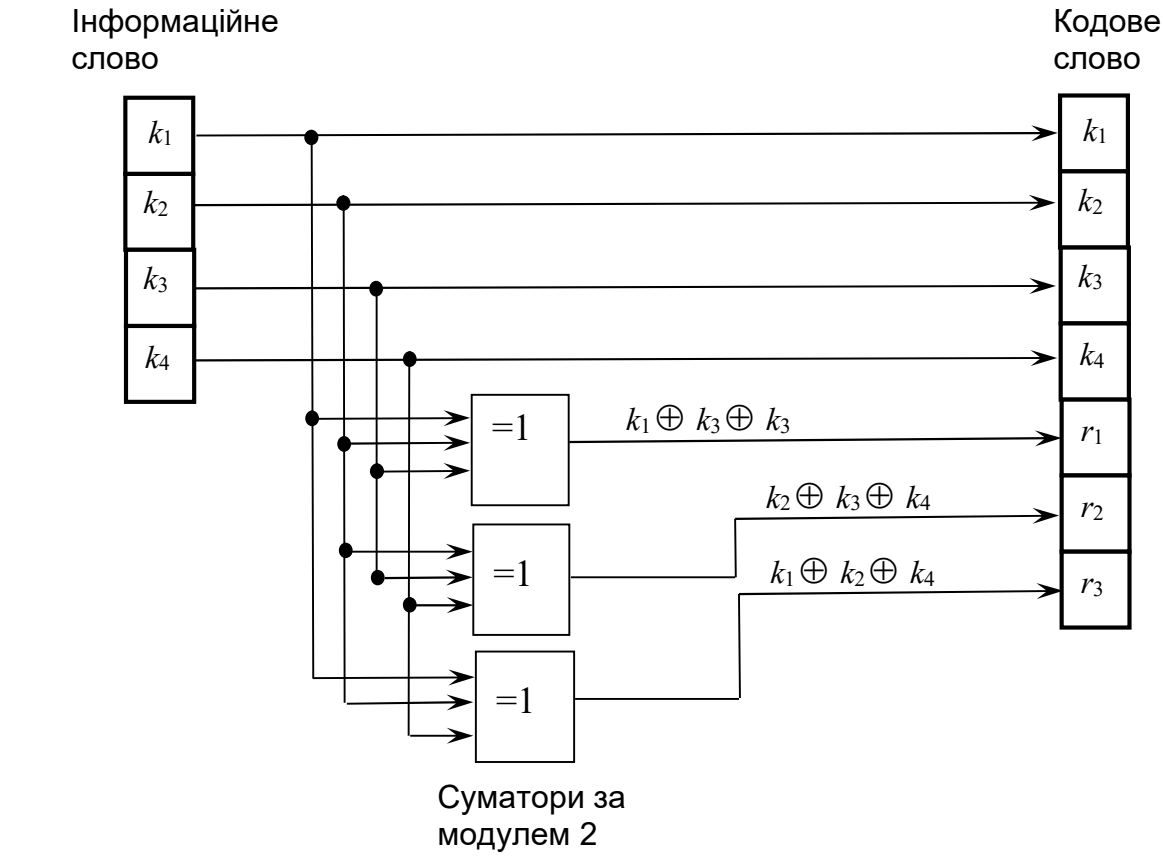
Як складність алгоритмів формування, так і велика кількість контрольних символів, у значній мірі ускладнюють технічні засоби кодувальної та декодувальної електронної апаратури. Враховуючи коштовність відповідного електронного обладнання, а також те, що в сучасній електронній апаратурі та у надійних системах зв'язку імовірність виникнення помилок великої кратності є вкрай низькою, лінійні коди із підвищеною коректувальною здатністю сьогодні майже не використовуються [5, 33, 56, 57, 64, 65]. Особливості технічної реалізації сучасної цифрової кодувальної та декодувальної електронної апаратури для кодів Хеммінга розглядатимуться у наступному підрозділі.

2.4.3 Схеми цифрових електронних пристроїв для формування лінійних кодів та їхнього декодування

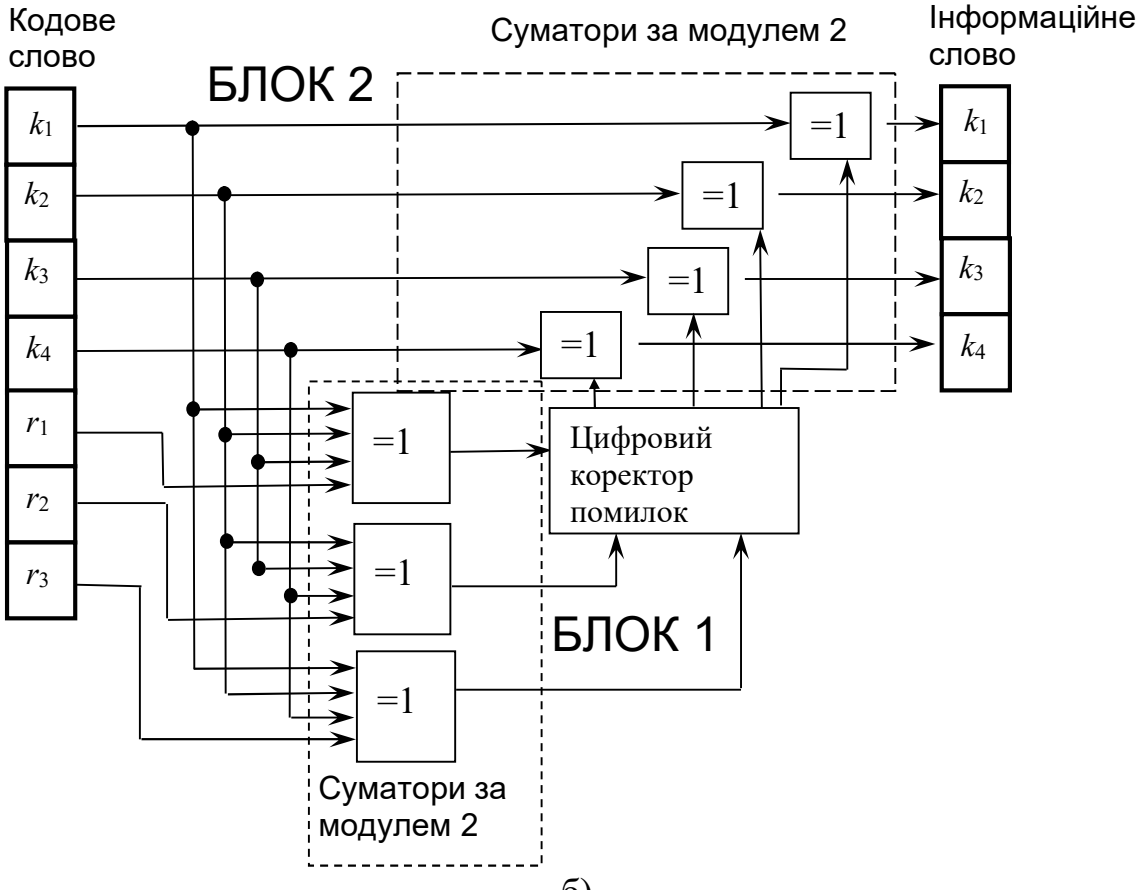
Узагальнена схема кодера коду Хеммінга (7, 4) наведена на рис. 2.10, а, а відповідна схема декодера – на рис. 2.10, б. На вхід кодувального пристрою подається чотирьохрозрядне інформаційне слово, а контрольні символи формуються з використанням логічних схем сумування за модулем 2 на основі наступних контрольних сум [5, 33, 56, 57, 64, 65]:

$$\begin{aligned} S_1 &= n_1 \oplus n_3 \oplus n_5 \oplus n_7 = r_1 \oplus k_1 \oplus k_2 \oplus k_3. \\ S_2 &= n_4 \oplus n_5 \oplus n_6 \oplus n_7 = r_3 \oplus k_2 \oplus k_3 \oplus k_4. \\ S_3 &= n_2 \oplus n_3 \oplus n_6 \oplus n_7 = r_2 \oplus k_1 \oplus k_2 \oplus k_4. \end{aligned} \qquad (2.67)$$

Таким чином, згідно із структурною схемою, наведеною на рис. 2.10, а, із інформаційного слова формується кодове.



а)



б)

Рис. 2.10 Узагальнені схеми кодера (а) та декодера (б) коду Хеммінга

Головним функціональним елементом декодувального пристрою, структурна схема якого зображена на рис. 2.10, б, є цифровий коректор помилок [5, 33, 64, 65]. На вхід декодера подається кодове слово і, з використанням логічних схем сумування за модулем 2 першого блоку, перевіряються контрольні суми (2.67). Якщо результат такої перевірки не дорівнює 0, цифровий коректор подає сигнал 1 на відповідну схему сумування за модулем 2 другого блоку і таким чином виправляється помилковий інформаційний розряд. Процес декодування кодових послідовностей у значній мірі спрощується тим, що контрольні розряди на вихід декодера не виводяться і тому не має необхідності їх виправляти.

Узагальнені схеми, наведені на рис. 2.10, є простими для аналізу, проте недолік таких схем з технічної точки зору полягає у тому, що вони не пояснюють принципи формування цифрового сигналу. Крім того, схема кодувального пристрою, наведена на рис. 2.10, а, є асинхронною, що не надає можливості узгоджувати процес кодування із часом формування цифрового сигналу. Хоча швидкодія асинхронних електронних пристроїв зазвичай є більш високою [2, 3, 8, 15 – 24, 33], проте неузгодженість такої кодувальної апаратури із схемами формування цифрового сигналу може стати додатковою причиною спотворення інформаційних потоків [1, 5]. Для позначення логічного елементу «ВИКЛЮЧНОГО АБО» на рис. 2.10, і надалі, на всіх електронних схема, використовується стандартний символ =1. Це позначення було введено у підрозділі 2.2.1 другої частини посібника під час описання логічних функцій алгебри Буля [48].

Більш деталізована схема синхронного цифрового електронного пристрою для формування коду Хеммінга (7, 3), яка побудована на базі тригерів та суматорів за модулем 2, наведена на рис. 2.11, а відповідна схема декодувального пристрою – на рис. 2.12 [5, 64, 65].

Розглянемо принцип роботи кодувальної схеми, наведеної на рис. 2.11. Загалом він базується на формуванні контрольних сум згідно із співвідношеннями (2.67). Під час надходження імпульсу синхронізації чотирьохрозрядне двійкове число, яке кодується, записується в інформаційні розряди регістра пам’яті розрядності n. В результаті виконання цієї операції встановлюються стани RS – тригерів.

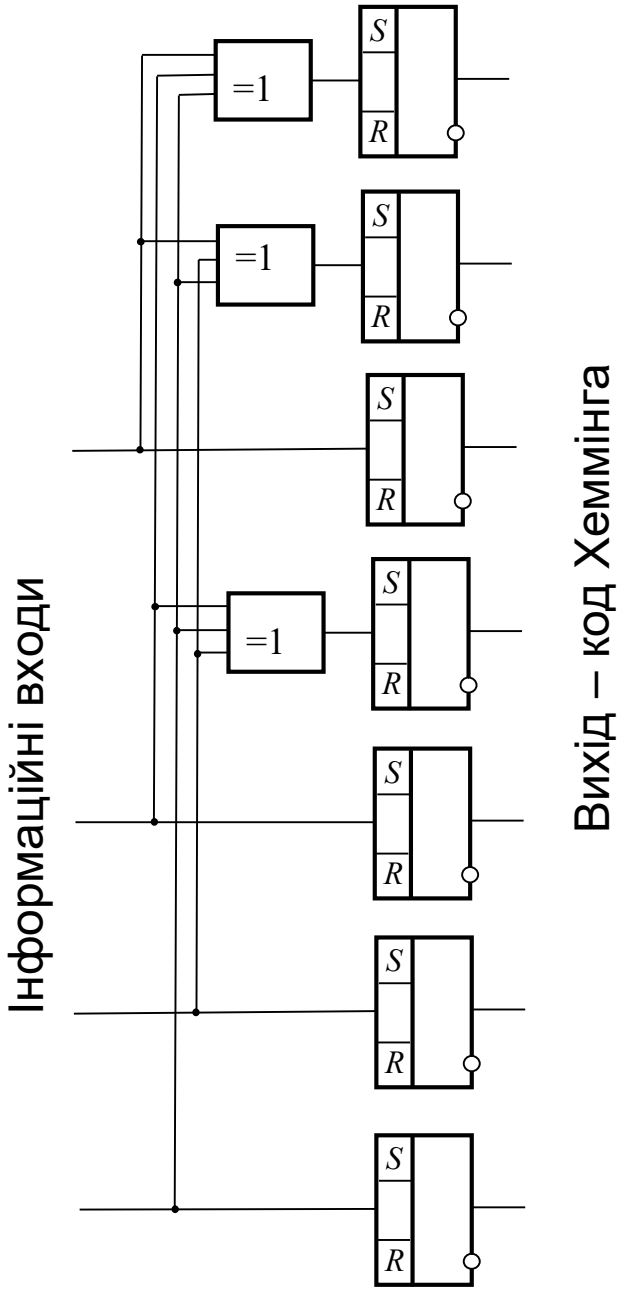


Рис. 2.11 Схема формування коду Хеммінга (7, 3), реалізована на тригерах та суматорах за модулем 2

Із відповідною затримкою формуються сигнали від суматорів за модулем 2, які встановлюють значення контрольних розрядів r_1, та r_2 та r_3 згідно із співвідношеннями (2.67). Результат виконання цієї операції за сигналом, який надходить з блока керування, послідовно або паралельно надходить до буфера даних, який пов’язаний із каналом зв’язку.

Схема декодувального пристрою для коду Хеммінга (7.3), який виправляє одиночні помилки, наведена на рис. 2.12, а на рис. 2.13 представлена схема декодувального пристрою для лінійного коду (8.2), призначеного для виправлення подвійних помилок. Принцип роботи такого коду був розглянутий у прикладі 2.10 [5, 64, 65].

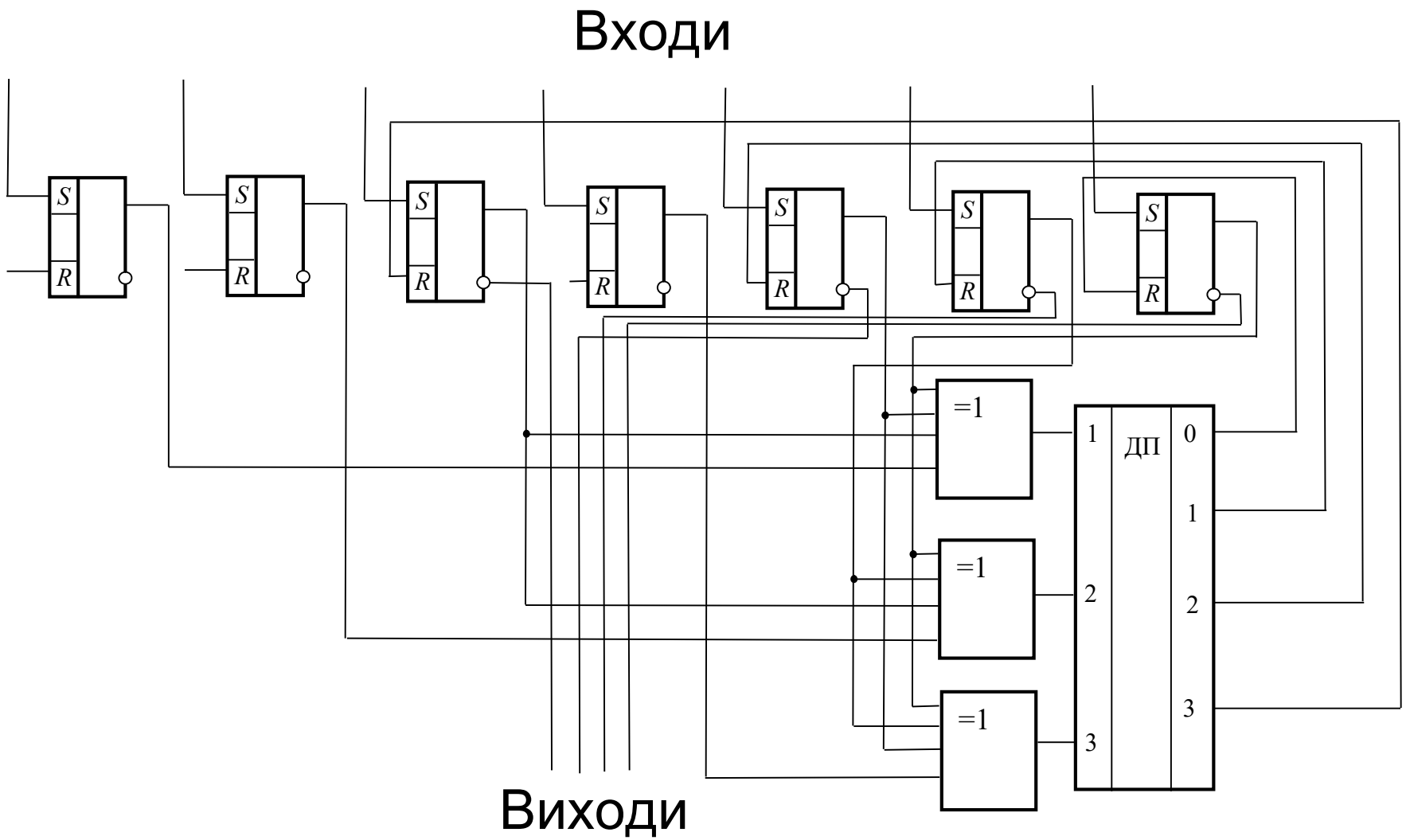


Рис. 2.12 Схема декодера коду Хеммінга (7, 3), реалізована на тригерах та суматорах за модулем 2

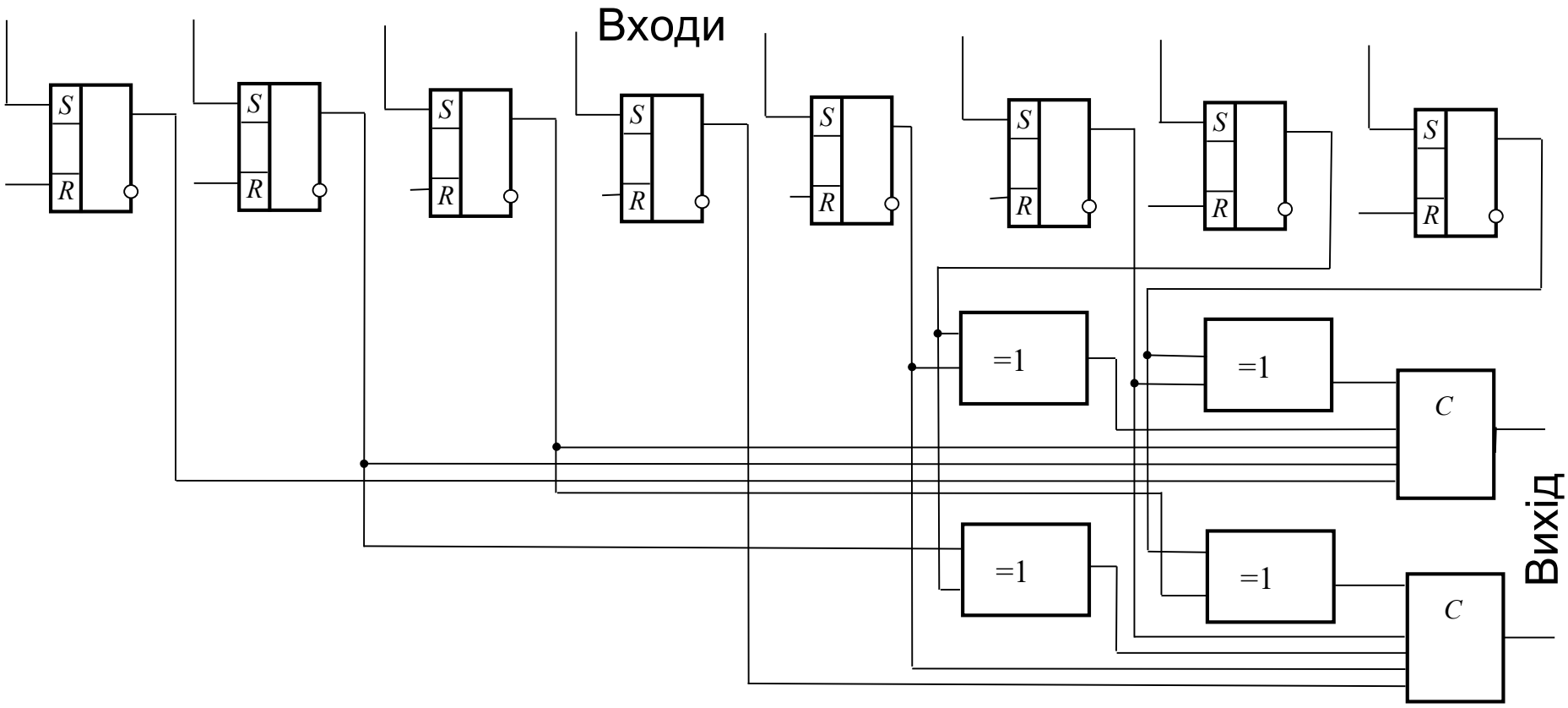


Рис. 2.13 Схема декодера коду Хеммінга (8, 2), розглянутого у прикладі 2.10, реалізована на тригерах та суматорах за модулем 2

Схема декодера, яка наведена на рис. 2.12, працює наступним чином [5]. Кодова комбінація, у якій може бути помилка, надходить на вхід регістру розрядності n, побудованого на основі RS – тригерів. Після завершення перехідних процесів у тригерах з блоку керування на суматори надходить імпульс опитування. Вхідні імпульси суматорів встановлюють RS – тригери в положення 0 або 1. Якщо контрольні суми (2.67) виконуються, всі тригери встановлюються в положення 0, і це свідчить про відсутність помилки. За умови наявності помилки її вектор записується до регістру, який побудований на RS – тригерах, а дешифратор помилки ДП встановлює відповідність між синдромом помилки та відповідними розрядами. Під час опитування вихідних ключових пристроїв дешифратора сигнали корекції надходять лише на ті розряди, для яких вектор синдрому помилки має значення 1. Коли такі сигнали корекції діють на лічильні входи RS – тригерів, ці тригери змінюють свій стан на протилежний, і, таким чином, проводиться виправлення помилки. Зрозуміло, що у разі зчитування лише інформаційних бітів на тригери, яким відповідають контрольні розряди числа, імпульсів корекції можна не надсилати, і це у значній мірі спрощує схему декодера.

Розглянемо тепер принцип роботи декодера для лінійного коду (8, 2), схема якого наведена на рис. 2.13. Така схема називається схемою мажоритарного декодування (англійський термін – majoritar decoding) [5, 64, 65]. Наведемо відповідне визначення.

Визначення 2.6. Мажоритарним декодуванням називається такий спосіб декодування, для якого кожний інформаційний символ кодової комбінації визначається кількома лінійними виразами через інші символи, а правильним вважається те значення інформаційного символу, яке зустрічається найчастіше [52, 53, 62, 63 – 65].

Сутність мажоритарного декодування можна пояснити наступним чином. Якщо прийнята кодова комбінація не містить помилок, всі контрольні

суми дають однакові результати для відповідного інформаційного розряду і прийнята кодова комбінація залишається без змін. У разі помилки у будь-якому із розрядів значення інформаційного біту, обчислені через контрольні суми, різняться, і тоді рішення про правильну кодову комбінацію приймається за мажоритарним принципом. Якщо в обчислених значеннях інформаційного біту більша кількість нулів, ніж одиниць, правильним значенням вважається 0, а в противному випадку – одиниця. У випадку, якщо перевірки контрольних сум дають однакову кількість нулів та одиниць, правильним вважається прийняте значення відповідного біту у кодовій комбінації.

Розглянемо приклад використання принципу мажоритарного декодування для пошуку подвійної помилки у лінійному коді (8, 2).

Приклад 2.12. З використанням лінійного коду (8, 2) закодована кодова комбінація 11, і в результаті отриманий код 01101101. Перевірити отриману кодову комбінацію за контрольними сумами, і якщо отриманий код є невірним – зробити корекцію помилки з використанням методу мажоритарного декодування.

Скористаємося таблицею 2.10 та співвідношеннями (2.66). Вважаючи, що $k_1 = n_5 = 1$ та $k_2 = n_8 = 1$, через отримані вирази для контрольних сум (2.66) розрахуємо значення елементів коду n_5 та n_8. Зрозуміло, що значення інформаційного розряду n_5 входить до сум S_1, S_2, S_3, S_4, а розряду n_8 – до сум S_1, S_2, S_5, S_6. Особливість розрахунків за мажоритарним принципом полягає у тому, що на етапі аналізу контрольних сум наявне значення відповідного інформаційного розряду у кодовій комбінації не враховується, він обчислюється через контрольні суми. Наприклад, якщо $S_1 = n_1 \oplus n_5 \oplus n_8$ та аналізується значення n_5, будемо вважати, що $n_5 = n_1 \oplus n_8$. Тоді, із співвідношень (2.66), отримуємо наступні значення для розрядів n_5 та n_8:

$$S_1 = n_1 \oplus n_5 \oplus n_8 \Rightarrow n_5 = n_1 \oplus n_8 = 1;$$

$$S_2 = n_2 \oplus n_5 \oplus n_8 \Rightarrow n_5 = n_2 \oplus n_8 = 0;$$

$$S_3 = n_3 \oplus n_5 \Rightarrow n_5 = n_3 = 1;$$

$$S_4 = n_4 \oplus n_5 \Rightarrow n_5 = n_4 = 1;$$

$$S_1 = n_1 \oplus n_5 \oplus n_8 \Rightarrow n_8 = n_1 \oplus n_5 = 1;$$

$$S_2 = n_2 \oplus n_5 \oplus n_8 \Rightarrow n_8 = n_2 \oplus n_5 = 0;$$

$$S_5 = n_6 \oplus n_8 = \Rightarrow n_8 = n_6 = 1;$$

$$S_6 = n_7 \oplus n_8 \Rightarrow n_8 = n_7 = 1.$$

Тобто, результати перевірки першого та другого інформаційних бітів за мажоритарним принципом дають значення 1011. Тому вважаємо, що $n_5 = 1$ та $n_8 = 1$.

Розглянутий мажоритарний принцип декодування дуже спрощує аналіз складних кодових послідовностей та виправлення помилок великої кратності. Виявляється, що синдром помилки взагалі не потрібно знати, достатньо для отриманої кодової послідовності порахувати контрольні суми та порівняти кількість отриманих нулів та одиниць. Дійсно, якщо проаналізувати співвідношення (2.66), лише контрольні суми S_1 та S_2 містять лінійні залежності між інформаційними елементами коду n_5 та n_8, а решта контрольних сум дублюють ці інформаційні розряди. Наприклад, із контрольних сум S_3 та S_4 зрозуміло, що $n_3 = n_4 = n_5$, а із контрольних сум S_5 та S_6 видно, що $n_6 = n_7 = n_8$. Тобто, мажоритарний підхід дозволяє відійти від необхідності аналізу великої кількості синдромів помилок, який потребує проведення великої кількості обчислень та порівнянь та вкрай ускладнює електронну декодувальну апаратуру. Дійсно, навіть для простого коду (8, 2) кількість подвійних помилок відповідає кількості сполучень C_8^2 і складає [48]:

$$C_8^2 = \frac{8!}{2!\cdot(8-2)!} = \frac{7 \cdot 8}{2} = 28,$$

тоді сумарна кількість одиночних та подвійних помилок становить 28 + 8 = 36.

Крім цього, формування такої кількості синдромів помилок потребує введення до електронних схем відповідної кількості електричних з’єднань, що

також у значній мірі ускладнює електронну апаратуру, а у разі використання мажоритарного принципу задача декодування стає значно простішою. Достатньо поставити у схему два лічильника-компаратора *C*, які підраховують кількість одиниць та нулів у контрольних сумах і на основі виконання цієї операції видають значення відповідного інформаційного біту. Саме так і побудована схема декодувального пристрою, наведена на рис. 2.13.

Слід відзначити, що з цієї точки зору лінійні коди із високою коректувальною здатністю у деякій мірі схожі на коди із повторенням елементів, які були розглянуті у підрозділі 2.3. Дійсно, як вже було відмічено вище, у співвідношеннях (2.66) лише перша та друга контрольні суми характеризують лінійні залежності між інформаційними та контрольними символами коду, а решта контрольних сум, тобто суми S_3 – S_6, лише повторюють значення інформаційних розрядів. За таких умов використання мажоритарного принципу декодування в електронній апаратурі дійсно є виправданим, а іноді, навіть, і ефективним [5, 64, 65].

Проте слід відзначити, що головним недоліком лінійних кодів із високою коректувальною здатністю зазвичай є не необхідність обчислення великої кількості контрольних сум, а надто високе значення коефіцієнту надлишковості кодових комбінацій. Саме тому сьогодні такі коди рідко використовуються в електронній апаратурі та у сучасних стандартах зв'язку, що обумовлено непомірним зростанням потужності інформаційних потоків в електронних, комутувальних та в телекомунікаційних системах [5, 15 – 24, 28 – 31, 33 – 38, 64, 65].

2.4.4 Подання лінійних кодів через матричні перетворення

Перед вивченням цього підрозділу необхідно повторити четвертий та п'ятий розділи другої частини посібника

У загальному вигляді подання лінійних кодів через матричні

перетворення було розглянуто у підрозділі 5.4 другої частини посібника [48]. Нагадаємо, що для формування кодової комбінації лінійного коду використовується породжувальна матриця, а для формування інформаційних бітів із кодової комбінації – перевірочна матриця. Породжувальна матриця формується за визначеними контрольними сумами лінійного коду, а перевірочна – через породжувальну матрицю шляхом виділення із неї матриці доповнення **P** та її транспортування. Тобто, породжувальна матриця є результатом об'єднання одиничної матриці **E** та матриці доповнення **P** за стовпчиками, а перевірочна матриця – результатом об'єднання цих матриць за рядками. В результаті множення породжувальної матриці на вхідне інформаційне слово формується кодова комбінація, а результатом множення кодової комбінації на перевірочну матрицю є синдром помилки. Відповідно, у математичній формі [48]:

$$\overline{b} = P\left(\overline{a}\right) = \overline{a^{\mathrm{T}}} \cdot P,\ \mathbf{P} = \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1k} \\ p_{21} & p_{22} & \cdots & p_{2k} \\ \cdots & \cdots & \cdots & \cdots \\ p_{\rho 1} & p_{\rho 2} & \cdots & p_{\rho k} \end{pmatrix} = \begin{pmatrix} \overline{P_1} \\ \overline{P_2} \\ \cdots \\ \overline{P_\rho} \end{pmatrix}$$

$$\mathbf{H}^{\mathbf{T}} = \left(\mathbf{P}^{\mathbf{T}} \middle| \mathbf{E}_{\langle \boldsymbol{\rho} \rangle}\right) = \begin{pmatrix} p_{11} & p_{21} & \cdots & p_{\rho 1} \\ p_{12} & p_{22} & \cdots & p_{\rho 2} \\ \cdots & \cdots & \cdots & \cdots \\ p_{1k} & p_{2k} & \cdots & p_{\rho k} \\ 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \ldots & 1 \end{pmatrix}, \qquad (2.68)$$

де $\overline{b}$ – вектор контрольних розрядів числа, $\overline{a}$ – вектор інформаційних розрядів. Спосіб формування співвідношень (2.68), їх зв'язок із теорією груп, полями Галуа та із теорією матричних відношень, а також відповідні властивості породжувальної та перевірочної матриці лінійних кодів були розглянуті у п'ятому розділі другої частини посібника [48].

У третьому співвідношенні системи рівнянь (2.68) спосіб формування

перевірочної матриці $\mathbf{H^T}$ є таким, що для всіх елементів цієї матриці виконується умова [5, 64, 65]:

$$S_i = \sum_{i=1}^{k} c_i P_{ij} - c_j = 0. \qquad (2.69)$$

де **C** – вектор кодового слова, c_i, c_j – його елементи.

За умови правильної кодової комбінації виконується співвідношення (2.60), а якщо кодова комбінація **C** спотворена вектором помилки ξ, вектор синдрому помилки **S** залежить лише від вектора ξ і, згідно із співвідношенням (2.69), його компоненти S_i обчислюється як

$$S_i = \sum_{i=1}^{k} \xi_i P_{ij} - \xi_j. \qquad (2.70)$$

Більш досконало способи формування перевірочної матриці лінійного коду, її властивості та особливості її використання, були розглянуті у п’ятому розділі другої частини посібника [48].

Розглянемо приклади використання породжувальної та перевірочної матриць для формування різних типів лінійних кодів та для виправлення помилок у кодових комбінаціях.

Приклад 2.13. Сформувати породжувальну та перевірочну матрицю для коду Хеммінга (7, 4).

Для коду (7, 4) матриця доповнення повинна мати 7 стовпчиків та 4 рядки. Відповідно із способом формування коду, заданого контрольними сумами (2.67) та властивістю 2.2, рядки матриці доповнення відповідають контрольним розрядам r_1, r_2 та r_3. Наприклад, якщо контрольна сума r_1 визначається як $r_1 = k_1 \oplus k_2 \oplus k_4$, перший рядок матриці доповнення буде мати вигляд $\mathbf{S}_{<1>}$ = (1,1,0,1). Зрозуміло, що у даному випадку використаний лівосторонній порядок запису розрядів числа, який зазвичай використовується в матрицях та векторах. Для другого рядка, оскільки $r_2 = k_1 \oplus k_3 \oplus k_4$, $\mathbf{S}_{<2>}$ = (1,0,1,1). Для третього рядка $\mathbf{S}_{<3>}$ = (0,1,1,1). Через рядки $\mathbf{S}_{<1>}$, $\mathbf{S}_{<2>}$ та формується транспонована матриця доповнення $\mathbf{P^T}$ [5]. Відповідно, маємо:

$$\mathbf{P}^{\mathbf{T}} = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}, \qquad \mathbf{P} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}.$$

Слід відзначити, що головною умовою побудови породжувальної матриці є формування виразів для значень контрольних розрядів через відповідні інформаційні розряди, тобто, контрольних сум [52, 53, 62, 63 – 65].

Тоді систематична породжувальна матриця, яка формується як результат об'єднання за рядками одиничної матриці **E** та отриманої матриці **P**, має наступний вигляд:

$$\mathbf{M} = (\mathbf{E}, \mathbf{P}) = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

Тоді для того, щоб отримати код (7, 4) для будь-якого чотирьохрозрядного числа, необхідно вектор цього числа помножити за модулем два на отриману породжувальну матрицю **M**. Наприклад, для числа 0011 маємо:

$$(1,1,0,0) \cdot \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} = (((1\text{^}1) \oplus (1\text{^}0) \oplus (0\text{^}0) \oplus (0\text{^}0)),$$

$$((1\text{^}0) \oplus (1\text{^}1) \oplus (0\text{^}0) \oplus (0\text{^}0)), ((1\text{^}0) \oplus (1\text{^}0) \oplus (0\text{^}1) \oplus (0\text{^}0)),$$

$$((1\text{^}0) \oplus (1\text{^}0) \oplus (0\text{^}0) \oplus (0\text{^}1)), ((1\text{^}1) \oplus (1\text{^}1) \oplus (0\text{^}0) \oplus (0\text{^}1)),$$

$$((1\text{^}1) \oplus (1\text{^}0) \oplus (0\text{^}1) \oplus (0\text{^}1)), ((1\text{^}0) \oplus (1\text{^}1) \oplus (0\text{^}1) \oplus (0\text{^}1))) =$$

$$= ((1 \oplus 0 \oplus 0 \oplus 0), (0 \oplus 1 \oplus 0 \oplus 0), (0 \oplus 0 \oplus 1 \oplus 0), (0 \oplus 0 \oplus 0 \oplus 1),$$

$$(1 \oplus 1 \oplus 0 \oplus 0), (1 \oplus 0 \oplus 0 \oplus 0), (0 \oplus 1 \oplus 0 \oplus 0) = (1,1,0,0,0,1,1).$$

Слід відзначити, що у разі використання для формування лінійного коду систематизованої породжувальної матриці **M** порядок слідування розрядів буде наступним: k_1, k_2, k_3, k_4, r_1, r_2, r_3. Для зміни цього порядку згідно до структури коду, яка описується властивістю 2.2, необхідно поміняти місцями стовпчики матриці. На перше місце становляться п'ятий та шостий стовпчики, перший стовпчик стає третім, на четвертому місті буде розташований восьмий, а далі, підряд, ідуть другий, третій та четвертий

стовпчики. Така породжувальна матриця буде виглядати наступним чином:

$$\mathbf{M_m} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

Така модифікація породжувальної матриці лінійного коду пов'язана із лінійними перетвореннями, і, згідно із властивостями лінійних кодів, описаними у п'ятому розділі другої частини посібника, визначені матриці **M** та $\mathbf{M_m}$ є еквівалентними [48].

Правила проведення алгебраїчних операції множення двійкових матриць та векторів, які використані у даному прикладі, розглядалися у четвертому та п'ятому розділах другої частини посібника [48].

Враховуючи, що $\mathbf{P^T} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$, згідно із третім рівнянням системи (2.68) перевірочну матрицю **H** для коду Хеммінга (7, 4) можна записати наступним чином:

$$\mathbf{H} = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

Перевіримо, як можна використовувати перевірочну матрицю **H** для пошуку помилок у лінійних кодах. Для цього використаємо матричне співвідношення, еквівалентне співвідношенню (2.70):

$$\xi = \mathbf{C}\mathbf{H}^{\mathbf{T}}, \qquad (2.71)$$

де ξ – вектор помилки, **C** – вектор кодової комбінації.

Приклад 2.14. Використати результати розрахунків, отримані в прикладі 2.12, для перевірки коректності кодової комбінації 1100011. Провести також розрахунки за умови, що у третьому розряді визначеної кодової комбінації виникла помилка.

Скористаємося співвідношенням (2.71). Для вектора $\mathbf{C} = (1,1,0,0,0,1,1)$, отриманого у прикладі 2.12, отримуємо наступний результат:

$$\xi = (1,1,0,0,0,1,1)\cdot\begin{pmatrix}1&1&0\\1&0&1\\0&1&1\\1&1&1\\1&0&0\\0&1&0\\0&0&1\end{pmatrix} =$$

$$= (((1\text{^}1)\oplus(1\text{^}1)\oplus(0\text{^}0)\oplus(0\text{^}1)\oplus(0\text{^}1)\oplus(1\text{^}0)\oplus(1\text{^}0)),$$
$$((1\text{^}1)\oplus(1\text{^}0)\oplus(0\text{^}1)\oplus(0\text{^}1)\oplus(0\text{^}0)\oplus(1\text{^}1)\oplus(1\text{^}0)),$$
$$((1\text{^}0)\oplus(1\text{^}1)\oplus(0\text{^}1)\oplus(0\text{^}1)\oplus(0\text{^}0)\oplus(1\text{^}0)\oplus(1\text{^}1))) =$$
$$= ((1\oplus1\oplus0\oplus0\oplus0\oplus0\oplus0),(1\oplus0\oplus0\oplus0\oplus0\oplus1\oplus0),$$
$$(0\oplus1\oplus0\oplus0\oplus0\oplus0\oplus1) = (0,0,0).$$

Дійсно, для правильної кодової комбінації синдром помилки дорівнює 0. Тепер припустимо, що у третьому розряді коду виникла помилка і прийнята кодова комбінація 1100111. Для цього випадку маємо:

$$\xi = (1,1,1,0,0,1,1)\cdot\begin{pmatrix}1&1&0\\1&0&1\\0&1&1\\1&1&1\\1&0&0\\0&1&0\\0&0&1\end{pmatrix} =$$

$$= (((1\text{^}1)\oplus(1\text{^}1)\oplus(1\text{^}0)\oplus(0\text{^}1)\oplus(0\text{^}1)\oplus(1\text{^}0)\oplus(1\text{^}0)),$$
$$((1\text{^}1)\oplus(1\text{^}0)\oplus(1\text{^}1)\oplus(0\text{^}1)\oplus(0\text{^}0)\oplus(1\text{^}1)\oplus(1\text{^}0)),$$
$$((1\text{^}0)\oplus(1\text{^}1)\oplus(1\text{^}1)\oplus(0\text{^}1)\oplus(0\text{^}0)\oplus(1\text{^}0)\oplus(1\text{^}1))) =$$
$$= ((1\oplus1\oplus0\oplus0\oplus0\oplus0\oplus0),(1\oplus0\oplus1\oplus0\oplus0\oplus1\oplus0),$$
$$(0\oplus1\oplus1\oplus0\oplus0\oplus0\oplus1) = (0,1,1).$$

Оскільки отримане двійкове число 011 є синдромом помилки у третьому розряді кодової комбінації, можна зробити висновок, що помилка виявлена правильно і може бути виправленою.

Приклад 2.15. Побудувати породжувальну та перевірочну матриці для лінійного коду (8, 2), розглянутого у прикладі 2.9.

Контрольні суми для лінійного коду (8, 2), задані співвідношеннями

(2.66), спочатку слід переписати через значення інформаційних та контрольних розрядів. Відповідні співвідношення запишемо наступним чином:

$$S_1 = n_1 \oplus n_5 \oplus n_8 = r_1 \oplus k_1 \oplus k_2;\ S_2 = n_2 \oplus n_5 \oplus n_8 = r_2 \oplus k_1 \oplus k_2;\ S_3 = r_3 \oplus k_1;$$

$$S_4 = n_4 \oplus n_5 = r_4 \oplus k_1;\ S_5 = n_6 \oplus n_8 = r_5 \oplus k_2;\ \ S_6 = n_7 \oplus n_8 = r_6 \oplus k_2. \qquad (2.72)$$

Якщо в співвідношеннях виразити значення всіх контрольних розрядів через інформаційні, можна записати наступну систему лінійних рівнянь:

$$r_1 = k_1 \oplus k_2;\ r_2 = k_1 \oplus k_2;\ r_3 = k_1;\ r_4 = k_1;\ r_5 = k_2;\ r_6 = k_2.$$

Тоді транспонована матриця доповнення буде мати вигляд:

$$\mathbf{P^T} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix},$$

а матриця **P**, відповідно:

$$\mathbf{P} = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \end{pmatrix}.$$

Тоді, згідно із співвідношеннями (2.68), систематизована породжувальна матриця **M** буде мати вигляд:

$$\mathbf{M} = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \end{pmatrix},$$

а перевірочна матриця

$$\mathbf{H} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Слід відзначити, що, як і для коду Хеммінга (7, 4) у прикладі 2.13, отримана систематизована породжувальна матриця **M** не відповідає структурі коду. Якщо отримувати кодову комбінацію з використанням

першого рівняння системи (2.68), розряди в ній будуть розташовані наступним чином: k_1, k_2, r_1, r_2, r_3, r_4, r_5, r_6. Щоб отримати структуру коду, визначену у таблиці 2.10, необхідно в матриці **M** поміняти порядок стовпчиків. Перший стовпчик має бути розташований на п'ятій позиції, другий – на восьмій, а розташування інших стовпчиків не змінюється.

На перший погляд здається, що простішим є формування лінійних кодів описаним у підрозділі 2.4.1 звичайним способом, з використанням контрольних сум, ніж через матричні відношення. Проте слід мати на увазі, що матричний спосіб є більш універсальним, а алгоритми матричних відношень легко алгоритмізуються для застосування у комп'ютерних програмах. Математичний апарат, на якому базується матрична теорія завадостійких лінійних кодів, був досконало описаний у підрозділі 5.4 другої частини посібника. Також у додатку Р другої частини посібника був наведений код комп'ютерної програми, яка дозволяє розраховувати контрольні розряди кодової комбінації через породжувальну матрицю. Програма написана на мові програмування системи Matlab. Більш розширені теоретичні відомості про можливості використання теорії матриць та бінарних відношень для формування завадостійких лінійних кодів можна знайти в навчальній літературі [52, 53, 62, 63].

У наступному підрозділі буде розглянутий алгоритм формування кодів Хеммінга через контрольні суми та наведена відповідна комп'ютерна програма написана мовою програмування системи MatLab, у який реалізований цей алгоритм.

2.4.5 Комп'ютерна реалізація алгоритмів формування коду Хеммінга та декодування його кодових послідовностей

Розглянемо алгоритм формування кодів Хеммінга різної довжини, залежно від кількості бітів у вхідному інформаційному слові, та реалізацію цього алгоритму з використанням засобів програмування системи MatLab [13, 14]. Припустимо також, що разом з алгоритмом кодування у комп'ютерній програмі має бути реалізований відповідний алгоритм декодування кодових

послідовностей та виправлення в них одиночних помилок.

Зрозуміло, що алгоритм формування коду Хеммінга насамперед визначається способом побудови контрольних сум. Загалом способи побудови контрольних сум залежать від коректувальної здатності лінійного коду та від синдромів помилок, які мають бути сформовані. Наприклад, синдроми помилок для коду Хеммінга (12, 8) формуються згідно із таблицею 2.3, а синдроми помилок для коду (17, 12) – згідно із таблицею 2.7. Згідно з цим контрольні суми для коду (12, 8) визначаються співвідношеннями (2.54) – (2.57), а контрольним сумам для коду (17, 12) відповідають співвідношення (2.58) – (2.62). Контрольні суми для модифікованого коду Хеммінга (8, 4), який дозволяє виправляти одиночні та виявляти подвійні помилки, задаються співвідношеннями (2.63), (2.64). Із співвідношень (2.54) – (2.64) зрозуміло, що алгоритми побудови контрольних сум для кодів Хеммінга є стандартними. Вони визначаються властивостями 2.1, 2.2 для кодів із виправленням одиночних помилок або узагальненими властивостями 2.3 – 2.6 для кодів із виправленням подвійних помилок, проте кількість доданків у контрольних сумах залежить від розрядності коду. Тому для визначення алгоритмів побудови коду Хеммінга та декодування його кодових послідовностей насамперед слід конкретизувати задачу і задати діапазон розрядності для інформаційних та кодових слів. Будемо вважати, що мінімальна розрядність інформаційних слів, які можуть бути закодовані та декодовані, складає 4, а максимальна – 31. У цьому випадку мінімальна кількість контрольних розрядів і контрольних сум, які їм відповідають, для коду (7, 4) становить $r = 3$, а максимальна кількість контрольних розрядів, для коду (31, 26), складає $r = 5$. Будемо також розглядати можливість побудови модифікованого коду Хеммінга із виправленням одиночних та виявленням подвійних помилок, спосіб формування якого був описаний у прикладі 2.8.

На основі аналізу співвідношень (2.54) – (2.62) запишемо у загальному вигляді п’ять контрольних сум для всіх кодових комбінацій, кількість розрядів у яких є меншою за 32. Відповідні співвідношення є наступними:

$$\begin{cases} S_1 = n_1 \oplus n_3 \oplus n_5 \oplus n_7 \oplus n_9 \oplus n_{11} \oplus n_{13} \oplus n_{15} \oplus n_{17} \oplus n_{19} \oplus \\ \oplus n_{21} \oplus n_{23} \oplus n_{25} \oplus n_{27} \oplus n_{29} \oplus n_{31}; \\ S_2 = n_2 \oplus n_3 \oplus n_6 \oplus n_7 \oplus n_{10} \oplus n_{11} \oplus n_{14} \oplus n_{15} \oplus n_{18} \oplus n_{19} \oplus \\ \oplus n_{22} \oplus n_{23} \oplus n_{26} \oplus n_{27} \oplus n_{30} \oplus n_{31}; \\ S_3 = n_4 \oplus n_5 \oplus n_6 \oplus n_7 \oplus n_{12} \oplus n_{13} \oplus n_{14} \oplus n_{15} \oplus n_{20} \oplus n_{21} \oplus \\ \oplus n_{22} \oplus n_{23} \oplus n_{28} \oplus n_{29} \oplus n_{30} \oplus n_{31}; \\ S_4 = n_8 \oplus n_9 \oplus n_{10} \oplus n_{11} \oplus n_{12} \oplus n_{13} \oplus n_{14} \oplus n_{15} \oplus n_{24} \oplus n_{25} \oplus \\ \oplus n_{26} \oplus n_{27} \oplus n_{28} \oplus n_{29} \oplus n_{30} \oplus n_{31}; \\ S_5 = n_{16} \oplus n_{17} \oplus n_{18} \oplus n_{19} \oplus n_{20} \oplus n_{21} \oplus n_{22} \oplus n_{23} \oplus n_{24} \oplus \\ \oplus n_{25} \oplus n_{26} \oplus n_{27} \oplus n_{28} \oplus n_{29} \oplus n_{30} \oplus n_{31}. \end{cases} \quad (2.73)$$

Із властивості 2.2 зрозуміло, що контрольні розряди розташовуються на першій, другій, четвертій, восьмій та шістнадцятій позиціях кодової комбінації. Тобто, структура кодової комбінації (31, 26) із максимальною кількістю розрядів має вигляд, наведений у таблиці 2.13.

Таблиця 2.13 – Порозрядна структура коду Хеммінга (31, 26)

Номер розряду	n_{11}	n_{10}	n_9	n_8	n_7	n_6	n_5	n_4	n_3	n_2	n_1
Структура коду	k_7	k_6	k_5	r_4	k_4	k_3	k_2	r_3	k_1	r_2	r_1
Номер розряду	n_{22}	n_{21}	n_{20}	n_{19}	n_{18}	n_{17}	n_{16}	n_{15}	n_{14}	n_{13}	n_{12}
Структура коду	k_{17}	k_{16}	k_{15}	k_{14}	k_{13}	k_{12}	r_5	k_{11}	k_{10}	k_9	k_8
Номер розряду	–	–	n_{31}	n_{30}	n_{29}	n_{28}	n_{27}	n_{26}	n_{25}	n_{24}	n_{23}
Структура коду	–	–	k_{26}	k_{25}	k_{24}	k_{23}	k_{22}	k_{21}	k_{20}	k_{19}	k_{18}

Для кодових комбінацій із меншою кількістю розрядів останні розряди у таблиці 2.13 можна просто відкидати і не використовувати у сумах (2.73), вважаючи, що вони дорівнюють 0 [5]. Наприклад, для випадку n = 12 після

виконання цієї операції отримуємо співвідношення (2.54) – (2.57), для $n = 17$ – співвідношення (2.58) – (2.62), а для мінімального значення $n = 7$ наслідком системи лінійних рівнянь (2.64) є співвідношення (2.63).

Перепишемо контрольні суми (2.73) іншим чином, виражаючи значення контрольних розрядів через інформаційні. З урахуванням даних, наведених у таблиці 2.13, можна записати наступні вирази для значень контрольних розрядів:

$$\begin{cases} r_1 = n_1 = n_3 \oplus n_5 \oplus n_7 \oplus n_9 \oplus n_{11} \oplus n_{13} \oplus n_{15} \oplus n_{17} \oplus n_{19} \oplus \\ \quad \oplus n_{21} \oplus n_{23} \oplus n_{25} \oplus n_{27} \oplus n_{29} \oplus n_{31} = k_1 \oplus k_2 \oplus k_4 \oplus \\ \quad \oplus k_5 \oplus k_7 \oplus k_9 \oplus k_{11} \oplus k_{12} \oplus k_{14} \oplus k_{16} \oplus k_{18} \oplus k_{20} \oplus \\ \quad \oplus k_{22} \oplus k_{24} \oplus k_{26}; \\ r_2 = n_2 = n_3 \oplus n_6 \oplus n_7 \oplus n_{10} \oplus n_{11} \oplus n_{14} \oplus n_{15} \oplus n_{18} \oplus n_{19} \oplus \\ \quad \oplus n_{22} \oplus n_{23} \oplus n_{26} \oplus n_{27} \oplus n_{30} \oplus n_{31} = k_1 \oplus k_3 \oplus k_4 \oplus \\ \quad \oplus k_6 \oplus k_7 \oplus k_{10} \oplus k_{11} \oplus k_{13} \oplus k_{14} \oplus k_{17} \oplus k_{18} \oplus \\ \quad \oplus k_{21} \oplus k_{22} \oplus k_{25} \oplus k_{26}; \\ r_3 = n_4 = n_5 \oplus n_6 \oplus n_7 \oplus n_{12} \oplus n_{13} \oplus n_{14} \oplus n_{15} \oplus n_{20} \oplus n_{21} \oplus \\ \quad \oplus n_{22} \oplus n_{23} \oplus n_{28} \oplus n_{29} \oplus n_{30} \oplus n_{31} = k_2 \oplus k_3 \oplus k_4 \oplus \\ \quad \oplus k_8 \oplus k_9 \oplus k_{10} \oplus k_{11} \oplus k_{15} \oplus k_{16} \oplus k_{17} \oplus k_{18} \oplus k_{23} \oplus \\ \quad \oplus k_{24} \oplus k_{25} \oplus k_{26}; \\ r_4 = n_8 = n_9 \oplus n_{10} \oplus n_{11} \oplus n_{12} \oplus n_{13} \oplus n_{14} \oplus n_{15} \oplus n_{24} \oplus n_{25} \oplus \\ \quad \oplus n_{26} \oplus n_{27} \oplus n_{28} \oplus n_{29} \oplus n_{30} \oplus n_{31} = k_5 \oplus k_6 \oplus k_7 \oplus \\ \quad \oplus k_8 \oplus k_9 \oplus k_{10} \oplus k_{11} \oplus k_{19} \oplus k_{20} \oplus k_{21} \oplus k_{22} \oplus k_{23} \oplus \\ \quad \oplus k_{24} \oplus k_{25} \oplus k_{26}; \\ r_5 = n_{16} = \oplus n_{17} \oplus n_{18} \oplus n_{19} \oplus n_{20} \oplus n_{21} \oplus n_{22} \oplus n_{23} \oplus n_{24} \oplus \\ \quad \oplus n_{25} \oplus n_{26} \oplus n_{27} \oplus n_{28} \oplus n_{29} \oplus n_{30} \oplus n_{31} = k_{12} \oplus k_{13} \oplus \\ \quad \oplus k_{14} \oplus k_{15} \oplus k_{16} \oplus k_{17} \oplus k_{18} \oplus k_{19} \oplus k_{20} \oplus k_{21} \oplus k_{22} \oplus \\ \quad \oplus k_{23} \oplus k_{24} \oplus k_{25} \oplus k_{26}. \end{cases} \qquad (2.74)$$

Отримані співвідношення (2.74) можуть бути використані для формування кодових послідовностей визначених інформаційних слів, а співвідношення (2.73) – для формування синдромів помилки та декодування кодових послідовностей. Вектор синдрому помилки для стандартного коду із виправленням одиночних помилок формується наступним чином:

$$\mathbf{S_e} = [S_5, S_4, S_3, S_2, S_1]. \qquad (2.75)$$

Для формування контрольних сум та обчислення значень контрольних розрядів модифікованого коду Хеммінга, який дозволяє виправляти одиночні та виявляти подвійні помилки, разом із співвідношеннями (2.73), (2.74) використовується співвідношення (2.64), а пошук синдрому помилки здійснюється згідно з даними, наведеними у таблиці 2.9.

Вкрай важливим для коректного виконання обчислювальних процесів створення кодів Хеммінга та декодування їхніх послідовностей, як і в усій теорії кодування, є порядок слідування бітів числа. Ця проблема неодноразово обговорювалась у п'ятому розділі другої частини посібника та у підрозділі 1.3.4 цієї частини. Сутність її полягає у тому, що у арифметичному запису двійкових чисел зазвичай використовується порядок читання розрядів числа зправа-наліво, а під час обробки векторів – зворотний порядок, зліва-направо. Оскільки, як відмічалось в підрозділі 1.3.4, зміна порядку слідування розрядів числа є простою обчислювальною процедурою і реалізується одним командним рядком системи MatLab, будемо дотримуватись під час написання програми таких правил. Для ведення та виведення двійкових послідовностей, з метою зручності їхнього наочного сприйняття, будемо використовувати прямий порядок слідування розрядів числа, до якого звикли більшість користувачів програмного забезпечення, включаючи математиків та інженерів. А внутрішню обробку кодових послідовностей з застосуванням векторних макроопераций системи MatLab та засобів структурного програмування [13, 14] зручніше реалізовувати через використання зворотного порядку запису двійкових чисел. Слід відзначити, що форма подання синдрому помилки (2.75) також відповідає прямому порядку запису бітів числа, хоча цей синдром записується він у вигляді вектора. За умови використання такого порядку слідування розрядів у двійкових послідовностях структура програми буде мати наступний вигляд.

1. Введення вхідної кодової послідовності з використанням прямого порядку запису бітів числа.

2. Зміна порядку запису бітів числа на зворотний.

3. Обробка бітів числа з використанням обчислювальних процедур формування коду або декодування кодової послідовності, залежно від типу визначеної операції.

4. Зміна порядку слідування бітів в векторі отриманого результату із зворотного на прямий.

5. Виведення результатів обчислень на екран.

Наприклад, розглянемо двійкове число 1011. Модифікований код

Хеммінга для цього числа був сформований у прикладі 2.9, це кодова послідовність 01010101. Для спрощення проведення дій над векторами вхідну послідовність 1011 перепишемо у зворотному порядку, це буде послідовність [1,1,0,1]. Для такої послідовності перший елемент вектора відповідає першому розряду числа, другий – другому, останній – останньому розряду. Простота проведення обчислень з використанням такого зворотного подання полягає в тому, що для обчислення контрольних розрядів кодової комбінації можна безпосередньо використовувати співвідношення (2.63), (2.64), не змінюючи номера елементів у контрольних сумах. Але тоді результат розрахунків буде записаний не у вигляді (2.65), тобто

$$\mathbf{C} = [r_4, 1, 0, 1, r_3, 1, r_2, r_1],$$

а у зворотному порядку,

$$\mathbf{C}_{\text{зв}} = [r_1, r_2, 1, r_3, 1, 0, 1, r_4].$$

Тобто, замість кодової комбінації 01010101, в результаті розрахунків отримуємо вектор із зворотною послідовністю бітів [1, 0, 1, 0, 1, 0, 1, 0]. Тепер, перед виведенням результату, для спрощення читання коду, варто змінити зворотний порядок слідування бітів на прямий.

Єдина проблема полягає у тому, що якщо в процесі обробки результатів розрахунків користувач хоче змінити один із бітів числа, він повинен пам’ятати про те, що у вихідному векторі біти записані у зворотному порядку. Наприклад, операція **C**(8) = 0 означає, що змінюється перший розряд коду, оскільки згідно з адресацією елементів вектора – це восьмий розряд. Тобто, в результаті виконання цієї операції над послідовністю бітів 01010101 буде отримана послідовність 01010100. Особливо важливо пам’ятати про цю особливість адресації, коли змінений вектор кодової комбінації **C** використовується для подальших обчислень, наприклад, для декодування кодової комбінації 01010100 та для пошуку позиції помилки у спотвореній кодовій комбінації.

З урахуванням наведених вище міркувань, алгоритм кодування інформаційних слів довжиною від 4 до 26 бітів можна відобразити у вигляді блок – схеми, наведеної на рис. 2.14. Відповідний алгоритм декодування кодових послідовностей довжиною від 7 до 32 бітів наведений на рис. 2.15.

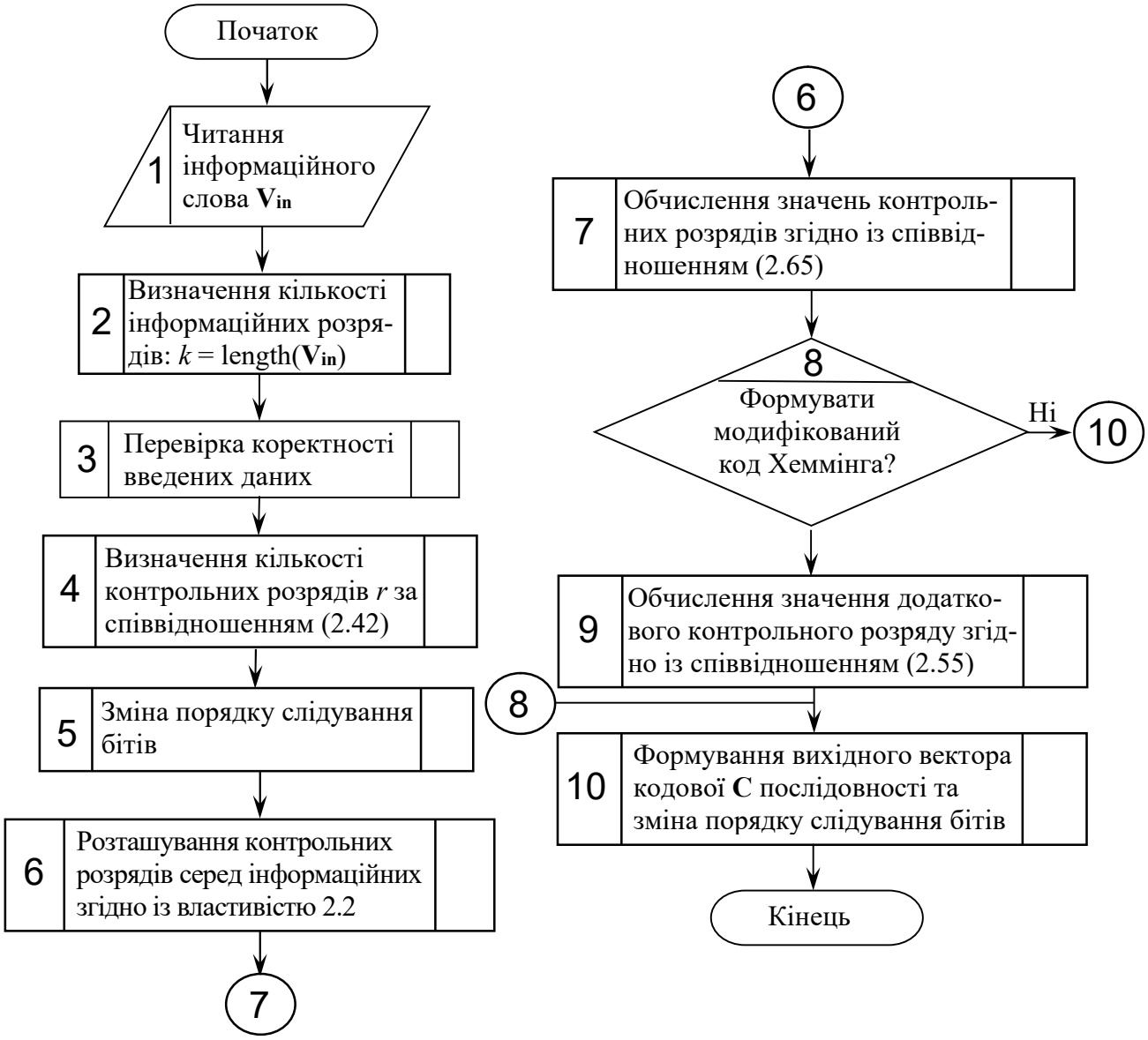


Рис. 2.14 Узагальнений алгоритм формування коду Хеммінга

Програма **`Hamming_Coding`**, призначена для формування коду Хеммінга та декодування його кодових послідовностей, написана на мові програмування системи науково-технічних розрахунків MatLab, а також тестові результати роботи цієї програми, наведена у додатку З. Програма написана з використанням засобів структурного та модульного програмування системи MatLab. Розглянемо головні особливості цієї програми. В програмі **`Hamming_Coding`** реалізовані описані вище алгоритми формування кодів Хеммінга та декодування їхніх кодових послідовностей.

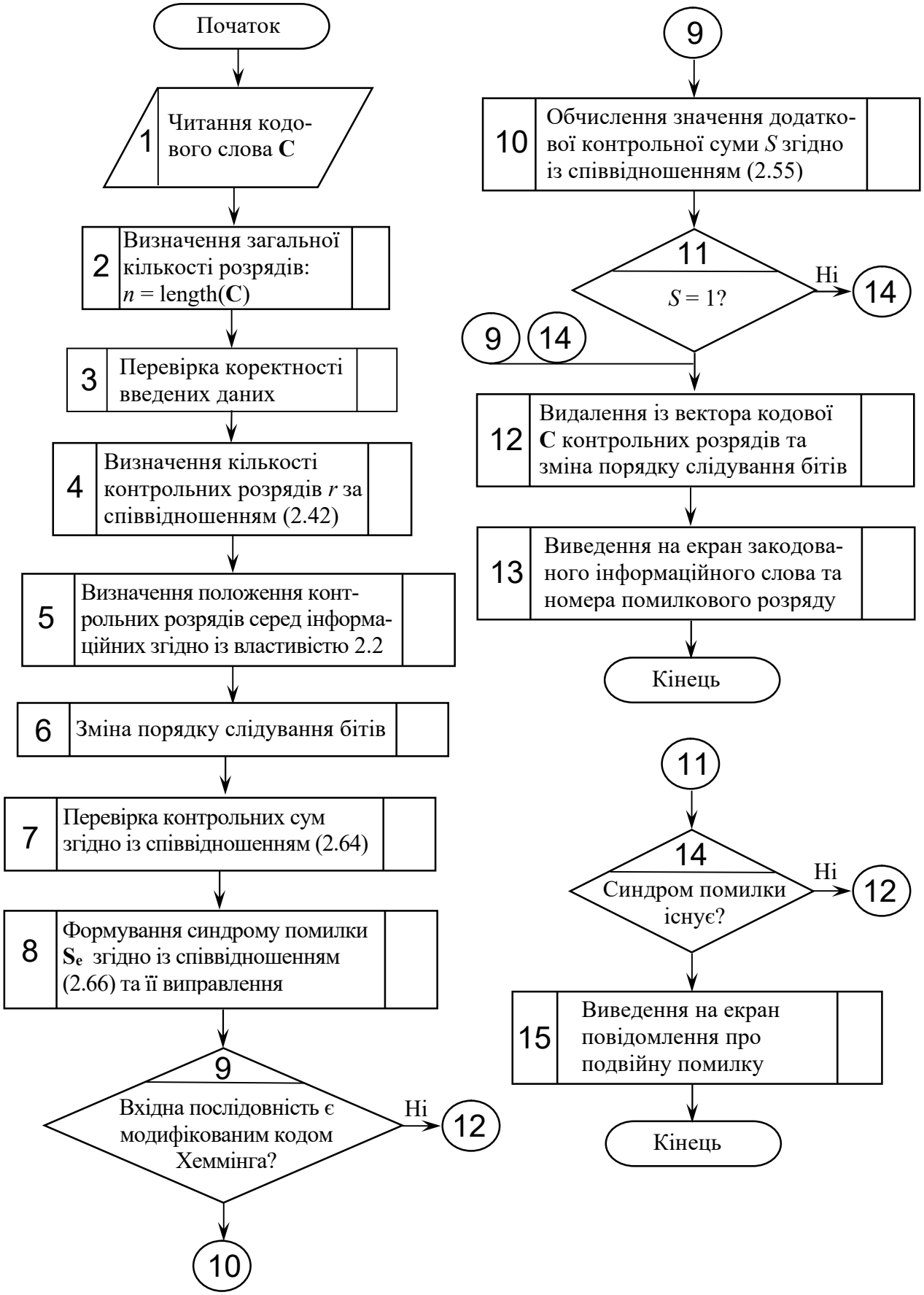


Рис. 2.15 Узагальнений алгоритм декодування послідовностей коду Хеммінга

1. Для виклику функції **Hamming_Coding** використовуються три вхідних параметри: вектор вхідної послідовності або кодової комбінації, номер операції та тип коду. Параметри номера операції **nop** та типу коду **TOC** можуть мати наступні значення:

nop = 1 – виконати операцію кодування;

nop = 2 – виконати операцію декодування;

TOC = 1 – для стандартного коду Хеммінга;

TOC = 2 – для модифікованого коду Хеммінга.

Наприклад, команда **Hamming_Coding ([1,0,1,1,1],1,2)** означає, що необхідно сформувати модифікований код Хеммінга для числа 10111.

2. Для введення та виведення даних використовується прямий порядок слідування бітів числа, зправа-наліво, а для проведення операцій над векторами – зворотний порядок, зліва-направо.

3. Під час формування кодової послідовності спочатку вважається, що всі значення контрольних розрядів дорівнюють 0. У цьому випадку нульові значення цих розрядів не впливають на контрольні суми (2.73), а обчислені значення цих сум відповідають дійсним значенням контрольних розрядів.

4. Розрахунок контрольних розрядів за співвідношеннями (2.74) та контрольних сум за співвідношеннями (2.73) виконується за однаковим алгоритмом через виклик окремої функції **Hamming_Summing**. Оскільки під час формування коду початково вважається, що значення контрольних розрядів кодової комбінації дорівнюють 0, співвідношення (2.73) та (2.74) є еквівалентними. Використання засобів модульного програмування системи MatLab у програмі **Hamming_Coding** дозволяє у значній мірі скоротити та спростити програмний код.

5. Якщо кодова послідовність, яка формується, містить менше, ніж 31 розряд, на етапі проведення розрахунків вважається, що у сумах (2.73) та (2.74) старші розряди коду, які не використовуються, також дорівнюють 0.

Перед виконанням функції виведення значення отриманого вектора на екран ці зайві розряди відкидаються. Такий підхід дозволяє уникнути зайвих перевірок та спростити алгоритм обчислення контрольних сум.

6. Під час декодування кодових послідовностей значення помилкового розряду кодової комбінації розраховується через синдром помилки (2.75) з використанням співвідношення (1.18), тобто:

$$n_{\text{пом}} = 16 \cdot S_5 + 8 \cdot S_4 + 4 \cdot S_3, + 2 \cdot S_2 + S_1.$$

7. У разі декодування послідовностей модифікованого коду Хеммінга для пошуку синдромів помилки використовуються дані, наведені у таблиці 2.9.

8. У разі виконання функції кодування результатом роботи програми є сформований код, а у разі виконання функції декодування – початкове інформаційне слово, яке було закодоване, та позиція розряду, у якому виявлена та виправлена помилка.

Всі розглянуті особливості написаної програми у загальному вигляді відображені на блок-схемах алгоритмів, які наведені на рис. 2.14 та 2.15. Особливості роботи з програмою та її використання можна побачити через аналіз відповідних програмних рядків, наведених у додатку З.

2.5 Циклічні коди

Перед вивченням цього підрозділу необхідно повторити перший, третій, четвертий, п'ятий та сьомий розділи другої частини посібника

2.5.1 Загальне поняття про циклічні коди та спосіб їх побудови з використанням алгебраїчних операцій над двійковими поліномами

Загалом способи формування циклічних кодів будуються на теорії чисел, теорії груп та теорії поліномів, також вони пов'язані із теорією циклічних пересувань та теорією кілець. Відповідний математичний апарат був досконало розглянутий у другій частині посібника, а узагальнена теорія формування циклічних кодів була наведена у підрозділі 5.3 другої частини посібника [48].

Взагалі узагальнена теорія циклічних та лінійних кодів тісно пов'язані між собою. У підрозділі 2.4.4 було показано, що будь-який систематичний код (n, k) може бути записаний через твірну матрицю, яка містить k лінійно незалежних рядків по n стовпчиків у кожному. Окремий клас циклічних кодів відрізняється тим, що незалежні рядки твірної матриці задовольняють умові циклічності. Ця умова полягає у тому, що кожний із рядків твірної матриці можна розглядати як результат циклічного зсуву ліворуч елементів першого рядка на відповідну кількість позицій. Наведемо приклад такої матриці **G**. Припустимо, що у першому рядку записана початкова кодова комбінація 001011. Тоді, з урахуванням правила циклічного зсуву ліворуч, матриця **G** буде мати наступний вигляд:

$$\mathbf{G} = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

Проте, на відміну від лінійних кодів, для формування циклічних кодів матричне подання використовується вкрай рідко, частіше кодові комбінації розрядності n подаються у вигляді двійкових поліномів степені n відносно фіктивної змінної x. Теорія двійкових поліномів та алгебраїчні операції над ними були розглянуті у третьому розділі другої частини посібника, а спосіб використання поліноміального подання двійкових чисел для формування циклічних кодів – у підрозділі 5.3 другої частини посібника [48]. Нагадаємо головні положення теорії двійкових поліномів та алгебраїчних операцій над ними, які необхідно знати для розуміння способів побудови циклічних кодів.

Спочатку сформулюємо головну властивість циклічних кодів, яка пов'язана із розглянутим вище принципом циклічності дозволених кодових комбінацій [5, 16].

Властивість 2.7. У разі циклічного зсуву праворуч або ліворуч будь-

якої дозволеної комбінації циклічного коду створюється інша дозволена кодова комбінація.

Інакше кажучи, якщо будь-який вектор $\mathbf{V} = (a_0, a_1, a_2, \ldots, a_{n-1}, a_n)$ є коректною кодовою комбінацією циклічного коду, тоді дозволеною комбінацією цього коду буде також вектор $\mathbf{V'} = (a_n, a_0, a_1, a_2, \ldots, a_{n-1})$. Це дуже зручно з технічної точки зору, оскільки операція циклічного зсуву є однією із головних в електронній обчислювальній апаратурі [39, 40, 45, 46], тому апаратна реалізація алгоритмів кодування циклічних кодів вкрай спрощується. Також операція циклічного зсуву бітів числа у разі його векторного подання може бути легко реалізована з використанням матричних макрооперацій системи науково-технічних розрахунків MatLab, що у значній мірі спрощує реалізацію алгоритмів формування циклічного коду та декодування його кодових послідовностей з використанням сучасних засобів програмування.

Надамо визначення алгебраїчного двійкового поліному [5, 16, 64, 65].

Визначення 2.7. Якщо розряди двійкового числа записати у вигляді вектора $[a_{n-1}, a_{n-2}, a_{n-3}, \ldots a_2, a_1, a_0]$, тоді еквівалентне подання такого числа забезпечується через двійковий поліном виду:

$$G(x) = a_{n-1} \cdot x^{n-1} + a_{n-2} \cdot x^{n-2} + \ldots + a_1 \cdot x + a_0. \qquad (2.76)$$

Подання двійкових кодів через поліном (2.76) значно спрощується, оскільки коефіцієнти $a_{n-1}, a_{n-2}, a_{n-3}, \ldots a_2, a_1, a_0$ можуть приймати лише два значення: 0 або 1. Наприклад, число 1010101 у поліноміальній формі записується так:

$$1010101 = 1 \cdot x^6 + 0 \cdot x^5 + 1 \cdot x^4 + 0 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x + 1 = x^6 + x^4 + x^2 + 1.$$

Сформулюємо головні правила роботи з двійковими поліномами під час виконання елементарних арифметичних дій.

1. Складання та віднімання завжди зводиться до сумування за модулем 2 коефіцієнтів при рівних степенях фіктивної змінної x.

2. Множення здійснюється за звичайними правилами множення поліномів, але під час виконання цієї операції коефіцієнти при рівних

степенях сумуються за модулем 2.

3. Ділення поліномів здійснюється за стандартною схемою Горнера [48, 51] з урахуванням остач, але під час реалізації цього алгоритму всі операції віднімання також заміняються на сумування за модулем 2.

4. Операція циклічного пересування змінних розглядається як множення або ділення полінома на фіктивну змінну x.

З першого погляду здається, що операція множення двійкових поліномів не є зімкненою на заданій множені і не відповідає правилам теорії полів Галуа, розглянутій у другому розділі другої частини посібника [48]. Дійсно, з точки зору класичної алгебри результатом множення поліному степені n на поліном степені r є поліном степені $n + r$, а це означає, що кількість розрядів у кодовій комбінації збільшується. Проте якщо звернутися до теорії циклотомічних класів, яка була розглянута у підрозділі 3.6.10 другої частини посібника, стає зрозумілим, що у двійковій арифметиці це алгебраїчне правило не виконується. Це пов'язано з тим, що, починаючи з відповідного граничного числа, степені елементів скінченного поля Галуа починають циклічно повторюватись, а не збільшуються до нескінченності. Враховуючи принцип циклічності, сформульоване вище правила множення двійкових поліномів можна переписати наступним чином [5, 64, 65].

1. Двійкові поліноми множаться за звичайними алгебраїчними правилами, але зведення подібних членів здійснюється через сумування за модулем два.

2. Якщо старша степінь добутку поліномів не перевищує граничного значення n для циклотомічного класу, тоді результат множення залишається без змін.

3. Якщо старша степінь добутку поліномів є вищою за граничне значення n для циклотомічного класу, тоді результат ділиться на поліном степені n і остаточним результатом вважається результат цього ділення.

У цьому аспекті циклотомічні класи для двійкових поліномів є аналогом класів лишків в алгебрі чисел. Дійсно, якщо степінь отриманого

поліному перевищує задану граничну степінь, остачі від ділення починають повторюватись і можна розглядати остачу як ознаку результату множення поліномів. Відповідний математичний апарат був розглянутий у третьому розділі другої частини посібника.

Враховуючи можливість поліноміального подання двійкових чисел та вважаючи, що, як і раніше, кількість інформаційних розрядів дорівнює n, а контрольних – k, можна дати таке визначення циклічного коду [2 – 5, 16, 64, 65].

Визначення 2.8. Циклічним кодом (англійський термін – cyclic code, recurrent code) називається код розмірності (n, k), який створюється множенням початкового поліному $P(x)$ розмірності k на твірний поліном $Q(x)$ розмірності $n - k$, або у математичній формі:

$$S(x) = Q(x) \cdot P(x). \tag{2.77}$$

Декодування циклічного коду проводиться у зворотному порядку. Для того, щоб знайти закодований поліном, достатньо поділити результат кодування $S(x)$ на твірний поліном $Q(x)$, який завжди є відомим. Відсутність остачі в процесі ділення свідчить про те, що код прийнятий без спотворень і частка являє собою закодоване число. Якщо ж код спотворений і остача існує, помилка шукається з використанням алгоритму зсувів бітів коду ліворуч, які відповідають множенню результуючого полінома на фіктивну змінну x, та послідовного ділення результатів такого зсуву на твірний поліном. У загальному вигляді цей алгоритм можна записати наступним чином:

$$\frac{S(x) \cdot x^n}{P(x)} = Q(x) + \frac{R(x)}{P(x)}, \tag{2.78}$$

де $R(x)$ – остача, або

$$S(x) \cdot x^n = Q(x) \cdot P(x) + R(x). \tag{2.79}$$

Особливості практичної реалізації алгоритму ділення (2.79) будуть розглянуті далі.

Під час побудови циклічних кодів за співвідношенням (2.79) множення початкового поліному $P(x)$ можна здійснювати лише на незвідні твірні поліноми $Q(x)$, і при тому тільки на такі, степені яких є дільниками розрядності початкового числа. Визначення незвідного поліному наводилось у третьому розділі другої частини посібника. Нагадаємо його.

Визначення 2.9. Двійковий поліном називається незвідним (англійський термін – irreducible polynomial), якщо він не має жодного іншого дільника, крім одиниці та самого себе.

Зрозуміло, що незвідні поліноми в алгебрі є аналогом простих чисел у арифметиці.

Наприклад, для трьохрозрядного числа необхідно брати поліноми степенів 1 та 3. Всі незвідні поліноми першого, другого, третього та четвертого ступенів наведені у таблиці 2.14 [2 – 5, 16, 64, 65]. Таблиця незвідних поліномів більш високих порядків, до восьмого, наведена у додатку І [5, 64, 65].

Таблиця 2.14 – Твірні незвідні поліноми

<table>
<tr><th>Цифрове подання</th><th>Поліноміальне подання</th><th rowspan="2">Цифрове подання</th><th rowspan="2">Поліноміальне подання</th></tr>
<tr><td colspan="2">Поліноми першої степені</td></tr>
<tr><td>10</td><td>$x + 1$</td><td colspan="2" rowspan="2"></td></tr>
<tr><td colspan="2">Поліноми другої степені</td><td colspan="2">Поліноми четвертої степені</td></tr>
<tr><td>111</td><td>$x^2 + x + 1$</td><td>10011</td><td>$x^4 + x + 1$</td></tr>
<tr><td colspan="2">Поліноми третьої степені</td><td>11111</td><td>$x^4 + x^3 + x^2 + x + 1$</td></tr>
<tr><td>1011</td><td>$x^3 + x + 1$</td><td rowspan="2">11001</td><td rowspan="2">$x^4 + x^3 + 1$</td></tr>
<tr><td>1101</td><td>$x^3 + x^2 + 1$</td></tr>
</table>

Необхідність обрання незвідних поліномів під час використання співвідношення (2.79) обумовлена тим, що вони забезпечують найбільшу кількість остач в процесі ділення, тому імовірність помилкового співпадання є найменшою. Відповідна теорія класів лишків була розглянута у розділі 1 другої частини посібника, а теорія незвідних поліномів – у розділі 3 [48].

Кількість контрольних символів для циклічних кодів, які виправляють одиночні помилки, тобто, згідно з теорією кодування, мають мінімальну кодову відстань $d_{\min} = 3$, обчислюється через співвідношення:

$$r = \left[\log_2\left((k+1) + \log_2(k+1)\right)\right]. \qquad (2.80)$$

Зрозуміло, що співвідношення (2.80) безпосередньо випливає із границі Хеммінга, яка визначається співвідношенням (2.42).

Для невеликих значень n та k замість співвідношення (2.80) можна користуватися таблицею 2.15 [2 – 5, 16, 64, 65]. Дані, наведені у таблиці 2.15, є досить універсальними, їх можна використовувати і при формуванні кодів Хеммінга.

Таблиця 2.15 – Кількість контрольних символів у завадостійких кодах з виправленням одиночної помилки

Параметри коду	Значення									
Загальна кількість розрядів n	3	4	5	6	7	8	9	10	11	12
Кількість інформаційних розрядів k	1	1	2	3	4	4	5	6	7	8
Кількість контрольних розрядів r	2	3	3	3	3	4	4	4	4	4

Алгоритм виправлення помилок у циклічних кодах є досить складним і у вигляді тезових формулювань його можна записати наступним чином.

1. Якщо прийнята кодова комбінація ділиться на твірний поліном без остачі, можна вважати, що код був прийнятий без помилок.

2. Якщо існує остача, необхідно підрахувати її вагу w.

3. Позначимо коректувальну здатність коду, тобто кількість помилок, які він повинен виправляти, через j. Якщо $w \le j$, необхідно, згідно із

співвідношенням (2.79), додати остачу до отриманої кодової комбінації, це і буде правильний результат.

4. Якщо $w > j$, необхідно виконати операцію циклічного зсуву кодової комбінації на 1 розряд ліворуч та знову розділити отриманий результат на твірний поліном.

5. Якщо в результаті ділення за пунктом 4 алгоритму вага остачі $w > j$, знову повертаємося до пункту 4, а у противному випадку переходимо до пункту 6.

6. Додаємо остачу до дільника (тобто до спотвореної кодової комбінації, зсунутої на відповідну кількість позицій, згідно з пунктом 4 алгоритму).

7. Виконуємо зворотний циклічний зсув отриманої кодової комбінації праворуч на таку ж саму кількість позицій, на скільки вона була зсунута ліворуч під час виконання пункту 4 алгоритму.

Блок-схема описаного алгоритму виправлення помилки у циклічному коді наведена на рис. 2.16.

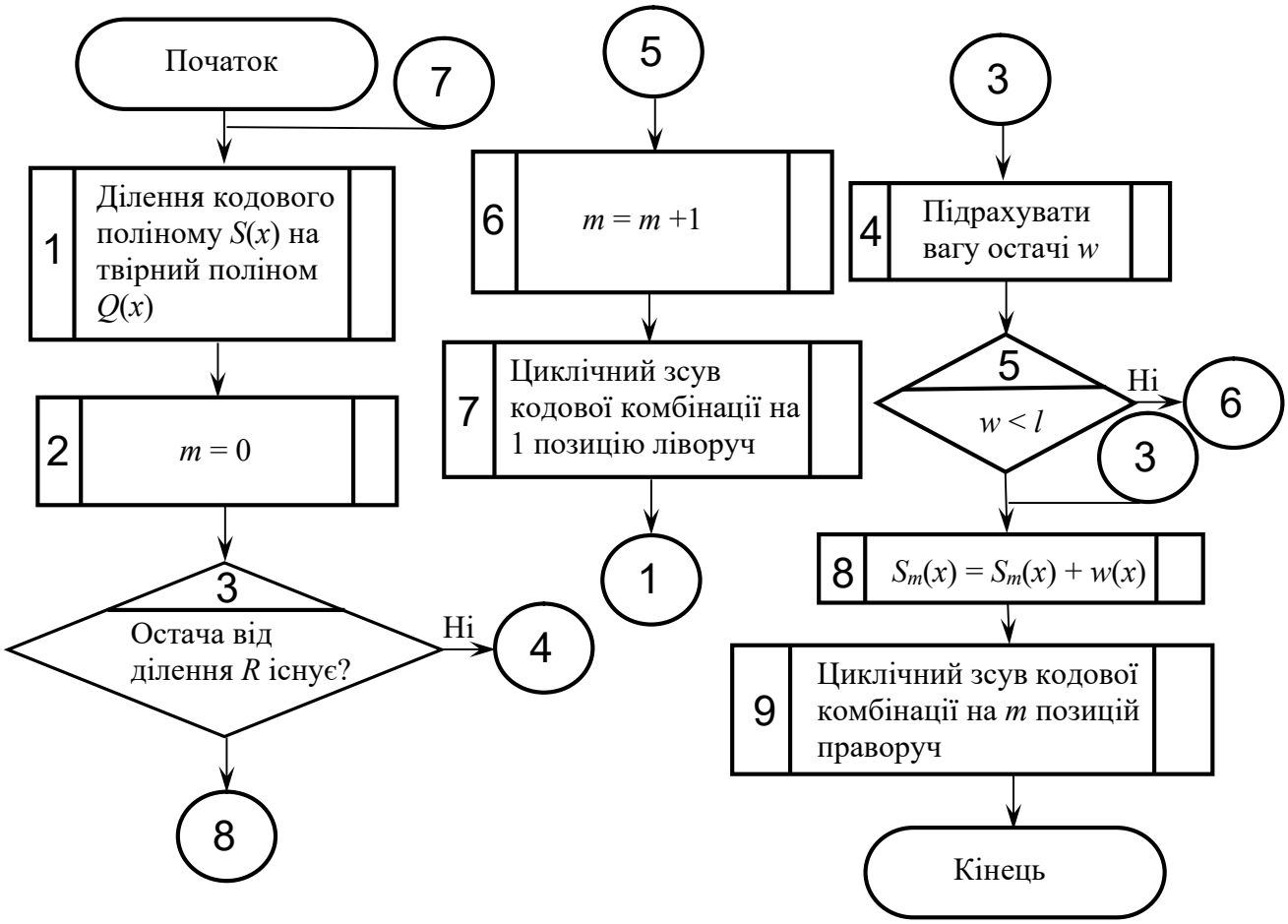


Рис. 2.16 Алгоритм виправлення помилки у кодовій комбінації циклічного коду, оснований на діленні двійкових поліномів

2.5.2 Головні властивості циклічних кодів та відповідні теореми теорії кодування

Наведений алгоритм формування циклічного коду потребує серйозного теоретичного обґрунтування з точки зору теорії двійкових поліномів. Тому розглянемо тепер деякі важливі властивості алгебраїчних операцій над двійковими поліномами, які використовуються в процесі формування циклічних кодів. Спочатку необхідно визначити операції над двійковими поліномами, які відповідають циклічному зсуву бітів числа ліворуч. Будемо виходити із наступних міркувань. Зрозуміло, що множення будь-якого поліному $G(x)$ степені $n-1$ на x дає наступний результат:

$$G(x) = (x^{n-1} + x^{n-2} + \ldots + x + 1)\, x = x^n + x^{n-1} + \ldots + x^2 + x. \qquad (2.81)$$

Для того, щоб результат такого множення відповідав для поліному $G(x)$ циклічному зсуву ліворуч на одну позицію, необхідно, щоб виконувалась тотожність $x^n = 1$. Така заміна еквівалентна остачі від ділення многочлена $G(x)$ на поліном $x^n + 1$. Тобто, можна сформулювати наступну властивість алгебраїчних операцій над поліномами [5, 52, 53, 55, 62 – 65].

Властивість 2.8. Результатом множення поліному на x та взяття остачі від ділення результату на цього множення $x^n + 1$ є циклічний зсув бітів числа ліворуч.

Надамо тепер деякі важливі визначення, які використовуються у теорії формування циклічних кодів [5, 52, 53, 55, 62 – 65].

Визначення 2.10. Підмножина всіх поліномів, які є кратними деякому поліному $g(x)$, називається ідеалом (англійський термін – ideal).

Визначення 2.11. Поліном $g(x)$, який створює визначений ідеал, називається породжувальним поліномом ідеалу (англійський термін – generator polynomial of ideal).

Із наведених визначень зрозуміло, що розглянутий у попередньому підрозділі циклічний код є ідеалом, а головним завданням формування такого коду є пошук породжувального поліному $g(x)$.

Згідно із визначенням циклічного коду 2.8 та співвідношеннями (2.78),

(2.79), всі поліноми, які є правильними кодовими комбінаціями, мають ділитися на поліном $g(x)$ без остачі. Враховуючи те, що правильні кодові комбінації, згідно із визначенням циклічного коду, є результатом циклічного зсуву бітів числа ліворуч, можна, враховуючи співвідношення (2.81), записати наступний вираз:

$$g(x) = g(x)\cdot x^i + c\cdot(x^n + 1), \qquad (2.82)$$

де $c = 1$, якщо степінь поліному є меншою за $n - 1$, або $c = 0$, якщо степінь цього поліному є більшою за $n - 1$.

Із формули (2.82) випливають наступні теореми теорії циклічних кодів [5].

Теорема 2.3. Всі поліноми, які створені в результаті циклічного зсуву ліворуч заданої кодової комбінації, діляться без остачі на твірний поліном $g(x)$ тоді та лише тоді, коли $g(x)$ ділиться без остачі на поліном $x^n + 1$.

Теорема 2.4. Якщо поліном $g(x)$ степені $n - k$ є дільником поліному $x^n + 1$ та кодова комбінація $S(x)$ є елементом ідеалу, тоді $S(x)$ також ділиться без остачі на поліном $g(x)$. Тобто, $g(x)$ є твірним поліномом.

Теорема 2.5. Якщо поліном $g(x)$ степені $m = n - k$ є дільником поліному $x^n + 1$ та кодова комбінація $S(x)$ не є елементом ідеалу, тоді остачею від ділення поліному $S(x)$ на поліном $g(x)$ є поліном $r(x)$, степінь якого завжди є меншою за $m - 1$.

Як було відмічено у попередньому підрозділі, коректувальна здатність циклічного коду визначається тим, скільки остач може виникати під час ділення помилкових кодових комбінацій, які не належать до ідеалу, на твірний поліном $g(x)$. Тому як твірні завжди використовують незвідні поліноми, оскільки вони дають найбільшу кількість остач, яка становить $2^m - 1$.

2.5.3 Приклад побудови циклічного коду та пошуку помилки в ньому

Розглянемо простий практичний приклад формування циклічного коду та пошуку помилки у ньому [16].

Приклад 2.16. Закодувати з використанням циклічного коду десяткове число 10_{10}.

Спочатку слід записати число 10_{10} у вигляді двійкового поліному, що робиться наступним чином:

$$10_{10} = 1010_2 = x^3 + x.$$

Оскільки $k = 4$, згідно з таблицею 2.15 кількість контрольних розрядів буде 3. Тобто, із таблиці 2.14 необхідно обрати твірний поліном третього ступеня. Таких поліномів два, і можна використати або перший, або другий. Нехай це буде поліном $x^3 + x + 1$. Використовуючи співвідношення (2.77), створюємо циклічний код.

$$(x^3 + x)\cdot(x^3 + x + 1) = x^6 + x^4 + x^3 + x^4 + x^2 + x = x^6 + x^3 + x^2 + x =$$
$$= 1\cdot x^6 + 0\cdot x^5 + 0 + x^4 + 1\cdot x^3 + 1\cdot x^2 + 1\cdot x + 0 = 1001110.$$

Циклічний код сформований. Тепер перевіримо його коректувальну здатність. Спочатку визначимо, чи ділиться отримана кодова комбінація 1001110 на твірний поліном 1011. Зрозуміло, що тут ділення здійснюється без остачі і часткою є початкове число 1010_2. Оскільки процес ділення поліномів за схемою Горнера, розглянутий у третьому розділі другої частини посібника [48], є цілком еквівалентним процесу ділення двійкових чисел, надалі будемо ділити числа, оскільки ділення чисел є більш простим та наочним, ніж ділення поліномів. Процес ділення показаний на рис. 2.17.

$$\begin{array}{r|l} \oplus\ 1001110 & 1011 \\ \underline{1011} & \hline 1010 \\ \oplus\ 1011 & \\ \underline{1011} & \\ 0 & \end{array}$$

Рис. 2.17 Ділення циклічного коду на твірний поліном, частка – початкове число

Тепер припустимо, що під час передавання даних у четвертому розряді коду виникла помилка, в результаті якої прийнята спотворена кодова комбінація 1000110. Згідно з наведеним алгоритмом, наведеним на рис. 2.17, проведемо перевірку прийнятої кодової комбінації шляхом ділення на твірний поліном 1011. Процес ділення показаний на рис. 2.18.

Як бачимо, тепер в процесі ділення отримана остача 11, а вага остачі $w = 2$, тобто $w > j$. У цьому випадку, згідно із пунктом 4 алгоритму, необхідно виконати циклічний зсув кодової комбінації ліворуч на одну позицію. Цей процес показаний на рис. 2.19.

$$
\begin{array}{l}
\oplus\,1000110 \;\big|\; 1011 \\
\underline{\;1011\;\;} \;\big|\; \overline{\;1011\;} \\
\oplus\,1111 \\
\underline{\;\;1011} \\
\oplus\,1000 \\
\underline{\;\;\;1011} \\
\;\;\;11
\end{array}
$$

Рис. 2.18 Ділення спотвореного циклічного коду на твірний поліном

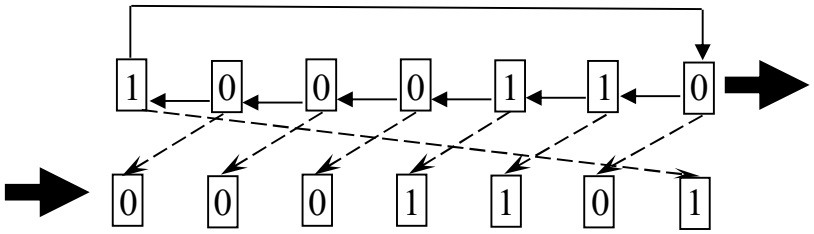


Рис. 2.19 Ілюстрація виконання операції циклічного зсуву ліворуч числа 1000110 на одну позицію

Тобто, в результаті операції циклічного зсуву ліворуч спотвореного коду на одну позицію одержуємо число 0001101. Відкидати послідовності нулів у старших розрядах для циклічного коду неприпустимо, оскільки, як ми побачимо надалі, на наступних циклічних зсувах вони переходять до молодших розрядів числа і впливають на результати ділення.

Тепер необхідно здійснити другу операцію ділення, тобто поділити отриманий результат зсуву 0001101 на твірний поліном 1011. Цей процес ділення показаний на рис. 2.20.

Як бачимо, знову вага остачі дорівнює 2 і перевищує коректувальну здатність коду. Тобто, процеси зсуву та ділення за пунктом 4 наведеного алгоритму необхідно продовжувати. Тому здійснюємо зсув отриманої

кодової комбінації 0001101 ще на одну позицію ліворуч. Цей процес і результат зсуву наочно показані на рис. 2.21.

$$\begin{array}{r|l} \oplus 0001101 & 1011 \\ \underline{\quad 1011} & \overline{\quad\quad 1} \\ 0110 & \end{array}$$

Рис. 2.20 Ділення результату зсуву спотвореної комбінації циклічного коду на твірний поліном

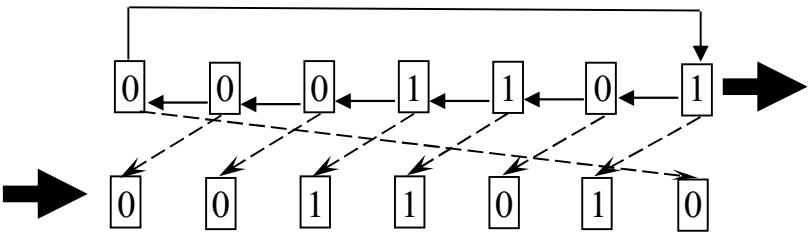


Рис. 2.21 Ілюстрація виконання операції циклічного зсуву ліворуч числа 0001101 на одну позицію

Отримане число 0011010 ще раз ділимо на твірний поліном 1011, процес ділення та результат наведені на рис. 2.22.

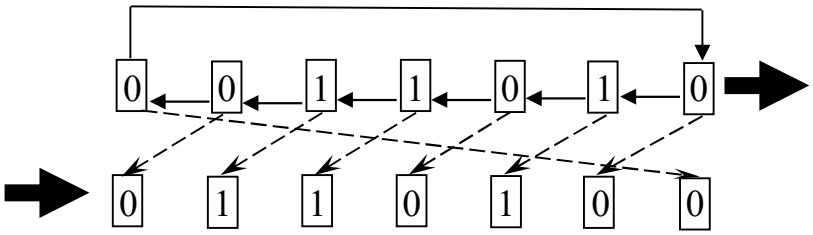
$$\begin{array}{r|l} \oplus 0011010 & 1011 \\ \underline{\quad 1011} & \overline{\quad\quad 11} \\ \oplus 1100 & \\ \underline{\quad 1011} & \\ 111 & \end{array}$$

Рис. 2.22 – Ділення результату подвійного зсуву спотвореної комбінації циклічного коду на твірний поліном

І тепер остача існує, і її вага є більшою за коректувальну здатність коду. Тобто, необхідно і надалі виконувати операції зсуву праворуч та ділення згідно з пунктом алгоритму 4. Результат четвертої та п’ятої ітерацій цього кроку алгоритму наведені на рис. 2.23. Видно, що на п’ятій ітерації вага остачі дорівнює коректувальній здатності коду, тому, згідно з описом алгоритму, процес зсуву та ділення на цьому кроці слід припинити і перейти

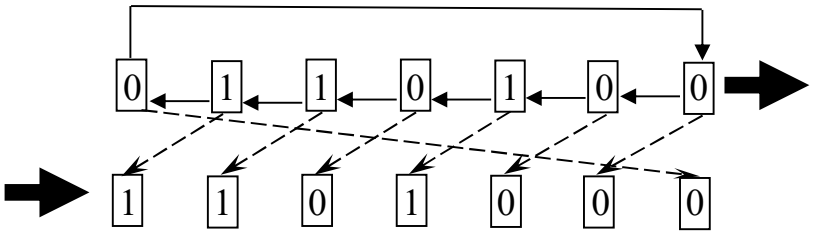
до виконання пункту 6, тобто додати остачу до отриманої кодової комбінації, використовуючи для цього операцію сумування за модулем 2. Виконання цієї операції дає такий результат: 1101000 ⊕ 0000001 = 1101001.

Четверта ітерація



$$\begin{array}{l} 0110100 \;\big|\; 1011 \\ \oplus\;\underline{1011} \;\big|\; \overline{111} \\ \oplus\;1100 \\ \quad\underline{1011} \qquad w > j \\ \oplus\;1110 \\ \quad\underline{1011} \\ \quad\;101 \end{array}$$

П’ята ітерація



$$\begin{array}{l} 1101000 \;\big|\; 1011 \\ \oplus\;\underline{1011} \;\big|\; \overline{1111} \\ \oplus\;1100 \\ \quad\underline{1011} \qquad w = j \\ \oplus\;1110 \\ \quad\underline{1011} \\ \oplus\;1010 \\ \quad\underline{1011} \\ \qquad\;1 \end{array}$$

Рис. 2.23 Четверта та п’ята ітерації виконання четвертого кроку алгоритму виправлення помилки у циклічному коді, блок-схема якого наведена на рис. 2.16

Тепер, згідно з пунктом 8 наведеного алгоритму, необхідно зробити циклічний зсув отриманого результату праворуч на стільки ж позицій, на скільки попередньо був здійснений зсув ліворуч, тобто, у нашому випадку, на

4 позиції. Цей ітераційний процес наочно показаний на рис. 2.24.

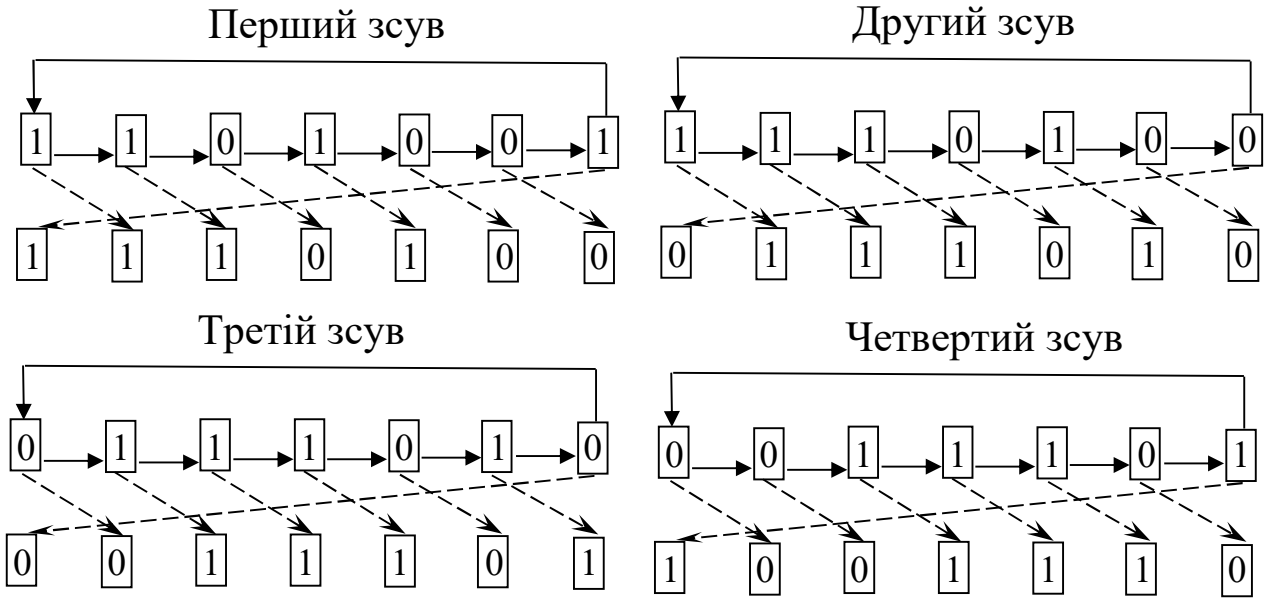


Рис. 2.24 Ілюстрація циклічного зсуву отриманої кодової комбінації 1101001 праворуч на 4 позиції

Тобто, в результаті послідовного виконання кроків наведеного вище алгоритму, помилка передавання даних виправлена і поновлена початкова кодова комбінація: 1001110.

2.5.4 Способи вибору незвідних поліномів залежно від коректувальної здатності циклічного коду

Зрозуміло, що у разі формування циклічного коду, який забезпечує виправлення одиночних помилок, можна користуватися співвідношенням (2.80), або, у разі невеликих значень *k*, таблицею 2.15. Розглянемо тепер спосіб визначення розрядності твірного поліному $Q(x)$ у загальному вигляді, для циклічних кодів різної коректувальної здатності [5, 52, 53, 55, 62, 63].

Почнемо з простого випадку формування коду із виявленням одиночної помилки. Спочатку запишемо співвідношення для кодової комбінації $S(x)$ з урахуванням можливості помилки у наступному вигляді:

$$H(x) = S(x) \oplus \xi(x), \tag{2.83}$$

де $S(x)$ – правильна кодова комбінація, $\xi(x)$ – вектор помилки.

У разі ділення поліному $H(x)$ на твірний поліном $Q(x)$ виникає остача, яка і є синдромом помилки. Тому помилка буде виявлена, якщо її вектор $\xi(x)$ не ділиться на $Q(x)$ без остачі. Вектор одиночної помилки $\xi(x)$ має значення 1 у спотвореному розряді кодової комбінації $H(x)$ та 0 в інших розрядах. Звідси можна зробити висновок, що вектору $\xi(x)$ відповідає поліном

$$\xi(x) = x^i \ , \ i = 1 \ldots n, \tag{2.84}$$

де i – номер помилкового розряду.

Для виявлення одиночної помилки поліном $\xi(x)$, заданий співвідношенням (2.84), повинен не ділитися на твірний поліном $Q(x)$. Серед незвідних поліномів, на які розкладається многочлен $x^n + 1$, ділення на який, згідно із теоремою 2.3, відповідає циклічному зсуву кодової комбінації праворуч, найпростішим є поліном першого порядку $x + 1$. Це означає, що остача від ділення спотвореної кодової комбінації на поліном $x + 1$ буде дорівнювати 0, якщо вона є правильною, або 1, якщо вона містить помилковий розряд. Зімкнене кільце такого циклічного коду містить ідеал, в який входять всі поліноми заданої розрядності із парною кількістю одиниць, та одного класу лишків, якому відповідає остача 1. Тобто, циклічний код із виявленням одиночних помилок – це простий код із перевіркою на парність, розглянутий у підрозділі 2.3.

Розглянемо приклад формування циклічного коду із виявленням однієї помилки.

Приклад 2.17. Сформувати циклічний код із виявленням однієї помилки для двійкового числа 1010.

Поліноміальна форма подання для числа 1010 була отримана у прикладі 2.16, це поліном $x^3 + x$. Циклічний код для числа 1010 отримуємо як результат множення цього поліному на твірний поліном $x + 1$:

$$(x^3 + x)\cdot(x + 1) = x^4 + x^3 + x^2 + x,$$

або у числовій формі 11110.

Тобто, кількість одиниць є парною. Перевіримо отриману кодову комбінацію шляхом її ділення на твірний поліном $x + 1$, який у числовій формі записується як число 11. Результат цього ділення показаний на рис. 2.25.

$$\begin{array}{r|l} 11110 & 11 \\ \hline \oplus\ 11 & 1010 \\ \hline \oplus\ 11 & \\ \hline 0 & \end{array}$$

Рис. 2.25 Перевірка коректності роботи циклічного коду для числа 1010 за умови відсутності помилки

Із результату ділення можна зробити висновок, що частка від ділення співпадає із початковим числом, а остача дорівнює нулю, що цілком відповідає принципу формування коду.

Тепер припустимо, що виникла помилка у третьому розряді числа, тобто, отримана спотворена кодова комбінація 11010 і кількість одиниць є непарною. Результат ділення цієї хибної кодової комбінації на твірний поліном $x + 1$ показаний на рис 2.26.

$$\begin{array}{r|l} 11010 & 11 \\ \hline \oplus\ 11 & 1001 \\ \hline \oplus\ 10 & \\ \hline 11 & \\ \hline 1 & \end{array}$$

Рис. 2.26 Перевірка коректності роботи циклічного коду для числа 1010 за умови наявності помилки у третьому розряді

Тобто, остача від ділення у цьому випадку дорівнює 1, що свідчить про наявність одиночної помилки у коді.

Слід також відзначити, що розглянутий циклічний код, який дозволяє виявляти одиночні помилки, виявляє також помилки непарної кратності, наприклад, потрійні. Проте помилки парної кратності, наприклад подвійної, таким кодом не виявляються [5].

Розглянемо тепер більш складний клас циклічних кодів, які дозволяють виправляти одиночні та виявляти подвійні помилки. Як і у випадку лінійних кодів, для того, щоб виправити помилку у кодовій комбінації циклічного коду, необхідно визначити, який із розрядів був спотворений, тобто, сформувати синдром помилки. Визначення синдрому помилки закладається безпосередньо у структурі коду, і це в значній мірі ускладнює спосіб його формування.

Оскільки алгоритм декодування циклічних кодів оснований на діленні кодової комбінації на твірний поліном і визначенні остачі від цього ділення, саме значення остачі і має бути синдромом помилки. А із цього випливає, що остачі від ділення всіх спотворених кодових комбінацій із одиночною помилкою на твірний поліном мають бути різними. Як було відмічено у попередньому підрозділі, саме тому для формування циклічних кодів використовують незвідні твірні поліноми, оскільки вони забезпечують найбільшу кількість остач. Для поліномів степені $m = n - k$ кількість остач, які не дорівнюють нулю, складає $2^{n-k} + 1$, а нульова остача, згідно із принципом формування циклічного коду, відповідає правильній кодовій комбінації.

Згідно з наведеними міркуваннями сформулюємо важливу властивість циклічних кодів, яка випливає із співвідношення (2.42) для границі Хеммінга [5, 52, 53, 55, 62, 63].

Властивість 2.9. Необхідною умовою виправлення будь-якої одиночної помилки у циклічному коді є виконання нерівності:

$$2^{n-k} - 1 \geq C_1^n = n, \qquad (2.85)$$

де C_1^n – кількість сполучень із n по 1, яке відповідає кількості можливих одиночних помилок та має значення n.

Із співвідношення (2.85) можна знайти простий явний вираз для степені твірного поліному $m = n - k$:

$$m \geq \log_2(n+1). \qquad (2.86)$$

Можливі значення n та k для значень m від 1 до 10 наведені у таблиці 2.16 [5].

Таблиця 2.16 – Залежність максимальної розрядності циклічного коду та кількості інформаційних розрядів від степені твірного поліному

m	1	2	3	4	5	6	7	8	9	10
n	1	3	7	15	31	63	127	255	511	1023
K	0	1	4	11	26	57	120	247	502	1013

Зрозуміло, що значення, наведені у таблиці 2.16, є правильними і для кодів Хеммінга із виправленням одиночної помилки. Наприклад, програма, наведена у додатку З, призначена для формування кодів (7, 4), (15, 11) та (31, 26).

Розглянемо без обґрунтування та доведення дві інші важливі властивості циклічних кодів [5, 52, 53, 55, 62, 63].

Властивість 2.10. Будь-який двочлен степені $x^{2m-1}+1$ може бути поданий як добуток незвідних поліномів, степені яких є дільниками числа m.

Властивість 2.11. Наслідок властивості 2.10. За умови $n > m$ для будь-якого значення m існує принаймні один незвідний поліном, який є дільником поліному $x^n + 1$.

Проте важливим завданням під час вибору незвідних поліномів є аналіз кількості можливих остач. Річ у тому, що класи лишків поліномів виду $x^n + 1$ іноді можуть перетинатися і тоді кількість остач для заданого значення n буде меншою за величину n, тобто, не виконується властивість 2.9.

Розглянемо відповідний приклад [5].

Приклад 2.18. Обрати твірний поліном для випадку $n = 15$ та $m = 4$.

Для розв’язування поставленої задачі необхідно з’ясувати, які незвідні поліноми порядку $m < 15$ є дільниками двочлену $x^{15} + 1$. Аналіз таких

поліномів дозволяє зробити висновок, що це поліноми першого, другого та четвертого порядку. Дійсно, згідно з таблицею твірних поліномів 2.14, а також із таблицею, наведеною у додатку І, можна записати:

$$(x+1)\cdot\left(x^2+x+1\right)\cdot\left(x^4+x+1\right)\cdot\left(x^4+x^3+1\right)\cdot\left(x^4+x^3+x^2+x+1\right)=$$
$$=\left(x^3+x^2+x+x^2+x+1\right)\cdot\left(x^4+x+1\right)\cdot\left(x^4+x^3+1\right)\cdot\left(x^4+x^3+x^2+x+1\right)=$$
$$=\left(x^3+1\right)\cdot\left(x^4+x+1\right)\cdot\left(x^4+x^3+1\right)\cdot\left(x^4+x^3+x^2+x+1\right)=$$
$$=\left(x^7+x^4+x^3+x^4+x+1\right)\cdot\left(x^4+x^3+1\right)\cdot\left(x^4+x^3+x^2+x+1\right)=$$
$$=\left(x^7+x^3+x+1\right)\cdot\left(x^4+x^3+1\right)\cdot\left(x^4+x^3+x^2+x+1\right)=$$
$$=\left(x^{11}+x^7+x^5+x^4+x^{10}+x^6+x^4+x^3+x^7+x^3+x+1\right)\times$$
$$\times\left(x^4+x^3+x^2+x+1\right)=\left(x^{11}+x^{10}+x^6+x^5+x+1\right)\times\left(x^4+x^3+x^2+x+1\right)=$$
$$=x^{15}+x^{14}+x^{10}+x^9+x^5+x^4+x^{14}+x^{13}+x^9+x^8+x^4+x^3+$$
$$+x^{13}+x^{12}+x^8+x^7+x^3+x^2+x^{12}+x^{11}+x^7+x^6+x^2+x+$$
$$+x^{11}+x^{10}+x^6+x^5+x+1=x^{15}+1.$$

Останній результат випливає із того, що всі степені змінної x в отриманому поліномі 15 степені, крім доданків x^{15} та 1, повторюються двічі.

Згідно із отриманим результатом будь-який незвідний поліном четвертої степені може бути взятим як твірний, але ще необхідно перевірити, чи не є він дільником поліномів меншого степеня. Наприклад, розглянемо поліном x^4+x^3+1, або, у числовій формі, 11001.

Синдроми перших чотирьох помилок, які можуть виникнути у першому, другому, третьому або четвертому розрядах, показані на рис. 2.27. Дійсно, оскільки номера розрядів є не більшими, ніж степінь твірного полінома $m = 4$, ознака помилки та її синдром у даному випадку співпадають.

Але починаючи з п’ятого розряду остача від ділення хибної кодової комбінації не співпадає із вектором помилки, тому під час формування коду кожну із таких остач потрібно аналізувати окремо і вони не повинні

повторюватись.

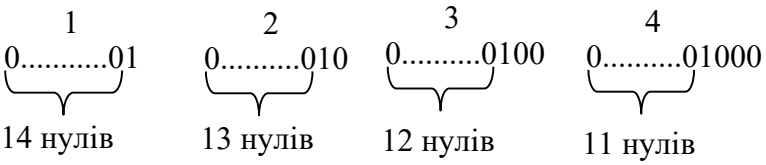


Рис. 2.27. Синдроми помилок у першому, другому, третьому та четвертому розрядах п’ятнадцятирозрядного циклічного коду

Наприклад, для кодової комбінації, яка відповідає помилці у п’ятому розряді, результат ділення наведений на рис. 2.28.

$$\begin{array}{l} \oplus\ 10000 \\ \underline{\ 11001\ } \\ \ \ 1001 \end{array} \Big| \begin{array}{l} 11001 \\ \hline 1 \end{array}$$

Рис. 2.28 Пошук синдрому помилки у п’ятому розряді п’ятнадцятирозрядного циклічного коду

Тобто, остача складає 1001 і не відповідає жодному із попередніх результатів.

Аналогічно можна отримати і інші остачі, проте є більш простий спосіб, який базується на наступній властивості циклічних кодів із виправленням одиночних помилок [5, 52, 53, 55, 62, 63].

Властивість 2.12. У разі ділення поліному x^n на незвідний поліном в процесі ділення, як проміжні, послідовно виникають всі остачі із класу лишків для цього незвідного поліному.

Згідно із властивістю 2.12, достатньо поділити число, перший розряд якого становить 1, а решта n – 1 розрядів – 0, на числову форму твірного поліному і виписати всі остачі, які виникають в процесі цього ділення. Відповідний процес покрокового ділення для поліномів x^{15} та $x^4 + x^3 + 1$ показаний на рис. 2.29.

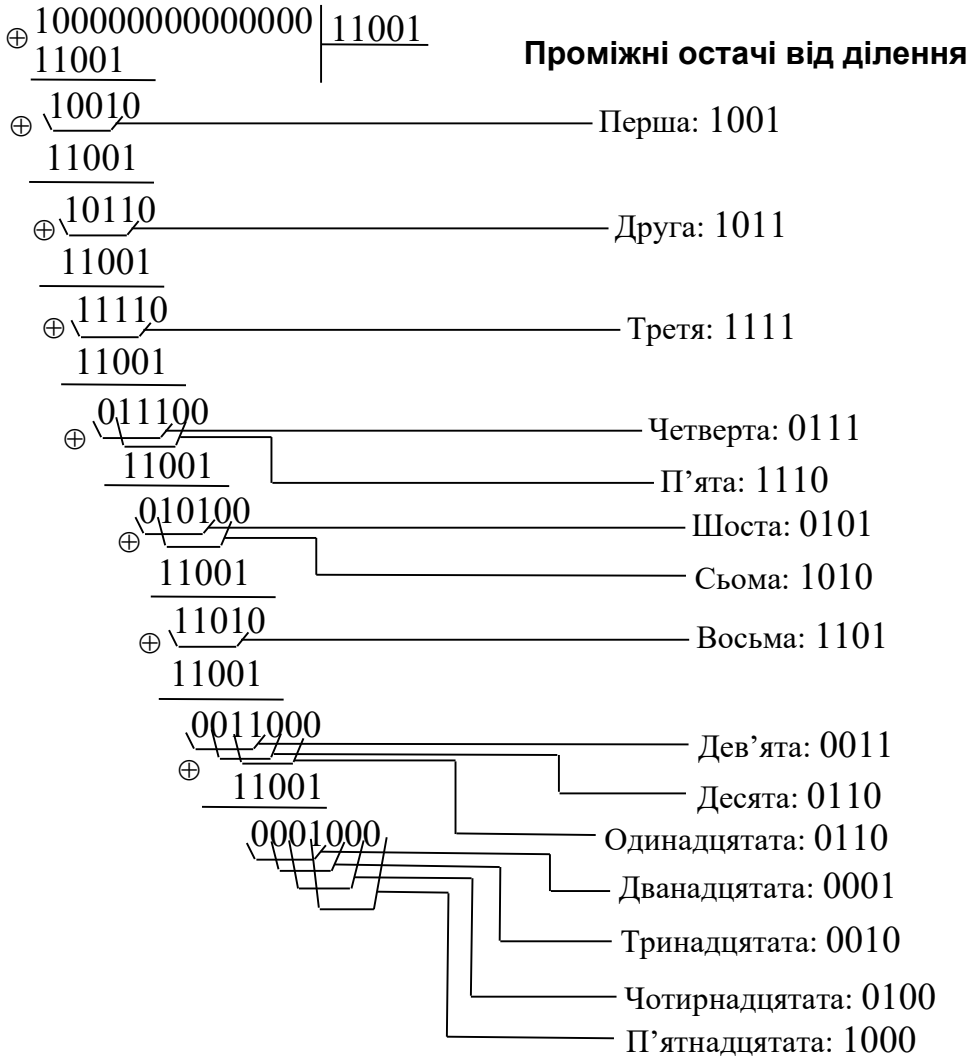


Рис. 2.29 Визначення проміжних остач від ділення поліному x^{15} на незвідний поліном $x^4 + x^3 + 1$

Як видно з рис. 2.29, особливість визначення проміжних остач під час ділення двійкових поліномів полягає в тому, що якщо результатом ділення є 0 і здійснюється перехід до наступного розряду числа, враховуються всі остачі. Наприклад, на останній, дев’ятій ітерації, після восьмого сумування остачі з дільником за модулем 2, спочатку отримуємо остачу 0001. Перехід до наступного, третього розряду діленого, дає остачу 0010, до другого розряду – остачу 0100, а до останнього, першого розряду – остачу 1000. З урахуванням цієї особливості ділення двійкових поліномів отримуємо 15

остач, що свідчить про правильність вибору твірного поліному $x^4 + x^3 + 1$.

Приклад 2.19. Для двійкового числа 101101110101 побудувати циклічний код, спосіб формування якого був розглянутий у прикладі 2.18. Перевірити коректність роботи коду для випадку наявності одиночної помилки у третьому розряді.

Запишемо число 101101110101 у поліноміальній формі, відповідний поліном буде мати вигляд:

$$P(x) = x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^2 + 1. \qquad (2.87)$$

Множення отриманого поліному $P(x)$ на твірний поліном

$$Q(x) = x^4 + x^3 + 1 \qquad (2.88)$$

дає наступний результат:

$$\begin{aligned} S(x) = P(x) \cdot Q(x) &= \left(x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^2 + 1\right) \cdot \left(x^4 + x^3 + 1\right) = \\ &= x^{15} + x^{13} + x^{12} + x^{10} + x^9 + x^8 + x^6 + x^4 + \\ &+ x^{14} + x^{12} + x^{11} + x^9 + x^8 + x^7 + x^5 + x^3 + \\ &+ x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^2 + 1 = \\ &= x^{15} + x^{14} + x^{13} + x^{10} + x^9 + x^8 + x^7 + x^3 + x^2 + 1. \end{aligned} \qquad (2.89)$$

Зрозуміло, що отриманий поліном $S(x)$ за визначених умов є циклічним кодом числа 101101110101. У числовій формі цей код записується як 1110011110001101. Розглянемо хибну кодову комбінацію

$$S_x = 1110011110001001. \qquad (2.90)$$

із помилкою у третьому розряді. Розділимо число 1110011110001001 на твірний поліном (2.89) та знайдемо остачу. Результат цього ділення показаний на рис. 2.30. Проаналізуємо отриманий результат згідно з алгоритмом визначення помилки у циклічному коді, наведеному на рис. 2.16. Вага остачі j у даному випадку дорівнює 1, тому циклічний зсув бітів числа у даному випадку не потрібний, а остача 100 вказує на помилку у третьому біті шістнадцятирозрядного коду. Додаючи за модулем 2 число 100 до хибної кодової комбінації S_x, заданої співвідношенням (2.90), отримуємо правильну

комбінацію $S(x)$, яка у поліноміальній формі задана співвідношенням (2.89).

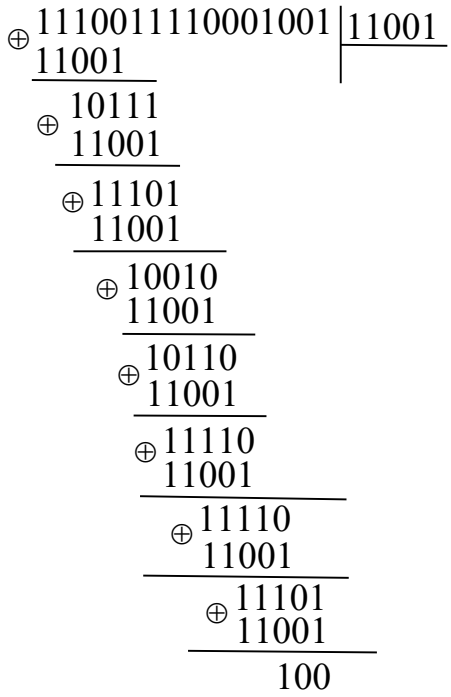


Рис. 2.30 Процес ділення хибної кодової комбінації (2.90) на твірний поліном (2.88) для прикладу 2.19

Циклічний зсув у даному прикладі не був потрібний з тієї причини, що помилка виникла у третьому розряді та номер помилкового розряду є нижчим за розрядність твірного поліному, і тому синдром помилки співпадає із остачею.

Розглянемо можливість формування п’ятнадцятирозрядного циклічного коду з використанням іншого твірного поліному четвертого порядку [5].

Приклад 2.20. Перевірити, чи можливо сформувати п’ятнадцятирозрядний циклічний код з використанням твірного поліному $x^4 + x^3 + x^2 + x + 1$.

Перевірку коректності використання поліному $x^4 + x^3 + x^2 + x + 1$ будемо проводити, як і в прикладі 2.18, шляхом ділення поліному x^{15} на цей поліном та визначення проміжних остач. Із рис. 2.31 видно, що для незвідного поліному $x^4 + x^3 + x^2 + x + 1$ існує не 15, а лише 5 остач, в які в процесі подальшого ділення починають циклічно повторюватись.

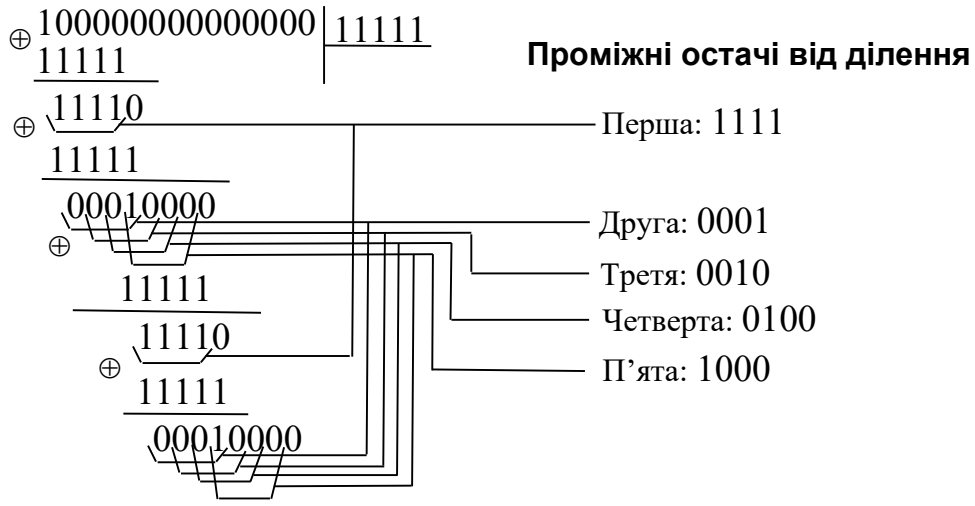


Рис. 2.31 Визначення проміжних остач від ділення поліному x^{15} на незвідний поліном $x^4 + x^3 + x^2 + x + 1$

Із отриманого результату можна зробити висновок, що незвідний поліном $x^4 + x^3 + x^2 + x + 1$ не може бути використаний як твірний для п’ятнадцятирозрядного циклічного коду. Це пояснюється тим, що поліном $x^4 + x^3 + x^2 + x + 1$ є дільником не лише поліному $x^{15} + 1$, але і поліному меншого порядку $x^5 + 1$. Дійсно:

$$\left(x^4 + x^3 + x^2 + x + 1\right) \cdot \left(x + 1\right) = x^5 + x^4 + x^3 + x^2 + x + x^4 + x^3 + x^2 + x + 1 = x^5 + 1.$$

Як висновок із розглянутих прикладів 2.18 та 2.20 можна навести відповідне визначення та сформулювати без доведення ще одну важливу властивість циклічних кодів [5, 52, 53, 55, 62, 63].

Визначення 2.12. Незвідний поліном $Q(x)$ вважається належним до степені n, якщо він є дільником поліному $x^n + 1$, але не є дільником жодного поліному $x^\lambda + 1$, якщо $\lambda < n$.

Властивість 2.13. Як твірні поліноми для формування циклічних кодів можна обирати лише належні поліноми.

Можлива кількість остач для деяких незвідних поліномів з першого до п’ятого порядку наведена у таблиці 2.17 [5]. Із наведених вище теоретичних міркувань зрозуміло, що кількість остач, які виникають під час ділення на твірний поліном, відповідає максимальній довжині циклічного коду.

Таблиця 2.17 – Кількість остач незвідних поліномів

№ з/п	Незвідні поліноми		Кількість остач
	Поліноміальне подання	Числове подання	
1.	$x^2 + x + 1$	111	3
2.	$x^3 + x + 1$	1011	7
3.	$x^3 + x^2 + 1$	1101	7
4.	$x^4 + x + 1$	10011	15
5.	$x^4 + x^3 + 1$	11001	15
6.	$x^5 + x^2 + 1$	100101	31
7.	$x^5 + x^3 + 1$	101001	31

Циклічні коди із виправленням одиночних помилок, аналогічні розглянутим у прикладах 2.16 та 2.19, також називаються кодами Хеммінга. На відміну від лінійних кодів Хеммінга, розглянутих у підрозділі 2.4, в систематичних циклічних кодах всі перевірочні розряди розташовуються в кінці кодової комбінації, але більшість класів циклічних кодів є несистематичними. В таких кодах інформаційні та контрольні розряди не розділені, і лише ділення кодового слова на твірний поліном дозволяє отримати закодовану інформацію. Саме такі коди були розглянуті у прикладах 2.16 та 2.19. Способи формування систематичних циклічних кодів, які найчастіше використовуються в сучасній обчислювальній техніці та в кодувальній електронній апаратурі, розглядатимуться у підрозділі 2.5.5.

Цікавим є те, що циклічні коди, сформовані на основі належних незвідних поліномів, які наведені у таблиці 2.14, також можуть бути використані для виявлення подвійних помилок. Перепишемо співвідношення (2.85) у вигляді:

$$H(x) = S(x) \oplus \xi(x), \qquad \xi(x) = x^i + x^j = x^i\left(x^{j-i} + 1\right), i < j. \qquad (2.91)$$

Із співвідношення (2.91) зрозуміло, що, оскільки $i - j < n$, а твірний

поліном $Q(x)$ не кратний x, цей поліном також не є дільником вектора помилки $\xi(x)$. Саме це і дозволяє виявляти подвійні помилки у кодовій комбінації як наявність остачі від ділення $S(x)/Q(x)$. У цьому полягає суттєва перевага циклічних кодів над лінійними кодами Хеммінга із виправленням одиночної помилки, які були розглянуті у підрозділі 2.4.1. Як було показано, за допомогою лінійних кодів, які дозволяють виправляти одиночні помилки, подвійні помилки не можуть бути виправлені. Здатність циклічних кодів, призначених для виправлення одиночних помилок, виявляти подвійні помилки обумовлена тим, що подвійна помилка у таких кодах не завжди приводить до переходу до іншої правильної кодової комбінації. Слід відзначити, що модифікований код Хеммінга також виявляє подвійні помилки, спосіб формування такого коду був описаний у підрозділі 2.4.2.

Розглянемо приклад такого використання циклічних кодів.

Приклад 2.21. Показати, що циклічний код 1110011110001101, який був отриманий у прикладі 2.18, може також бути використаний для виявлення подвійної помилки, вважаючи, що помилки виникли у третьому та п’ятому розрядах кодової комбінації.

Для випадку, який розглядається, хибна кодова комбінація має вигляд:

$$S_x = 1110011110011001. \qquad (2.92)$$

Результат ділення спотвореної кодової комбінації S_x на твірний поліном (2.88) показаний на рис. 2.32. Як видно із результатів ділення, остача дійсно існує.

$$
\begin{array}{l}
1110011110011001 \,\big|\, 11001 \\
\oplus\ 11001 \\
\hline
\quad 10111 \\
\quad \oplus\ 11001 \\
\quad \hline
\quad\ 11101 \\
\quad\ \oplus\ 11001 \\
\quad\ \hline
\quad\ \ 10010 \\
\quad\ \ \oplus\ 11001 \\
\quad\ \ \hline
\quad\ \ \ 10110 \\
\quad\ \ \ \oplus\ 11001 \\
\quad\ \ \ \hline
\quad\ \ \ \ 11111 \\
\quad\ \ \ \ \oplus\ 11001 \\
\quad\ \ \ \ \hline
\quad\ \ \ \ \ 11010 \\
\quad\ \ \ \ \ \oplus\ 11001 \\
\quad\ \ \ \ \ \hline
\quad\ \ \ \ \ \ 1101
\end{array}
$$

Рис. 2.32 Результат ділення спотвореної кодової комбінації (2.92) на твірний поліном (2.88)

Іншим цікавим прикладом циклічного коду є код із виправленням одиночних та виявленням подвійних помилок [67]. Твірний поліном для такого коду має порядок $m+1$ відносно порядку m належних незвідних поліномів, наведених у таблиці 2.17 та у додатку І, та записується як добуток такого поліному на поліном $x+1$, тобто [67]:

$$Q_{12}(x)=Q_1(x)\cdot Q_0(x)=Q_1(x)\cdot(x+1), \qquad (2.93)$$

де $Q_1(x)$ – твірний поліном для коду, який виправляє одиночні помилки, $Q_{12}(x)$ – твірний поліном для коду, який виправляє одиночні та виявляє подвійні помилки, $Q_0(x)$ – поліном першого порядку $x+1$.

Наприклад, для незвідного поліному $Q_1(x)=x^4+x^3+1$, який був використаний для формування циклічного коду у прикладі 2.19, можна записати:

$$Q_{12}(x)=\left(x^4+x^3+1\right)\cdot(x+1)=x^5+x^4+x+x^4+x^3+1=$$

$$=x^5+x^3+x+1.$$

Процедура декодування циклічних кодів із виправленням одиночних та виявленням подвійних помилок є досить простою [67]. Прийнята кодова комбінація спочатку ділиться на поліном $Q_1(x)$, а потім на поліном $Q_0(x)$. Після цього синдром помилки визначається за значеннями отриманих остач аналогічно до даних, наведених у таблиці 2.17 для контрольних сум коду Хеммінга. Якщо ділення кодової комбінації на обидві поліноми, $Q_1(x)$ та $Q_0(x)$, дає остачі, помилка виправляється, а якщо остача від ділення на поліном $Q_0(x)$ не існує, вважається, що виникла подвійна помилка і декодування хибної кодової комбінації не здійснюється [67]. Зрозуміло, що у випадку, коли обидві остачі дорівнюють нулю, помилка відсутня.

Розглянемо приклад формування циклічного коду із виправленням одиночних та виявленням подвійних помилок.

Приклад 2.22. Сформувати циклічний код із виправленням одиночної та виявленням подвійної помилки для двійкового числа 101101110101 з використанням твірного незвідного поліному $Q_1(x) = x^4 + x^3 + 1$. Перевірити коректність роботи отриманого коду за умови наявності одиночної помилки у третьому розряді та подвійної помилки у третьому та п'ятому розрядах кодової комбінації.

Для заданої кодової послідовності код із виправленням одиночних помилок був отриманий з використанням твірного поліному $Q_1(x)$ у прикладі 2.19 та задається співвідношенням (2.89). Тоді, згідно із співвідношенням (2.93):

$$S_2(x) = (x+1)\cdot\left(x^{15} + x^{14} + x^{13} + x^{10} + x^9 + x^8 + x^7 + x^3 + x^2 + 1\right) =$$
$$= x^{16} + x^{15} + x^{14} + x^{11} + x^{10} + x^9 + x^8 + x^4 + x^3 + x + x^{15} +$$
$$+ x^{14} + x^{13} + x^{10} + x^9 + x^8 + x^7 + x^3 + x^2 + 1 = x^{16} + x^{13} +$$
$$+ x^7 + x^2 + x + 1,$$

або у числовій формі:

$$S_2 = 10010000010000111. \qquad (2.94)$$

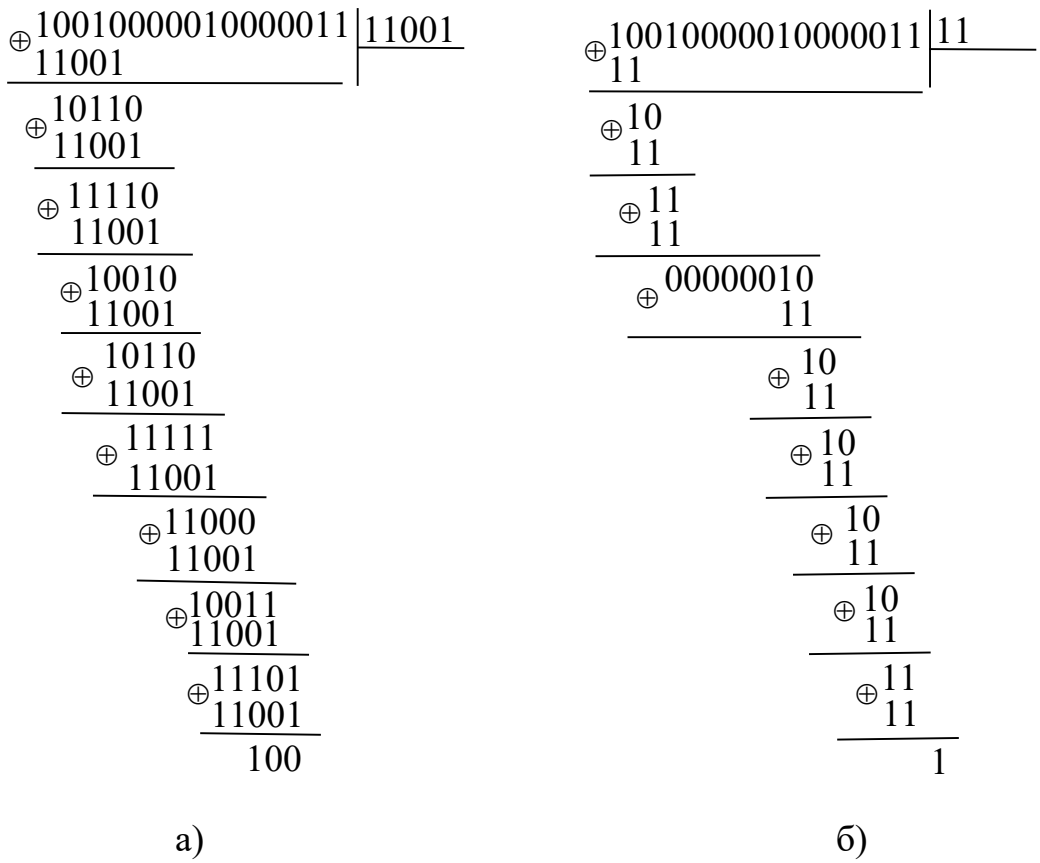
Із співвідношення (2.94) для заданої в умові прикладу одиночної помилки маємо:

$$S_{21x} = 10010000010000011, \qquad (2.95)$$

а для заданої подвійної помилки:

$$S_{22x} = 10010000010010011, \qquad (2.96)$$

Для кодової комбінації S_{21x}, заданої співвідношенням (2.95), результати ділення на поліном $Q_1(x)$ показані на рис. 2.33, а, а результати ділення на поліном $Q_0(x)$ – на рис. 2.30, б. Із наведених результатів видно, що остача від ділення на поліном $Q_1(x)$ дає синдром помилки у третьому розряді 100, а остача від ділення на поліном $Q_0(x)$ становить 1. Тобто, аналіз кодової комбінації через ділення на твірні поліноми $Q_0(x)$ та $Q_1(x)$ свідчить про наявність одиночної помилки у третьому розряді.



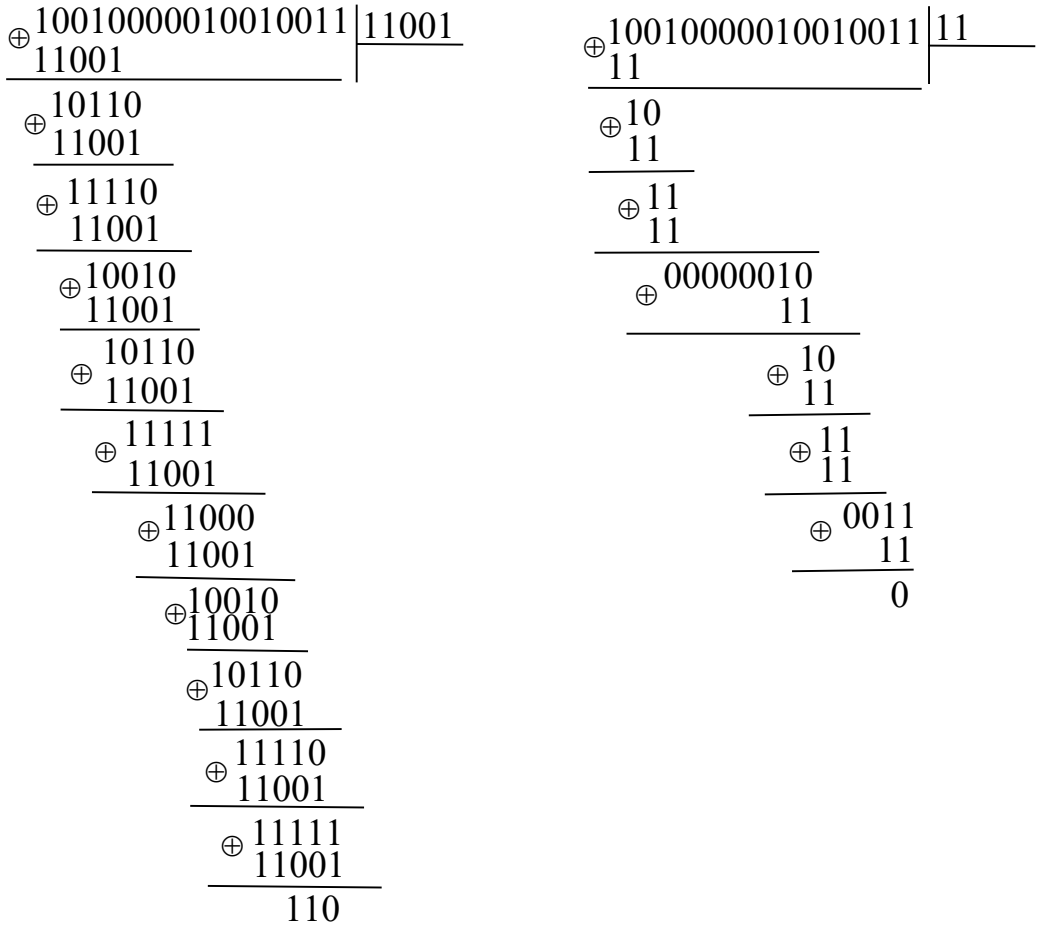
а) б)

Рис. 2.33 Результати ділення кодової комбінації (2.95) на твірні поліноми $Q_1(x)$ (а) та $Q_0(x)$ (б)

Для кодової комбінації S_{22x}, заданої співвідношенням (2.96), результати ділення на поліном $Q_1(x)$ показані на рис. 2.34, а, а результати ділення на поліном $Q_0(x)$ – на рис. 2.34, б. Для цього випадку остача від ділення на поліном $Q_0(x)$ дорівнює 0, що свідчить про подвійну помилку.

Як видно з рис. 2.34, а, вага остачі від ділення спотвореної кодової комбінації (2.96) на твірний поліном $Q_1(x)$ дорівнює 2, і у разі пошуку помилки, згідно із алгоритмом, наведеним на рис. 2.16, для подальшого пошуку необхідно зробити циклічний зсув кодової комбінації ліворуч. Проте, оскільки кодова комбінація S_{22x} ділиться на твірний поліном $Q_1(x)$ без остачі, що є ознакою подвійної помилки, для даного випадку подальший пошук помилкового розряду можна не продовжувати. Циклічні коди, у яких пошук синдрому помилки здійснюється через циклічний зсув ліворуч за

алгоритмом, наведеним на рис. 2.16, в літературі називаються також кодами Хаффмена [67]. Досконале описання процесу пошуку помилкового розряду у таких кодах та способу виправлення помилки були наведені у прикладі 2.15.



а) б)

Рис. 2.34 Результати ділення кодової комбінації (2.96) на твірні поліноми $Q_1(x)$ (а) та $Q_0(x)$ (б)

Покажемо, що циклічний код Хаффмена дійсно дозволяє виправляти помилки у кодових комбінаціях. Спочатку надамо відповідне визначення .

Визначення 2.13. Число послідовних зсувів ліворуч, після яких кодова комбінація повторюється, називається її періодом [67].

Наприклад, комбінація 10101010 має період 2, комбінація 111000111000 – період 4, комбінація 111111 – період 1, а комбінація 1110100 – період 7.

Визначення 2.14. Періодом циклічного коду $\tilde{\pi}$ називається найменший з періодів всіх його комбінацій [67].

Теорема 2.6. Через циклічний зсув кодової комбінації можна виправляти одиночні помилки [67].

Будемо вважати, що циклічний код породжується поліномом $Q(x)$ та його період дорівнює $\tilde{\pi}$. Тоді для синдрому помилки за умови її відсутності можна записати співвідношення:

$$Q(x)\left(x^{\tilde{\pi}}-1\right)=0.$$

звідки слідує, що

$$Q(x)\cdot\left(y^{\tilde{\pi}}-1\right)=e(x)\cdot\left(x^{n}-1\right). \tag{2.97}$$

За умовою (2.97) x^n-1 ділиться на твірний поліном $Q(x)$ без остачі. Тоді остаточно маємо:

$$x^{\tilde{\pi}}-1=e(x)\cdot G(x). \tag{2.98}$$

Оскільки за визначенням 2.14 число $\tilde{\pi}$ є найменшим періодом для всіх комбінацій циклічного коду, показник степені твірного поліному $Q(x)$ дорівнює $\tilde{\pi}$, а циклічний код, який створюється, має період $\tilde{\pi}$.

Цікавим є також те, що циклічні коди, завдяки своїй універсальності та ефективності, дозволяють виправляти пакети помилок. Для довільного блокового коду (n, k) зв'язок між кількістю контрольних символів та кількістю помилок b, які виправляються, встановлюється співвідношенням Рейджера:

$$n-k\geq 2b. \tag{2.99}$$

Якщо необхідно, щоб блоковий код виправляє пакети помилок довжиною b та одночасно виявляв помилки у пакетах довжиною $l > b$, тоді

мінімальна кількість контрольних символів складає $l + b$. Із циклічних кодів, які виявляють та виправляють пакетні помилки, найбільш відомими є коди Бартона, Файра та Ріда – Соломона. Коди Файра будуть досконало розглянуті у підрозділі 3.2, а коди Ріда – Соломона, які за своєю структурою є груповими, розглядатися у підрозділі 3.4.

2.5.5 Способи формування систематичних циклічних кодів

Існують різні способи формування циклічних кодів. Найпростішим з них, описаним у підрозділі 2.5.4, є множення та ділення двійкових послідовностей на твірний поліном. Суттєвий недолік такого способу полягає у тому, що отримані кодові комбінації не містять у явному вигляді інформаційних та контрольних розрядів, і лише ділення виправленої кодової комбінації на твірний поліном дозволяє отримати закодоване число. Такі коди, згідно із класифікацією, наведеною на рис 1.2, називаються несистематичними.

Іншим способом формування циклічного коду є розташування k інформаційних розрядів на старших позиціях кодової комбінації, а $r = n - k$ контрольних розрядів розташовуються на початку коду, який формується. Зрозуміло, що такий циклічний код, згідно із класифікацією, наведеною на рис. 1.2, відноситься до класу систематичних.

Систематичний циклічний код формується наступним чином [5]. Поліном $P(x)$, який відповідає кодовому слову, спочатку множиться на поліном x^m, де $m = n - k$, що відповідає приписуванню до початкового поліному m нулів, які розташовуються праворуч, тобто, у молодших розрядах числа. Після цього добуток $x^m \cdot P(x)$ ділиться на твірний поліном $Q(x)$, а остача від ділення $R(x)$ записується до молодших розрядів створеної кодової комбінації. Тоді

$$S(x) = x^m \cdot P(x) + R(x) = Q(x) \cdot P(x),$$

тобто, співвідношення (2.79), яке визначає спосіб формування циклічних кодів, виконується.

Такий спосіб побудови циклічних кодів є зручним, якщо кількість інформаційних розрядів є більшою, ніж контрольних [5].

Розглянемо приклад такого формування циклічного коду.

Приклад 2.23. Сформувати систематичний циклічний код із виявленням одиничної помилки для числа 1010 та перевірити коректність роботи коду за умови наявності помилки у п’ятому розряді кодової комбінації.

Як і у прикладі 2.16, використаємо як твірний поліном незвідний поліном $x^3 + x + 1$, або, у числовій формі, 1011. Тоді, згідно із описаним вище алгоритмом, код формується наступним чином.

1. $S_1(x) = 1010000 = x^7 + x^5$; $Q(x) = x^3 + x + 1 = 1011$.

2. Визначимо частку від ділення $S_1(x)/Q(x)$. Результат ділення показаний на рис. 2.35. $R(x) = 11 = x + 1$.

3. $S_2(x) = S_1(x) + R(x) = x^7 + x^5 + x + 1 = 1010011$.

4. Згідно із умовою задачі, заносимо помилку у п’ятий розряд коду. $S_{2x}(x) = 1000011$.

$$
\begin{array}{l|l}
\oplus\begin{array}{l}1010000\\1011\end{array} & 1011 \\
\hline
\quad\oplus\begin{array}{l}1000\\1011\end{array} & \\
\hline
\quad\ 11 &
\end{array}
$$

Рис. 2.35 Визначення остачі від ділення кодової комбінації 1010000 на твірний поліном $x^3 + x + 1$ для прикладу 2.23

5. Ділимо хибну кодову комбінацію $S_{2x}(x)$ на твірний поліном $Q(x)$. Згідно із алгоритмом, блок-схема якого наведена на рис. 2.16, якщо вага остача від ділення перевищує 1, необхідно здійснити циклічний зсув кодової комбінації ліворуч та повторювати ділення, доки вага остачі не стане рівною 1. Результати такого ділення показані на рис. 2.36, а – в.

Як видно із результатів ділення, на першій та другій ітерації вага остачі $w = 2$, тому, зважаючи на те, що коректувальна здатність коду $j = 1$, процес

ділення продовжується. На третій ітерації остача $R(x) = 100$, тобто її вага $w = 1$. Тому тепер, згідно із алгоритмом, описаним у підрозділі 2.5.2 та наведеним на рис. 2.16, необхідно виконати наступні дії.

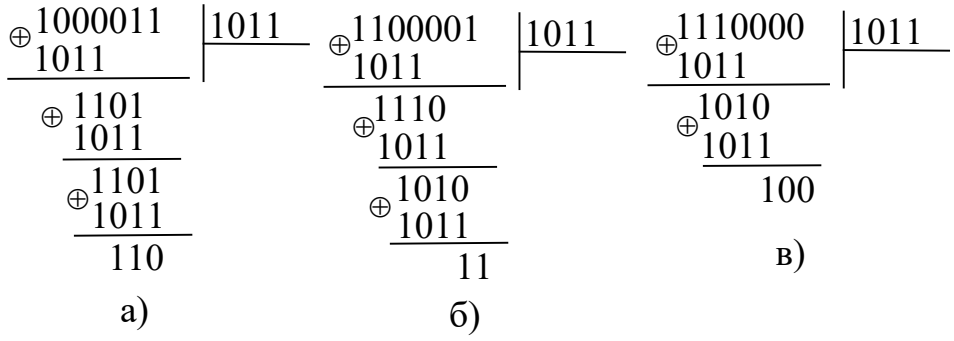


Рис. 2.36 Визначення остач від ділення спотвореної кодової комбінації 1000011 на твірний поліном $x^3 + x + 1$ для прикладу 2.23. Процес ділення із зсувом кодової комбінації праворуч здійснюється за алгоритмом, блок-схема якого наведена на рис. 2.16

1. Додати за модулем 2 отриману остачу до зсунутої ліворуч спотвореної кодової комбінації. В результаті отримуємо:

$1110000 \oplus 100 = 1110100$

2. Оскільки в процесі ділення спотворена кодова комбінація була циклічно зсунута на два розряди ліворуч, необхідно здійснити два циклічних зсуви отриманої послідовності бітів на 1 розряд праворуч. В результаті отримуємо:

після першого зсуву: 1101001;

після другого зсуву: 1010011.

Тобто, в процесі виконання всіх дій за алгоритмом, наведеним на рис. 2.16, виправлена помилка у систематичному циклічному коді.

Із розглянутого прикладу зрозуміла і перевага систематичних циклічних кодів. Після виправлення п’ятого розряду легко отримати закодоване число, достатньо відкинути перші три розряди коду. За такої умови відпадає необхідність здійснювати додаткове ділення правильної

кодової комбінації на твірний поліном, тобто, процес декодування значно спрощується.

Для систематичних циклічних кодів також існує інший алгоритм виправлення помилки, який через тезові формулювання можна записати наступним чином [67 – 74].

1. Знаходимо остачу $R_0(x)$ від ділення вектору помилки в останньому розряді $E(x) = [1,0,0,\ldots,0]$ розмірності n на твірний поліном $Q(x)$.

2. Знаходимо остачу $R_1(x)$ від ділення хибної кодової комбінації $S(x) = H_1(x)$ на твірний поліном $Q(x)$.

3. Порівнюємо значення $R_0(x)$ та $R_1(x)$.

4. За умови $R_0(x) = R_1(x)$ вважаємо, що помилка виникла в останньому розряді та переходимо до пункту 7.

5. За умови $R_0(x) \neq R_1(x)$ збільшуємо степінь поліному $H_1(x)$ на 1 та знаходимо остачу $R_l(x)$ від ділення $(H_n(x) \cdot x) / Q(x)$.

6. Якщо $R_0(x) \neq R_l(x)$, збільшуємо значення l на 1 та переходимо до пункту 5. Якщо $R_0(x) = R_l(x)$ – переходимо до пункту 7.

7. Вважаємо, що помилка виникла у розряді l та закінчуємо обчислення.

Блок-схема описаного вище алгоритму наведена на рис. 2.37.

Розглянемо приклад використання алгоритму, наведеного на рис. 2.37, для виправлення помилки у систематичному циклічному коді.

Приклад 2.24. Занести помилку у п’ятий розряд циклічного коду, сформованого у прикладі 2.23, та виправити її з використанням алгоритму, наведеного на рис. 2.37.

Згідно із результатами, отриманими у прикладі 2.23, систематичний циклічний код для числа 1010 становить $S_2(x) = 1010011$, а помилці у п’ятому розряді відповідає кодова комбінація $S_{2x}(x) = 1000011$.

Згідно із сформульованим алгоритмом декодування ітеративного коду для пошуку помилки слід виконати наступні кроки.

1. Поділити вектор помилки в останньому розряді $E(x) = [1,0,0,0,0,0,0]$ на твірний поліном $x^3 + x + 1$, або у числовій формі 1011. Результат цього ділення показаний на рис. 2.38, а, а остача від ділення $R_0(x)$ становить 101.

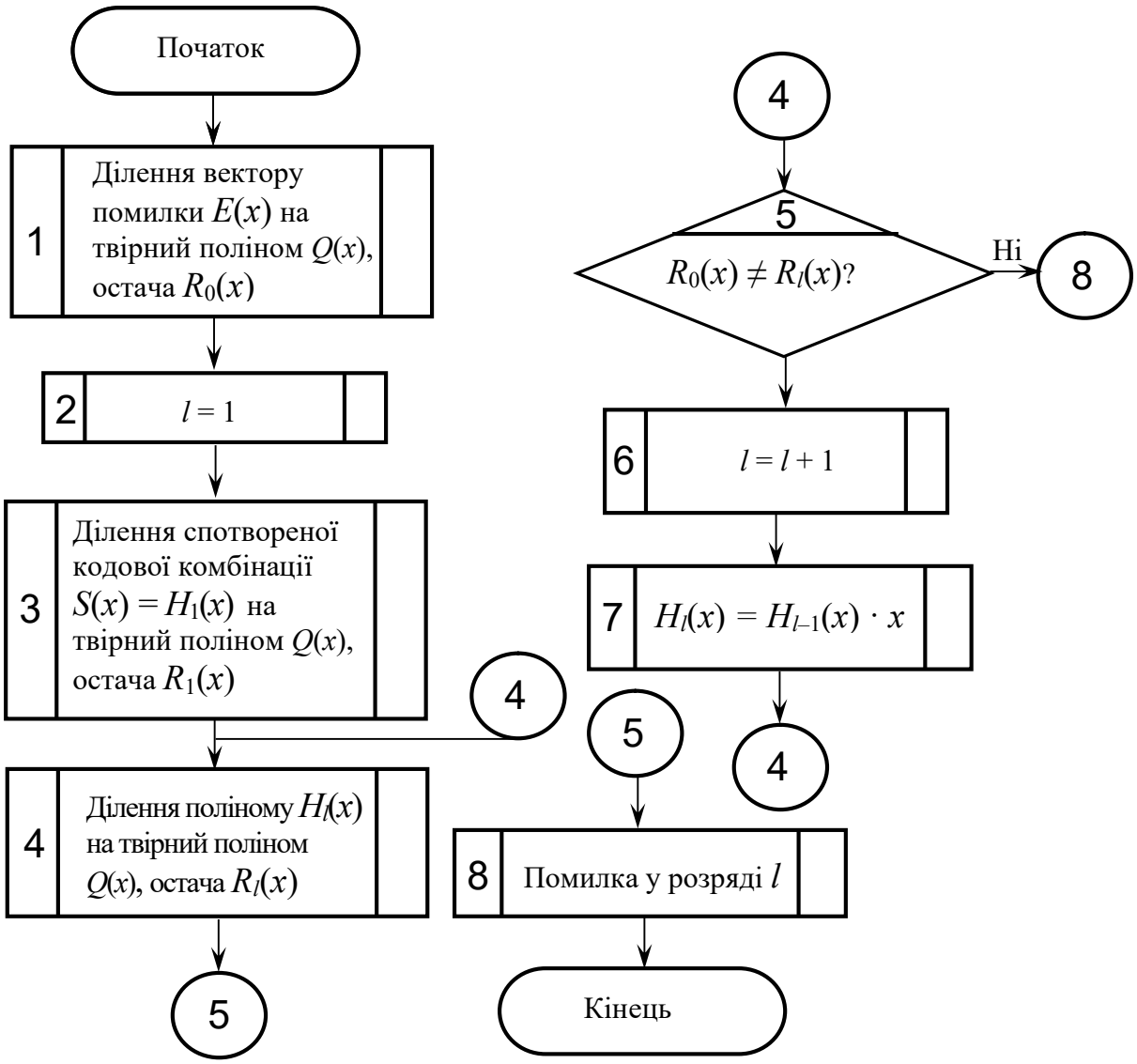


Рис. 2.37 Алгоритм пошуку помилки у систематичних лінійних кодах

$$\begin{array}{r|l} 1000000 & 1011 \\ \oplus\ 1011 & \\ \hline 1100 & \\ \oplus\ 1011 & \\ \hline 1110 & \\ \oplus\ 1011 & \\ \hline 101 & \end{array}$$

а)

$$\begin{array}{r|l} 1000011 & 1011 \\ \oplus\ 1011 & \\ \hline 1101 & \\ \oplus\ 1011 & \\ \hline 1101 & \\ \oplus\ 1011 & \\ \hline 110 & \end{array}$$

б)

$$\begin{array}{r|l} 10000110 & 1011 \\ \oplus\ 1011 & \\ \hline 1101 & \\ \oplus\ 1011 & \\ \hline 1101 & \\ \oplus\ 1011 & \\ \hline 1100 & \\ \oplus\ 1011 & \\ \hline 111 & \end{array}$$

в)

$$\begin{array}{r|l} 100001100 & 1011 \\ \oplus\ 1011 & \\ \hline 1101 & \\ \oplus\ 1011 & \\ \hline 1101 & \\ \oplus\ 1011 & \\ \hline 1100 & \\ \oplus\ 1011 & \\ \hline 1110 & \\ \oplus\ 1011 & \\ \hline 101 & \end{array}$$

г)

Рис. 2.38 Реалізація алгоритму, блок-схема якого наведена на рис. 2.37, для прикладу 2.24

2. Поділити спотворену кодову комбінацію $S_2(x) = H_1(x) = 1000011$ на 1011. Результат цього ділення показаний на рис. 2.38, б, а остача від ділення $R_1(x)$ становить 110, тобто, $R_1(x) \neq R_0(x)$.

3. Множимо спотворену кодову комбінацію на x і отримуємо новий поліном $H_2(x) = 10000110$. Результат ділення показаний на рис. 2.38, в. Остача від ділення $R_2(x)$ становить 111, тобто, $R_2(x) \neq R_0(x)$.

4. Множимо поліном $H_2(x)$ на x і отримуємо новий поліном $H_3(x) = 100001100$. Результат ділення показаний на рис. 2.38, г. Остача від ділення $R_3(x)$ становить 101, тобто, $R_3(x) = R_0(x)$. Процес ділення завершений. Помилка виявлена у третьому розряду від кінця закодованого числа, тобто, у п'ятому розряді, якщо рахувати їх у звичайному порядку, зправа-наліво.

Інший ефективний спосіб формування систематичного циклічного коду базується тому, що він розглядається як груповий. Тоді визначити контрольні розряди можна через розрахунок відповідних контрольних сум. Рівняння для визначення контрольних символів циклічного коду отримуються через наступні рекурентні співвідношення [5, 52, 53, 55 – 57, 62, 63, 67 – 74]:

$$a_{i+k} = \sum_{j=0}^{k-1} h_j a_{i+j}, \qquad i = 0,\ldots, k-1. \tag{2.100}$$

де $h(x)$ – генераторний поліном [5]:

$$h(x) = \frac{x^n + 1}{Q(x)} = h_0 + h_1 x + h_2 x^2 + \ldots + h_k x^k, \tag{2.101}$$

де $h_0 \ldots h_k$ – коефіцієнти генераторного поліному.

Після обчислення контрольних символів згідно із співвідношеннями (2.100 (2.101) вони розташовуються у молодших розрядах числа. Єдина особливість використання цих рівнянь полягає у тому, що у даному випадку використовується зворотна нумерація розрядів, зліва направо. В іншому два описаних алгоритми формування систематичного циклічного коду є еквівалентними і дають однаковий результат [5, 52, 53, 62, 63, 67].

Розглянемо приклад формування циклічного коду з використанням

генераторного поліному (2.101) та рекурентних співвідношень (2.100).

Приклад 2.25. Сформувати систематичний циклічний код із виявленням одиничної помилки для числа 1010 з використанням генераторного поліному (2.101). Порівняти отриманий результат із результатом, який був отриманий у прикладі 2.23.

Для прикладу, який розглядається, генераторний поліном $h(x)$ записується наступним чином:

$$h(x)=\frac{x^n+1}{Q(x)}=\frac{x^8}{x^3+x+1}=\frac{1000001}{1011}=1011.$$

Процес ділення поліномів, поданих у числовій формі, показаний на рис. 2.39.

$$\begin{array}{r|l} \oplus\,10000001 & 1011 \\ \underline{\;\;1011\;\;\;\;\;\;\;} & \underline{1011} \\ \oplus\,1100 & \\ \underline{\;\;1011\;\;} & \\ \oplus\,1110 & \\ \underline{\;\;1011\;\;} & \\ 1011 & \\ \underline{\;\;1011\;\;} & \\ 0 & \end{array}$$

Рис. 2.39 Пошук генераторного поліному для прикладу 2.25

Тобто, коефіцієнти генераторного поліному $h(x)$ мають такі значення: $h_0 = 1$; $h_1 = 0$; $h_2 = 1$; $h_3 = 1$.

Тепер для виконання обчислень з використанням системи рекурентних рівнянь (2.100) пронумеруємо розряди коду, а також інформаційні та контрольні розряди, із яких він складається, зправа-наліво. Відповідне розташування розрядів у структурі коду показано у таблиці 2.18.

Згідно із структурою коду, наведеною у таблиці 2.18, перепишемо рекурентні рівняння (2.100) наступним чином:

$$\begin{array}{ll} i=2, & a_{i+k}=a_6=h_0\cdot a_2\oplus h_1\cdot a_3\oplus h_2\cdot a_4\oplus h_3\cdot a_5; \\ i=1, & a_{i+k}=a_5=h_0\cdot a_1\oplus h_1\cdot a_2\oplus h_2\cdot a_3\oplus h_3\cdot a_4; \\ i=0, & a_{i+k}=a_4=h_0\cdot a_0\oplus h_1\cdot a_1\oplus h_2\cdot a_2\oplus h_3\cdot a_3. \end{array} \qquad (2.102)$$

Таблиця 2.18 – Структура систематичного сьомирозрядного циклічного коду

Номер розряду	a_6	a_5	a_4	a_3	a_2	a_1	a_0
Розташування інформаційних та контрольних розрядів	k_3	k_2	k_1	k_0	r_2	r_1	r_0
Значення інформаційних розрядів	1	0	1	0	–	–	–

Розв’язуючи систему рівнянь (2.102) з урахуванням визначених коефіцієнтів генераторного поліному h_0 – h_3, а також структури коду та значень інформаційних розрядів, наведених у таблиці 2.18, отримуємо наступні результати.

$$a_6 = h_0 \cdot a_2 \oplus h_1 \cdot a_3 \oplus h_2 \cdot a_4 \oplus h_3 \cdot a_5;\ \Rightarrow 1 = 1 \cdot r_2 \oplus 0 \cdot 0 \oplus 1 \cdot 1 \oplus 1 \cdot 0; \Rightarrow$$

$$\Rightarrow 1 = r_2 \oplus 1; \Rightarrow r_2 = 0;$$

$$a_5 = h_0 \cdot a_1 \oplus h_1 \cdot a_2 \oplus h_2 \cdot a_3 \oplus h_3 \cdot a_4;\ \Rightarrow 0 = 1 \cdot r_1 \oplus 0 \cdot r_2 \oplus 1 \cdot 0 \oplus 1 \cdot 1; \Rightarrow$$

$$\Rightarrow 0 = r_1 \oplus 1; \Rightarrow r_1 = 1;$$

$$a_4 = h_0 \cdot a_0 \oplus h_1 \cdot a_1 \oplus h_2 \cdot a_2 \oplus h_3 \cdot a_3;\ \Rightarrow 1 = 1 \cdot r_0 \oplus 0 \cdot r_1 \oplus 1 \cdot r_2 \oplus 1 \cdot 0 \Rightarrow$$

$$\Rightarrow 1 = 1 \cdot r_0 \oplus 0 \cdot 1 \oplus 1 \cdot 0 \oplus 0 = 1 \cdot r_0 \Rightarrow r_0 = 1.$$

Тобто значення контрольних розрядів, обчислені з використанням співвідношень (2.100), (2.101), співпадають із значеннями, отриманими у прикладі 2.23 шляхом ділення числа, яке кодується, на твірний поліном та визначення остачі.

Слід відзначити, що спосіб кодування із визначенням остачі є зручним та ефективним за умови невеликої кількості контрольних розрядів відносно інформаційних, тобто коли надлишковість коду за інформаційними

символами R_k, яка визначається із співвідношення (2.39), є меншою за 1. За умови $R_k > 1$ ефективнішим є обчислення значень контрольних розрядів коду з використанням співвідношень (2.100), (2.101). Можливість використання алгоритму кодування із визначенням остач у комп'ютерній програмі, призначеній для автоматичної побудови систематичних циклічних кодів, буде розглянута у підрозділі 2.5.8.

Слід відзначити, що більшість циклічних кодів, які використовуються на практиці, є систематичними, оскільки використання таких кодів значно спрощує алгоритм декодування. Схеми електронних пристроїв, які застосовуються для кодування та декодування циклічних кодів, будуть розглянуті у підрозділі 2.5.7.

Метод кодування із визначенням контрольних сум (2.100), (2.101) є досить універсальним та використовується також для побудови інших типів кодів, зокрема кодів Файра та Боуза – Чоудхаурі – Хоквінгема, які розглядатимуться у третьому розділі.

Крім розглянутих алгоритмів, для декодування послідовностей циклічного коду з використанням рекурентних співвідношень (2.100), (2.101), часто використовують принцип мажоритарного декодування, який був описаний у підрозділі 2.4.3. Особливості використання принципу мажоритарного декодування для циклічних кодів будуть описані у підрозділі 2.5.7. Велику кількість прикладів щодо побудови циклічних кодів за різними алгоритмами кодування та декодування можна знайти у навчальних посібниках та на сайтах Інтернет [52, 55 – 57, 62, 63, 67 – 74].

У наступному підрозділі розглядатиметься матрична форма подання циклічних кодів.

2.5.6 Матричне подання циклічних кодів

Перед вивченням цього підрозділу необхідно повторити четвертий розділ другої частини посібника та підрозділ 2.4.4 цієї частини посібника

Матричне подання циклічних кодів цілком відповідає матричному поданню для лінійних, яке було розглянуто у підрозділі 2.4.4. Базовий принцип матричного подання циклічного коду полягає у тому, що матриця

коду $\mathbf{M}_{<n,\,k>}$ має дві складові: одиничну матрицю $\mathbf{E}_{<k,\,k>}$ та матрицю доповнення $\mathbf{C}_{<k,\,n-k>}$, які з'єднуються за рядками тобто:

$$\mathbf{M}_{<n,\,k>} = \|\mathbf{E}_{<k,\,k>}, \mathbf{C}_{<k,\,n-k>}\|. \qquad (2.103)$$

Існують різні способи формування матриці доповнення. У разі, якщо для побудови систематичного циклічного коду використовуються рекурентні співвідношення (2.100), (2.101), можна формувати матрицю доповнення з застосуванням методів, які використовуються для матриць лінійних кодів та були описані у підрозділі 2.4.4, а також у підрозділі 5.4 другої частини посібника [48]. Проте якщо циклічний код формується через визначення остач від ділення поліномів, існують інші алгоритми отримання додаткової матриці. Розглянемо їх окремо.

Зазвичай за таких умов рядки одиничної матриці $\mathbf{E}_{<k,\,k>}$ розглядаються як вектори помилок. Тоді рядки матриця доповнення $\mathbf{C}_{<k,\,n-k>}$ можна формувати як синдроми помилок для визначеного вектора шляхом ділення рядків синдрому помилки на твірний поліном $Q(x)$ та пошуку остач $R(x)$. Згідно із принципами формування циклічних кодів, розглянутими у підрозділі 2.5.4, кожному синдрому помилки буде відповідати своя остача. Наприклад, для коду (15, 11) із породжувальним поліномом $Q(x) = x^4 + x^2 + 1$, розглянутого у прикладі 2.18, твірна матриця має вигляд, наведений на рис. 2.40.

Твірна матриця

$$\mathbf{M}_{\langle 15,11\rangle} = \left[\begin{array}{ccccccccccc|cccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0
\end{array}\right]$$

Одинична матриця

Матриця доповнення

Рис. 2.40 Вигляд твірної матриці для коду (15, 11), розглянутому у прикладі 2.18

Іншою модифікацією циклічних кодів, пов’язаних із їхнім матричним поданням, є скорочені циклічні коди [5]. Такі коди є вкрай ефективними з практичної точки зору, оскільки за умови їхнього використання зменшується кількість інформації, яку необхідно передати через канал зв’язку або зберігати у буфері пам’яті, що дає можливість спростити приймально-передавальне обладнання та зменшити його вартість.

Згідно із теоретичними відомостями, наведеними у підрозділі 2.5.4, головна проблема формування циклічних кодів полягає у виборі твірного поліному відповідної степені. Дійсно, не зважаючи на те, що кількість інформаційних розрядів може бути будь-яким натуральним числом, можливості вибори степені поліному, якою визначається розрядність коду, є вкрай обмеженими.

Таку надлишковість циклічних кодів можна пояснити на простому прикладі. У разі, якщо кількість інформаційних розрядів $k = 4$ можна використати незвідний твірний поліном третьої степені, і тоді $n = 7$. Оскільки кількість остач для поліному третьої степені також становить 7, можна сказати, що в даному випадку коректувальні можливості коду використовуються повністю. Але у випадку, коли кількість інформаційних розрядів становить $k = 5$, виникає дещо інша ситуація. У цьому випадку ми не можемо обрати многочлен третьої степені, оскільки він дає лише 7 остач, а кількість розрядів у коді $n = 8$. Тому слід обирати поліном четвертої степені, але він має не 9 остач, згідно із кількістю розрядів у кодовій комбінації, а 16, тобто, 7 остач є зайвими.

Можна легко уникнути цієї ситуації, якщо скоротити набір дозволених кодових комбінацій. Дійсно, якщо у коді не існує десятого розряду, не потрібний і синдром помилки у ньому, і відповідний рядок із твірної матриці, наведеної на рис. 2.40, можна викреслити. Теж саме можна сказати про рядки з 11 по 15. Разом із рядками викреслюються і зайві стовпчики одиничної

матриці, одиниці в яких відповідають векторам помилки у розрядах з десятого по шістнадцятий. Таким чином формується твірна матриця скороченого циклічного коду ($n - j$, $k - j$), де j – різниця між мінімально можливою кількістю розрядів для заданої кількості інформаційних символів та максимальною кількістю розрядів для заданого твірного поліному. Для прикладу, який розглядається, $j = 15 - 9 = 6$.

Наприклад, твірна матриця для коду (9, 5) створюється із матриці для коду (15, 11), наведеної на рис. 2.40, через викреслення в ній останніх шести рядків та перших шести стовпчиків. Така матриця має наступний вигляд:

$$\mathbf{M}_{\langle 9,5\rangle} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Надамо відповідне визначення та розглянемо властивості скорочених циклічних кодів.

Визначення 2.15. Скороченим циклічним кодом (англійський термін – shortened cyclic code) називається оптимальний код заданої коректувальної здатності, сформований із дозволених кодових комбінацій циклічного коду для відповідної степені твірного поліному через видалення зайвих рядків та стовпчиків твірної матриці.

Властивість 2.14. Мінімальна кодова відстань скороченого коду є не меншою, ніж для циклічного коду, із якого він створений.

Властивість 2.15. Результатом циклічного зсуву кодової комбінації скороченого коду не завжди є правильна кодова комбінація цього коду.

Тобто, для скорочених циклічних кодів не завжди виконується властивість 2.7.

Слід відзначити, що надлишкові циклічні коди формується лише у разі їх отримання через множення вхідного слова на надлишкову твірну матрицю. Якщо циклічний код формується з використанням алгоритмів, наведених на рис. 2.16 та 2.37, через множення та ділення поліномів та визначення остач, такий код завжди є оптимальним та щільноупакованим.

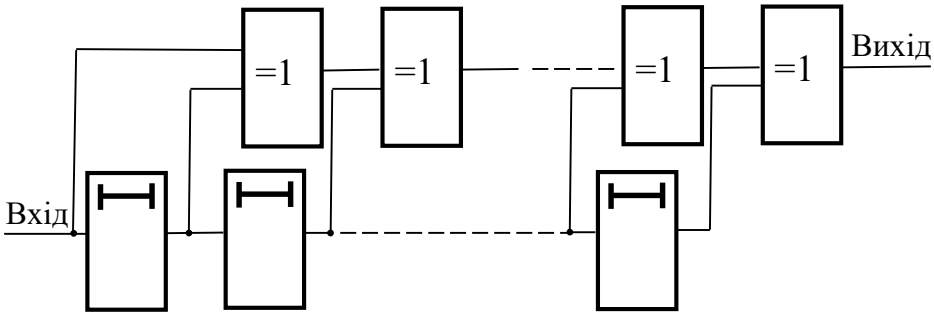
2.5.7 Схеми цифрових електронних пристроїв для формування циклічних кодів та їхнього декодування

Розглянемо окремо кодувальні схеми для формування несистематичних та систематичних циклічних кодів.

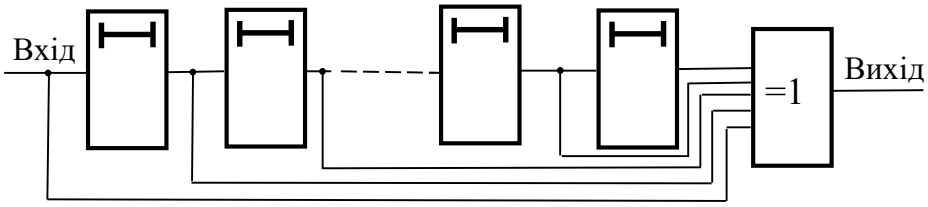
Цифрові електронні схеми для формування циклічних кодів базуються на виконанні множення та ділення двійкових поліномів через операцію сумування їх коефіцієнтів за модулем два. Зазвичай такі електронні схеми реалізуються на регістрах зсуву із зворотними зв'язками та із елементами пам'яті. Затримка на відповідну кількість тактів дозволяє сумувати різні біти чисел, які обробляються. Такі алгоритми множення та ділення двійкових чисел є аналогічними множенню та ділення в стовпчик і є відомим із основ комп'ютерної арифметики [31 – 37, 39, 40, 52, 53]. У технічній літературі цифрові схеми, основані на регістрах зсуву із зворотними зв'язками та елементами пам'яті, називаються лінійними схемами перемикання, або лінійними кодовими фільтрами Хаффмена [5, 33, 52, 56, 57]. Слід відзначити, що для двійкової арифметики додатковий пристрій множення або ділення не потрібний. Якщо відповідний поліноміальний коефіцієнт дорівнює 0, це відповідає відсутності зв'язку у схемі множення, а якщо 1 – його наявності. Зазвичай лінійні схеми перемикання є синхронними, і зсув інформації у регістрі здійснюється з приходом імпульсів з тактового генератора, проте цей елемент на схемах кодувальних та декодувальних пристроїв найчастіше не показується [5, 33, 52]. Також слід відзначити, що у більшості випадків кодувальні та декодувальні електронні схеми орієнтовані на один твірний поліном, і він визначається наявністю та відсутністю електричних зв'язків між лініями затримки. Тому на вхід кодувальних пристроїв надходить лише інформаційне слово, а на вхід декодувальних пристроїв – кодове слово. Важливим є також порядок слідування розрядів у коді. Існують схеми, орієнтовані на початкове надходження старших бітів числа, а для інших схем вважається, що спочатку надходять молодші біти послідовностей, які

кодуються або декодуються [5, 53].

Наприклад, схема множення із початковим надходженням старшого розряду наведена на рис. 2.41, а, а з початковим надходженням молодшого розряду числа – на рис. 2.41, б [5, 33, 52]. Ці схеми є узагальненими, кількість елементів пам’яті та ліній зв’язку в них залежить від розрядності вхідного слова та степені твірного полінома. Зрозуміло, що наявність або відсутність електричних зв’язків у таких схемах залежить від коефіцієнтів твірного полінома [5, 33, 52].



а)



б)

Рис. 2.41 Узагальнена схема поліноміального множення для зворотної (а) та прямої (б) послідовності надходження вхідних бітів числа

Зрозуміло, що з урахуванням затримки надходження бітів, коефіцієнти вихідного поліному формуються послідовно. Наприклад, розглянемо особливості роботи схеми, наведеної на рис. 2.41, а. Припустимо, що у початковий момент часу на всіх входах суматорів знаходяться нулі. Із надходженням першого вхідного біту, яким є коефіцієнт a_{k-1} поліному $a(x)$, на виході формується сигнал, рівний добутку $a_{k-1} \cdot g_{n-k}$. Цей сигнал дорівнює a_{k-1}, якщо $g_{n-k} = 1$ і існує зв’язок із входу схеми на вхід першого суматора,

або він завжди дорівнює 0 і не формується, якщо $g_{n-k} = 0$ і зв'язок із входу схеми на вхід першого суматора відсутній. На другому такті, у разі наявності першого та другого зв'язків, на вихід схеми з першого суматора надходить сума $a_{k-2} \cdot g_{n-k} \oplus a_{k-1} \cdot g_{n-k-1}$. На останньому такті на входах всіх суматорів, крім останнього, знаходяться нулі, і, таким чином, отримуємо останній результат $a_0 \cdot g_0$. Аналогічно працює схема, наведена на рис. 2.41, б. Наприклад, схема множення на незвідний поліном $x^3 + x^2 + 1$ для прямого порядку слідування бітів числа наведена на рис. 2.42.

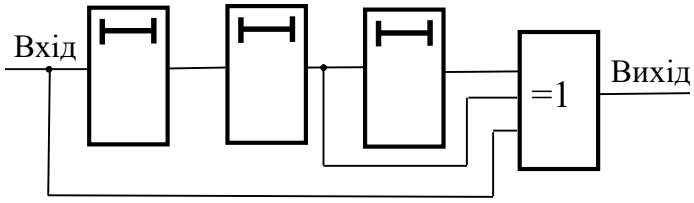


Рис. 2.42 Схема множення на твірний поліном $x^3 + x^2 + 1$ для прямого порядку слідування бітів числа

Узагальнена схема ділення довільної вхідної послідовності на твірний поліном наведена на рис. 2.43 [5].

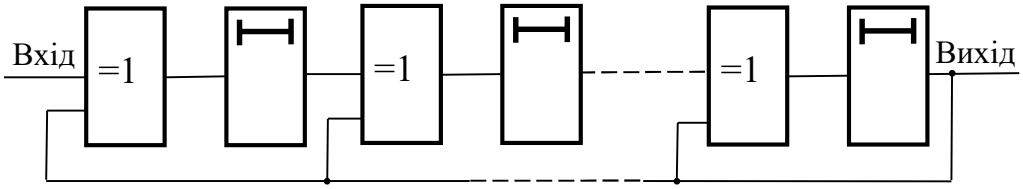


Рис. 2.43 Узагальнена схема поліноміального ділення

Особливістю схеми, наведеної на рис. 2.40 є те, що зворотні зв'язки з виходу на вхід схеми відповідають коефіцієнтам твірного полінома, які не дорівнюють 0, а кількість суматорів дорівнює $f - 1$, де f – кількість ненульових коефіцієнтів твірного поліному. Це пов'язано з тим, що старші розряди діленого та дільника сумувати не треба, оскільки результат такого сумування завжди дорівнює 0.

Схема ділення, наведена на рис. 2.43, працює наступним чином [5, 31,

52, 53]. За перші $n - k$ тактів коефіцієнти вхідного слова послідовно заповнюють регістри пам'яті так, що коефіцієнт, який відповідає старшій степені, розташовується у вихідній комірці. На наступному такті одиниця дільника, яка входить до старшого розряду, передається через ланцюжок зворотного зв'язку всім суматорам, що відповідає відніманню дільника від діленого. Якщо в результаті попередньої операції коефіцієнт біля старшої степені остачі був рівним 0, на наступному такті дільник віднімати не треба. У цьому разі необхідно просто зсунути дільник у напрямку від виходу до входу на 1 розряд. Такий алгоритм роботи схеми ділення цілком відповідає процесу ділення двійкових чисел у стовпчик, який неодноразово розглядався у цьому підрозділі на прикладах. Процес ділення завершується, коли на вхід схеми приходить останній символ дільника. Остаточна різниця двох поліномів має меншу степінь, ніж дільник, і вона є остачею.

Аналогічний алгоритм ділення двійкових чисел використовується у програмних засобах, призначених для формування циклічних кодів та декодування їхніх послідовностей. Такий рекурентний алгоритм, оснований на використанні матричних макроопераций системи науково-технічних розрахунків MatLab, буде розглянутий у підрозділі 2.5.8.

Приклад цифрової схеми ділення на незвідний поліном $x^3 + x^2 + 1$ наведений на рис. 2.44 [5, 33].

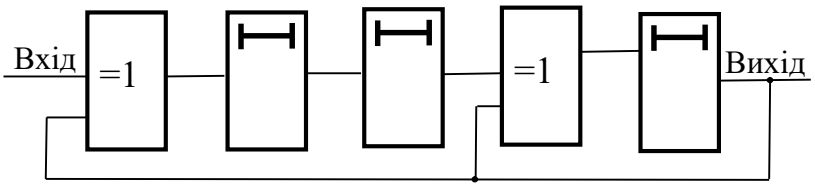


Рис. 2.44 Схема ділення вхідної послідовності на твірний поліном $x^3 + x^2 + 1$

Проте слід відзначити, що цифрові схеми, призначені для формування та декодування несистематичних циклічних кодів, основані на алгоритмах множення та ділення поліномів та наведені у загальному вигляді на рис. 2.41 та рис. 2.43, рідко використовуються у сучасній кодувальній електронній апаратурі [5, 33, 56, 57]. Це пов'язано з тим, що несистематичні коди не містять у явному вигляді інформаційних розрядів числа, що вкрай ускладнює процес декодування. Дійсно, як видно із прикладу, наведеному у підрозділі 2.5.3, пошук закодованого числа можливий лише через ділення кодової

комбінації на твірний поліном та визначення частки. Більш простими для декодування є систематичні циклічні коди, спосіб формування яких був описаний у підрозділі 2.5.5. Оскільки останні розряди систематичного циклічного коду цілком відповідають закодованому числу, процедура декодування спрощується до читання цих розрядів після виправлення виявлених помилок.

Розглянемо окремо різноманітні електронні пристрої, призначені для формування та декодування систематичних циклічних кодів, які використовуються у сучасній кодувальній електронній апаратурі [5, 33, 52, 53, 62, 63, 67]. Всі відомі кодувальні пристрої, як і розглянуті вище пристрої для формування несистематичних кодів, будуються на основі елементів пам'яті та поділяються на два класи. Схеми першого типу базуються на реалізації процесу ділення поліному $a(x)\cdot x^m$ на твірний поліном $q(x)$ згідно з алгоритмом, описаним у підрозділі 2.5.5 та наведеним на рис. 2.37, а схеми другого типу – на використанні рекурентних співвідношень (2.100), (2.101).

Схема кодувального пристрою, робота якого основана на застосуванні алгоритму ділення та визначені остач, наведена на рис. 2.45 [5, 33, 52]. Вона відрізняється від схеми, наведеної на рис. 2.43, тим, що коефіцієнти поліному, який кодується, підключаються до схеми зворотного зв'язку відразу, з першого такту, що дозволяє усунути розрив між інформаційними та контрольними символами. У початковому стані ключ K_1 знаходиться у положенні 1, і тоді інформаційні символи одночасно надходять як до лінії зв'язку, так і до регістру зсуву, де за k тактів створюється остача. На другому етапі роботи схеми електронний ключ K_1 переходить до положення 2 і остача від ділення також надходить до лінії зв'язку.

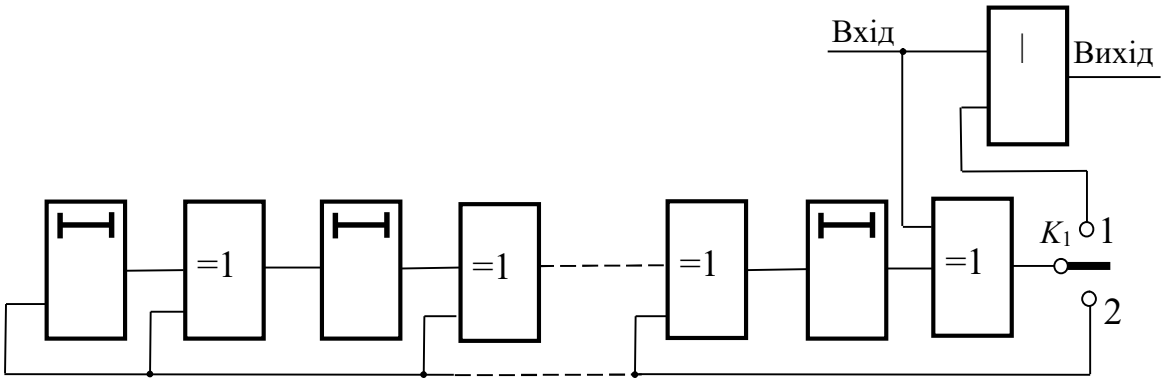


Рис. 2.45 Узагальнена схема формування циклічного коду через ділення на твірний поліном та взяття остачі

Узагальнена кодувальна схема другого типу наведена на рис. 2.46 [5, 33, 57, 58]. Робота цієї схеми основана на використанні рекурентних співвідношень (2.100), (2.101). Цей кодувальний пристрій базується на регістрі зсуву, який має розрядність k, а виходи комірок пам'яті підключаються до схем сумування за модулем 2 відповідно із коефіцієнтами генераторного поліному (2.101). Спочатку, протягом перших k тактів, перемикач K_1 знаходиться у положенні 1, і на протязі цього циклу інформаційні символи, які надходять на вхід схеми, заповнюють всі комірки регістра. Після цього електронний ключ K_1 переходить до положення 2. Другий цикл триває $n - k$ тактів. На протязі цього циклу один із інформаційних символів надходить із комірки пам'яті до каналу зв'язку та одночасно формується контрольний символ, який записується до останньої комірки регістра. Через $n - k$ тактів процес формування контрольних символів завершується та ключ K_1 знову повертається до положення 1. Протягом наступних k тактів вміст регістра передається до каналу зв'язку, але в той же час комірки регістра заповнюються значеннями нових інформаційних символів [5]. Тобто, процес декодування здійснюється неперервно.

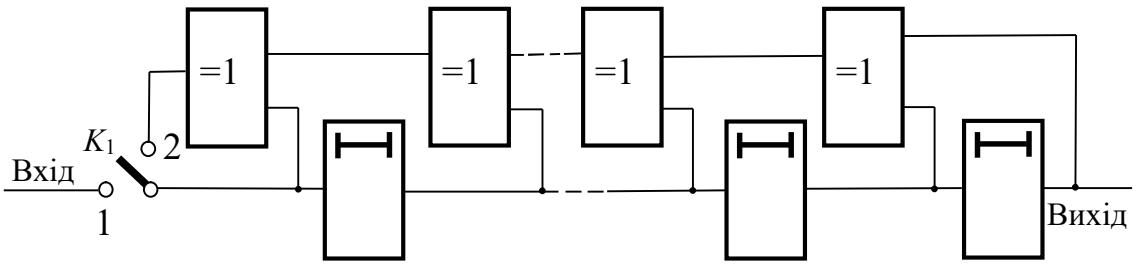


Рис. 2.46 Узагальнена схема формування циклічного коду через рекурентні співвідношення (2.100), (2.101)

Апаратне декодування циклічних кодів реалізується з використанням для роботи декодувальних електронних схем різних обчислювальних алгоритмів, які були описані у попередніх підрозділах. Загалом для виконання операції декодування використовують наступні обчислювальні алгоритми [5, 52, 62, 63].

1. Обчислення остач від ділення із циклічним зсувом кодової комбінації. Відповідний алгоритм наведений на рис. 2.16.

2. Обчислення остач від ділення із збільшенням розрядності кодового слова через множення на x, що відповідає дописуванню нулів у молодші розряди кодової комбінації. Відповідний алгоритм наведений на рис. 2.37.

3. Використання рекурентних співвідношень (2.100), (2.101).

4. Використання алгоритмів мажоритарного декодування, які у загальному вигляді були описані у підрозділі 2.4.3.

Кожний із цих методів має свої переваги та недоліки. Перший та другий методи є найбільш простими з точки зору розуміння алгоритмів роботи електронних схем, і в той же час вони характеризуються низьким відсотком невиявлених помилок заданої кратності. Зрозуміло, що перший метод частіше використовується для несистематичних, а другий – для систематичних кодів. Недоліком першого методу є необхідність виконання додаткових операцій зсуву бітів на кожній ітерації, а недоліком другого – необхідність додаткових розрядів для записування нулів або ускладнення обчислювальних електронних схем. Крім цього, загальним недоліком цих методів є відносна складність реалізації операції ділення на апаратному рівні [31, 35 – 40] та відносно велика кількість таких операцій. Третій метод є простішим з обчислювальної точки зору, але він потребує наявності великої кількості зворотних зв’язків між регістрами, що у значній мірі ускладнює реалізацію таких декодувальних схем. Як було відмічено у попередньому підрозділі, такий метод кодування та декодування найчастіше використовується для кодів із високою коректувальною здатністю, коли кількість контрольних символів перевищує кількість інформаційних. Метод мажоритарного кодування також може бути використаний лише для систематичних циклічних кодів. З обчислювальної точки зору він є найбільш простим, що спрощує схемну реалізацію таких декодерів. Це пов’язано з тим, що алгоритм мажоритарного декодування оснований на виконанні операції порівняння, яка для обчислювальних електронних пристроїв є однією із найпростіших та вважається елементарною [31, 35 – 40]. Єдиним недоліком

таких декодерів є можливість виникнення невиявлених помилок та відносно висока їх ймовірність [5, 33].

Розглянемо спочатку найбільш прості з точки зору розуміння принципу роботи декодувальні електронні пристрої, в яких для виявлення та виправлення помилок використовується ділення довільної вхідної кодової комбінації на твірний поліном $q(x)$. Такі пристрої будуються на регістрах зсуву та зазвичай є цілком аналогічними розглянутим вище кодувальним пристроям. Декодувальні пристрої несистематичних циклічних кодів, призначені для виявлення помилок, взагалі майже нічим не відрізняються від розглянутих вище кодувальних. Єдина різниця полягає у тому, що в декодувальних пристроях необхідно використовувати додатковий буфер для збереження кодового слова під час виконання операції ділення. У разі відсутності остачі інформація із цього буфера відразу передається на дешифратор повідомлення, а у разі наявності остачі надсилається сигнал передавальному пристрою на повторне передавання інформації.

Проте для циклічних кодів із виправленням помилок схема декодера у значній мірі ускладнюється. Це пов’язано з тим, що необхідно визначати не лише наявність помилки, але і її синдром, хоча вся інформацію про помилку, як і раніше, закладена в остачах від ділення на твірний поліном. Узагальнена структурна схема декодеру наведена на рис. 2.47.

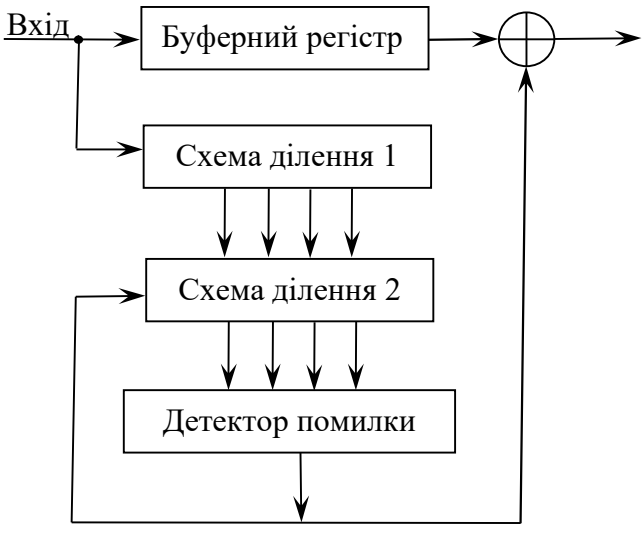


Рис. 2.47 Узагальнена структурна схема декодера циклічних кодів

Декодер, структурна схема якого наведена на рис. 2.47, працює наступним чином. Символи кодової комбінації, яка, можливо, є помилковою, послідовно, починаючи з першого розряду, вводяться до буферного регістру із розрядністю n. Згідно із структурою декодера, одночасно та ж сама комбінація надходить на схему ділення, де за n тактів визначається остача. Якщо схема є високошвидкодійною та працює у неперервному режимі, остача відразу переписується до другої, аналогічної схеми ділення [5, 52, 62, 63].

Починаючи за такту $n+1$ до буферного регістру вже послідовно починають надходити біти іншої кодової комбінації. Тобто, у цей період роботи схеми декодера на кожному такті із буферного регістра уходить 1 символ попередньої комбінації, і одночасно в регістрі другої схеми ділення з'являється нова остача. Тому у структурній схемі декодера, наведеній на рис. 2.47, важливу роль відіграє детектор помилок, який контролює стан комірок другої схеми ділення та, відповідно із сформованими остачами, виявляє синдром помилки S та виправляє необхідні розряди. Зазвичай синдром помилки з'являється, коли помилковий символ займає крайню праву позицію в буферному регістрі [5, 52]. У цьому випадку детектор помилки формує сигнал одиниці та надсилає його на суматор корекції, який цю помилку автоматично виправляє. Одночасно, через ланцюг зворотного зв'язку, сигнал помилки надходить на вхідний суматор другої схеми ділення. В результаті дії цього сигналу виділений синдром помилки змінюється так, що тепер він відповідає більш простому типу помилки, яку також необхідно виправити. Послідовно, через такі зсуви, виявляються всі виділені синдроми. Ознакою коректності виправленої комбінації циклічного коду є те, що всі комірки регістра зсуву другої схеми ділення повинні мати нульовий стан. Якщо ця умова не виконується, вважається, що прийнята кодова комбінація містить помилку, яку не можливо виправити. Описаний алгоритм роботи узагальненої схеми декодера, поданий у вигляді блок-схеми, наведений на рис. 2.48.

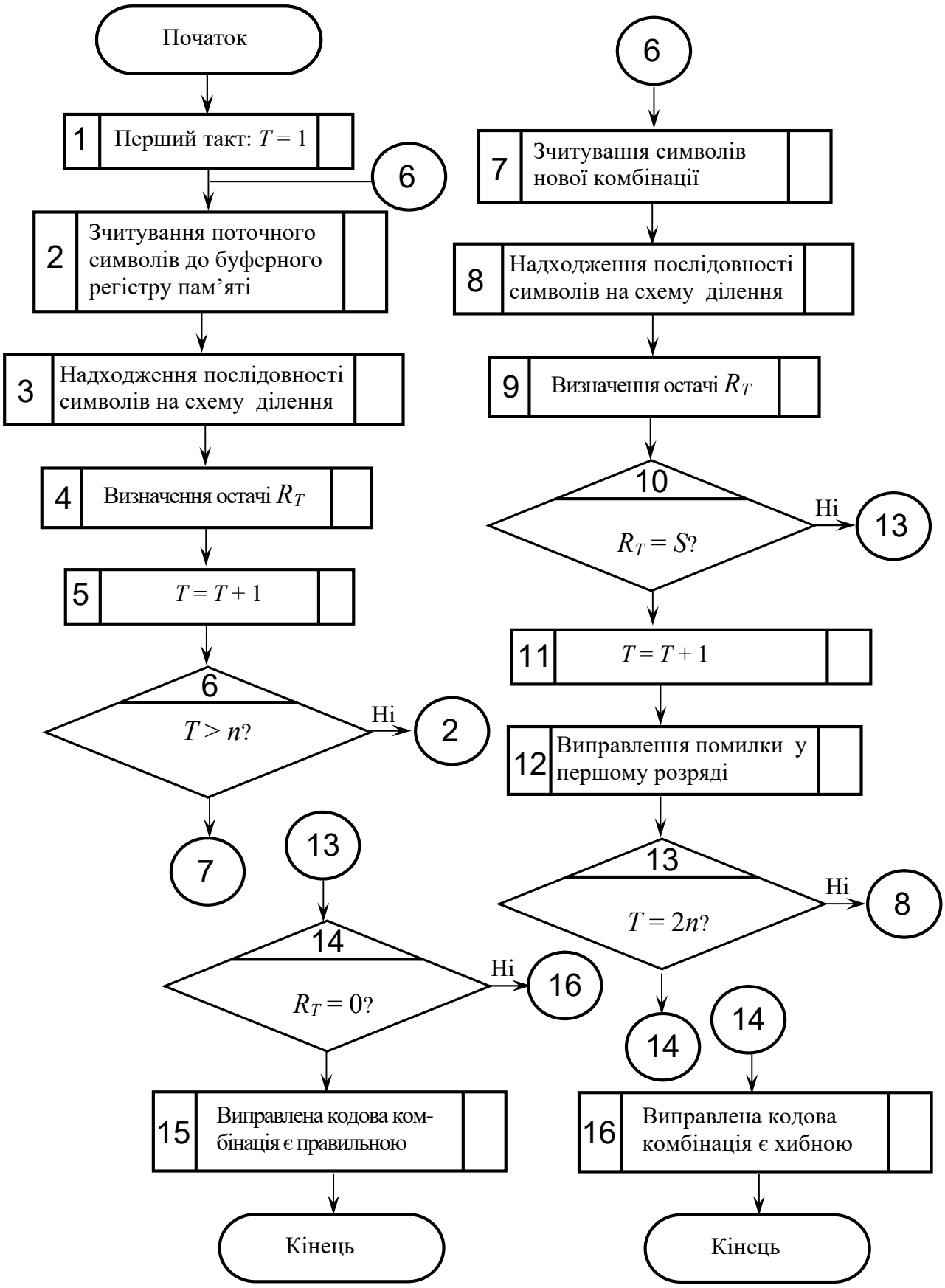


Рис. 2.48 Алгоритм роботи декодера циклічних кодів, оснований на послідовному діленні кодових комбінацій та визначені остач

Для описаного алгоритму розряди вхідної кодової комбінації розташовуються у зворотному порядку, зліва-направо, тому синдром помилки S з'являється раніше за все, а саме, у такті n, якщо помилковим є останній розряд. Для визначення синдрому цієї помилки, згідно із алгоритмом декодування систематичних циклічних кодів, наведеним на рис. 2.37, достатньо розділити кодову комбінацію із одиницею у старшому розряді та нулями у молодших, на твірний поліном $q(x)$. Залежно від номеру помилкового розряду виникають різні остачі, і таким чином визначаються синдроми помилок. Різниця між алгоритмами, наведеним на рис. 2.37 та 2.48 полягає лише у тому, що до молодших розрядів комбінації не дописуються нулі, а після такту n на вхід схеми починають надходити біти нової кодової комбінації, проте на вхід схеми другої ділення надходять нулі. Визначення синдрому помилки проводиться через аналіз остачі з використанням логічних схем. У разі, якщо остача співпадає із синдромом помилки, на вхід схеми суматора за модулем два, розташованої на виході декодера, надходить сигнал одиниці, в результаті чого змінюється значення розряду, який надходить із останньої схеми пам'яті буферного регістру. Зворотні зв'язки між елементами пам'яті у перший та другій схемах ділення відповідають коефіцієнтам твірного поліному, а зв'язки з другої схеми ділення до логічної схеми аналізу остачі відповідають синдромам помилок [5, 52, 62, 63, 67]. Для такого алгоритму роботи електронної схеми декодування немає необхідності у додаткових розрядах в схемі ділення, проте час виправлення помилки в 2 рази перевищує час зчитування кодової комбінації.

Розглянемо приклад побудови декодера циклічного коду для заданої коректувальної здатності для конкретного твірного поліному [5].

Приклад 2.26. Побудувати схему декодера систематичного циклічного коду $(7, 4)$ для твірного поліному $q(x) = x^3 + x^2 + 1$.

Результат ділення кодової комбінації 1000000 на твірний поліном, який у числовій формі має вигляд 1101, наведений на рис. 2.49. Остачі від ділення на першому, другому та третьому тактах відповідають вектору помилки,

тобто, становлять 001, 010 та 100. Згідно із рис. 2.49 остачі від ділення на четвертому, п’ятому, шостому та сьомому тактах складають 101, 111, 011 та 110. Якщо аналізувати помилку в останньому розряді, її синдром становить 110. Схема електронного пристрою, який формує остачу та аналізує появу такого синдрому, наведена на рис. 2.50 [5].

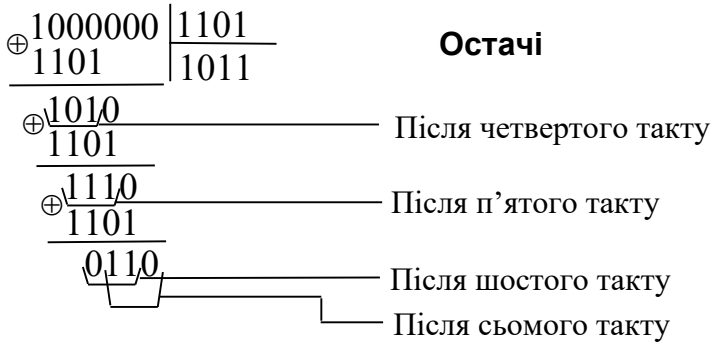


Рис. 2.49 Визначення синдрому одиночної помилки для циклічного коду, сформованого на основі твірного поліному $q(x) = x^3 + x^2 + 1$

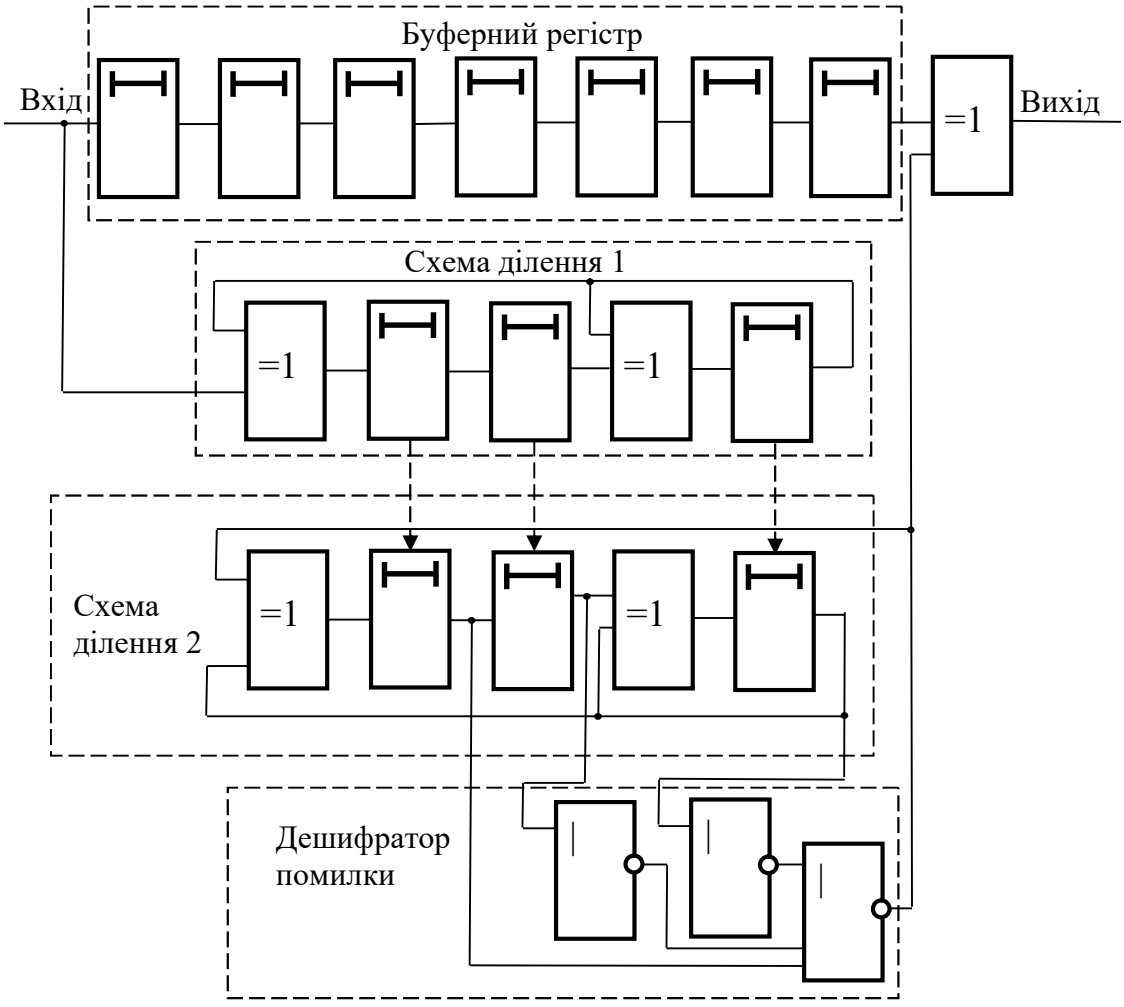


Рис. 2.50 Схема декодера циклічного коду для прикладу 2.26

Логічна електронна схема, наведена на рис. 2.50, складається з трьох частин: буферного регістру, побудована на елементах пам'яті, першої та другої схем ділення, структура яких є ідентичною та відповідає кодувальним схемам, аналогічним наведеній на рис. 2.45, та дешифратора помилки, побудованого на логічних елементах «АБО – НІ».

Проаналізуємо роботу схеми декодера на різних тактах у разі наявності помилки у четвертому розряді. Припустимо, що переданий код числа 1111, який для даного прикладу формується наступним чином:

$$1111 = x^3 + x^2 + x + 1.$$

$$(x^3 + x^2 + 1)\,(x^3 + x^2 + x + 1) = x^6 + x^5 + x^3 + x^5 + x^4 + x^2 + x^4 +$$

$$+ x^3 + x + x^3 + x^2 + 1 = x^6 + x^3 + x + 1 = 1001011.$$

Тобто, безпомилковою кодовою комбінацією є число 1001011, тоді помилці у четвертому розряді відповідає кодова послідовність 1000011.

Стан комірок пам'яті другої схеми ділення для декодера, зображеного на рис. 2.50, показаний в таблиці 2.18. Зрозуміло, що на перших семи тактах роботи декодера заповнюється його буферний регістр, тому на вихід схеми сигнал не надходить. На восьмому та дев'ятому тактах неспотворені символи кодової комбінації, починаючи з молодшого розряду, починаються надходити на вихід схеми. Оскільки остачі від ділення нової кодової комбінації на твірний поліном, яка формується схемою ділення 2, не співпадає із синдромом, на виході дешифратора помилки формується сигнал нуля, тому перші три символи надходять на вихід схеми без змін. Синдром помилки виникає в схемі ділення на десятому такті, оскільки спотворений символ знаходить у крайній правій комірці буферного регістра. За такої умови на виході дешифратора помилки формується сигнал одиниці, тому сигнал на вході правого, останнього суматора за модулем 2, інвертується. Тобто, замість нуля у четвертому біті кодової комбінації отримуємо одиницю. Результатом дії цього сигналу є також переведення всіх комірок пам'яті схеми ділення 2 у нульовий стан, що відповідає закінченню процесу пошуку помилки.

Таблиця 2.18 – Стан комірок схеми декодера, наведеної на рис. 2.50, у разі надходження на її вхід спотвореної кодової послідовності 1000011

Номер такту	Вхід	Стан комірок пам'яті першої та другої схем ділення			Вихід
		1	2	3	
1.	1	1	0	0	–
2.	0	0	1	0	–
3.	0	0	0	1	–
4.	0	1	0	1	–
5.	0	1	1	1	–
6.	1	0	1	0	–
7.	1	1	0	1	Переписується до схеми ділення 2
8.	0	1	1	1	1
9.	0	1	1	0	01
10.	0	1	0	1	001
11.	0	0	0	0	1001
12.	0	0	0	0	01001
13.	0	0	0	0	101001
14.	0	0	0	0	1101001

Існують також логічні електронні схеми декодерів циклічних кодів, які дозволяють визначати синдроми помилок не за n, а за k тактів [5, 52]. Такі схеми мають більшу кількість електричних зв'язків між елементами пам'яті та суматорами і є складнішими, проте передача бітів кодової комбінації на вихід декодера і в цьому випадку також здійснюється за $2n$ тактів. Відрізняються лише синдроми помилок, які для останніх $n - k$ тактів мають

$k-1$ розрядів, а їх значення становлять ...001, ...010, ...0100, 100... . З цієї точки зору схеми декодерів, які працюють за алгоритмами ділення за n тактів та за k тактів, мають майже однакові часові параметри, хоча схеми ділення за k тактів є дещо складнішими [5, 52].

Іншим класом декодувальних схем циклічних кодів є декодери, основані на принципі мажоритарного декодування [5, 52, 62, 63, 67]. Простота алгоритму роботи таких схем полягає у тому, що для будь-якої коректної кодової комбінації система лінійних перевірочних рівнянь, за якими проводиться мажоритарна перевірка, є однаковою. Крім того, схеми декодування, які працюють за мажоритарним принципом, є простішими, оскільки вони у своїй структурі не містять схем ділення.

Для циклічних кодів головна складність розробки алгоритмів мажоритарного декодування полягає у формуванні відповідної системи рівнянь для контрольних перевірок. Зазвичай це є рекурентні співвідношення, які базуються на системі рівнянь (2.100), (2.101). Запишемо ці співвідношення у розгорнутому вигляді через коефіцієнти генераторного поліному $h(x)$ та розряди кодової комбінації a_i [5, 52, 62]:

$$\begin{aligned} &h_0 a_0 \oplus h_1 a_1 \oplus \ldots \oplus h_{k-1} a_{k-1} \oplus a_k = 0; \\ &h_0 a_1 \oplus h_1 a_2 \oplus \ldots \oplus h_{k-1} a_k \oplus a_{k+1} = 0; \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ &h_0 a_{n-k-1} \oplus h_1 a_{n-k} \oplus \ldots \oplus h_{k-1} a_{n+2} \oplus a_{n-1} = 0. \end{aligned} \qquad (2.104)$$

Узагальнена система рівнянь (2.104) розв’язується відносно значень a_i, і таким чином формуються співвідношення для мажоритарних перевірок. Оскільки вигляд такої спрощеної системи рівнянь, сформованої на основі системи (2.104), для різних способів кодування може бути різним, розрізняють три головні варіанти формування та розв’язування такої системи [5, 52, 62].

1. Кожна контрольна рівність дозволяє виразити відповідний символ кодової комбінації a_i через лінійну комбінацію символів, які не входять до будь-яких інших рівнянь. Така система рівнянь називається системою розділених мажоритарних перевірок.

2. Систему роздільних перевірок вдається побудувати лише для деяких d символів, а інші $f = n - d$ символів лінійно зв'язані з ними. Така система рівнянь називається системою квазірозділених мажоритарних перевірок.

3. Кожна контрольна рівність дозволяє виразити будь-який символ a_i через лінійну комбінацію інших символів a_j, проте для будь-яких значень i та j не вдається уникнути повторення цих символів в інших рівностях в результаті алгебраїчних перетворень. Таку систему мажоритарних рівнянь називають λ-зв'язаною [5, 52, 62].

Для λ-зв'язаної системи мажоритарних рівнянь параметр λ визначається через кількість рівнянь, в які входить символ a_j ($j \neq i$). Зрозуміло, що для таких систем рівнянь алгоритми мажоритарних перевірок вкрай ускладнюються, оскільки загальна кількість перевірок, необхідних для виявлення та виправлення помилок у коді, у цьому випадку є значно більшою, ніж для розділених та квазірозділених перевірок [5, 52, 62]. Тому розглянемо приклад формування мажоритарної декодувальної схеми лише для найбільш складної λ-зв'язаної системи лінійних рівнянь [5].

Приклад 2.27. Побудувати схему декодера систематичного циклічного коду (7, 3) для твірного поліному $q(x) = (x + 1)(x^3 + x + 1)$ з використанням принципу мажоритарного декодування. Код із мінімальною кодовою відстанню $d_{\min} = 4$ призначений для виправлення одиночних та визначення подвійних помилок.

Для розв'язування цієї задачі насамперед необхідно визначити генераторний поліном $h(x)$ як результат ділення поліному $x^7 + 1 = 10000001$ на твірний поліном $q(x) = (x + 1)(x^3 + x + 1) = x^4 + x^2 + x + x^3 + x + 1 = x^4 + x^3 + x^2 + 1 = 11101$.

Процес цього ділення показаний на рис. 2.51.

Тепер, враховуючи, що генераторний поліном для циклічного коду, який розглядається, має вигляд $1101 = x^3 + x^2 + 1$, тобто, $h_0 = 1$, $h_1 = 0$, $h_2 = 1$, $h_3 = 1$, перепишемо систему рекурентних рівнянь (2.104) у наступному вигляді:

$$\begin{cases} a_3 = h_0 a_0 \oplus h_1 a_1 \oplus h_2 a_2; \\ a_4 = h_0 a_1 \oplus h_1 a_2 \oplus h_2 a_3; \\ a_5 = h_0 a_2 \oplus h_1 a_3 \oplus h_2 a_4; \\ a_6 = h_0 a_3 \oplus h_1 a_4 \oplus h_2 a_5. \end{cases} \Rightarrow \begin{cases} a_3 = a_0 \oplus a_2; \\ a_4 = a_1 \oplus a_3; \\ a_5 = a_2 \oplus a_4; \\ a_6 = a_3 \oplus a_5. \end{cases} \qquad (2.105)$$

$$
\begin{array}{l}
\oplus\begin{array}{l}10000001\\11101\end{array}\Big|\begin{array}{l}11101\\\hline 1101\end{array}\\
\quad\oplus\begin{array}{l}11010\\11101\end{array}\\
\quad\quad\oplus\begin{array}{l}011101\\\ \ 11101\end{array}\\
\quad\quad\quad\ \ 0
\end{array}
$$

Рис. 2.51 Результат ділення поліному $x^7 + 1$ на твірний поліном $x^3 + x^2 + 1$

Із (2.105), враховуючи тривіальну тотожність $a_0 = a_0$, через аналітичні перетворення отримуємо такі лінійні рівняння для мажоритарного пошуку значення символу a_0.

$$a_0 = a_3 \oplus a_2;$$

$$a_4 = a_1 \oplus a_3 = a_1 \oplus a_0 \oplus a_2 = a_1 \oplus a_0 \oplus a_5 \oplus a_4; \Rightarrow$$

$$\Rightarrow a_0 = a_1 \oplus a_4 \oplus a_5 \oplus a_4 = a_1 \oplus a_5;$$

$$a_5 = a_2 \oplus a_4; \Rightarrow a_6 \oplus a_3 = a_2 \oplus a_4; \Rightarrow a_6 \oplus a_3 = a_2 \oplus a_1 \oplus a_3; \Rightarrow$$

$$\Rightarrow a_6 = a_2 \oplus a_1; \Rightarrow a_6 = a_0 \oplus a_3 \oplus a_3 \oplus a_4 = a_0 \oplus a_4; \Rightarrow a_0 = a_4 + a_6.$$

Тобто, остаточно маємо:

$$\begin{cases} a_0 = a_3 \oplus a_2; \\ a_0 = a_1 \oplus a_5; \\ a_0 = a_4 \oplus a_6; \\ a_0 = a_0. \end{cases} \qquad (2.106)$$

Слід відзначити, що мажоритарне декодування циклічних кодів реалізується за n циклів і аналізується лише перший розряд коду a_0, який знаходиться у крайній правій комірці пом’яті. Враховуючи те, що співвідношення (2.106) виконуються для всіх комбінацій циклічного коду, ця система рівнянь є достатньою для визначення подвійної помилки або для виправлення одиночної у будь-якому розряді за n тактів. Відповідна електронна схема кодувального пристрою наведена на рис. 2.52, а значення сигналу у контрольних точках схеми у разі подання на її вхід кодової послідовності 1001110 – у таблиці 2.19.

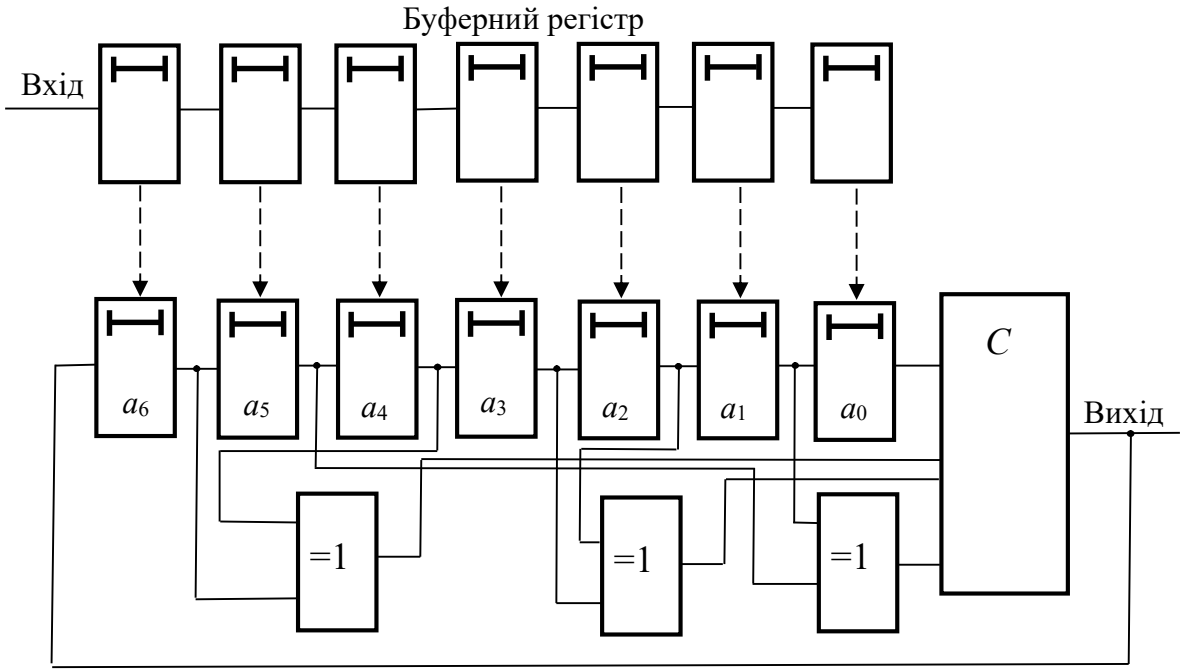


Рис. 2.52 Схема мажоритарного декодування циклічних кодів

Таблиця 2.19 – Стан комірок пам'яті та значення сигналів в контрольних точках схеми декодера, наведеного на рис. 2.52

Номер такту	Стан комірок регістра							Вхід компа-ратора C	Вихід
	a_6	a_5	a_4	a_3	a_2	a_1	a_0		
1.	0	1	1	0	0	0	1	0111	1
2.	1	0	1	1	0	0	0	1000	01
3.	0	1	0	1	1	0	0	0100	001
4.	0	0	1	0	1	1	0	1110	1001
5.	1	0	0	1	0	1	1	1111	11001
6.	1	1	0	0	1	0	1	1111	111001
7.	1	1	1	0	0	1	0	0000	0111001

Схема мажоритарного декодування, наведена на рис. 2.52, працює наступним чином. Кодова комбінація, яка за n тактів надходить в буферний регістр, паралельно записується в регістр декодера. На кожному з n наступних тактів здійснюється циклічний зсув бітів праворуч на одну

позицію, а на виході компаратора чотирьох сигналів C сигнал формується за мажоритарним принципом. Наприклад, на першому такті для записаного числа 1001110 маємо.

$a_6 = 0; a_5 = 1; a_4 = 1; a_3 = 0; a_2 = 0; a_1 = 0; a_0 = 1.$

$a_3 \oplus a_2 = 0;$ $a_1 \oplus a_5 = 1;$ $a_4 \oplus a_6 = 1;$ $a_0 = 1.$ На виході схеми мажоритарного порівняння буде сигнал 1, оскільки на вході – три одиниці та один нуль.

У схемі, наведеній на рис. 2.52, формування контрольних сум (2.106) здійснюється та трьох суматорах, зв'язаних з регістрами пам'яті декодера, а сигнал a_0 надходить на вхід схеми безпосередньо.

Відповідно, для другого такту маємо.

$a_6 = 1; a_5 = 0; a_4 = 1; a_3 = 1; a_2 = 0; a_1 = 0; a_0 = 0.$

$a_3 \oplus a_2 = 1;$ $a_1 \oplus a_5 = 0;$ $a_4 \oplus a_6 = 0;$ $a_0 = 0.$ На виході схеми мажоритарного порівняння буде сигнал 0, оскільки на вході – три нулі та одна одиниця.

Аналогічно аналізуються сигнали на решті п'яти тактах роботи схеми, результати цього аналізу наведені у таблиці 2.19.

Окремо розглядається випадок, коли із чотирьох сигналів на вході компаратора два дорівнюють одиниці, а два – нулю. У цьому випадку вважається, що кодова комбінація має подвійну помилку, яка виявляється, але не може бути виправленою.

Слід відзначити, що зазвичай у сучасних електронних системах та системах зв'язку процес кодування та декодування послідовностей циклічних кодів часто здійснюється з використанням відповідних програмних засобів. Для несистематичних кодів використовуються алгоритм кодування, пов'язаний із множенням поліномів та розглянутий у прикладі 2.16, та алгоритм декодування, наведений на рис. 2.16. Для систематичних кодів використовують алгоритми кодування та декодування, розглянуті у прикладах 2.23 та 2.24. Алгоритм декодування систематичних циклічних кодів у вигляді блок-схеми наведений також на рис. 2.37. Як бачимо, всі ці

алгоритми базуються на процедурах алгебраїчного множення та ділення поліномів та визначені остач. У наступному підрозділі буде розглянута комп'ютерна програма, призначена для формування кодових послідовностей циклічних кодів та їхнього декодування, написана з використанням відомих засобів структурного, матричного та модульного програмування системи науково-технічних розрахунків MatLab [13, 14, 48].

2.5.8 Комп'ютерна реалізація алгоритмів формування циклічних кодів та декодування їхніх кодових послідовностей

Програма для формування циклічних кодів та декодування їхніх кодових послідовностей, написана з використанням засобів програмування системи MatLab, наведена у додатку К. Головною особливістю реалізації цієї програми є блочно-модульний принцип побудови. Головними функціональними блоками програми є наступні.

1. Процедура для покрокового множення двійкових послідовностей **`Bin_Product`**.

2. Процедура для покрокового ділення двійкових послідовностей **`Bin_Division`**.

3. Процедура для циклічного зсуву векторів двійкових послідовностей **`CSHRL`**.

4. Процедура для формування несистематичних та систематичних циклічних кодів та їхнього декодування **`CRC`**.

У програмному модулі процедури **`CRC`** з використанням засобів структурного та матричного програмування системи MatLab реалізовані алгоритми формування послідовностей циклічних кодів та їхнього декодування, які були розглянуті у цьому підрозділі, і цей модуль не має суттєвих алгоритмічних особливостей. Із нього викликаються процедури поліноміального множення та ділення двійкових послідовностей **`Bin_Product`** та **`Bin_Division`**. Зрозуміло, що процедура **`Bin_Product`** застосовується лише для формування несистематичних кодів, відповідний

спосіб отримання кодових послідовностей через безпосереднє множення кодового слова на твірний поліном є досить простим та був розглянутий у підрозділі 2.5.1. Процедура ділення **Bin_Division** викликається із програмного модуля **CRC** значно частіше, оскільки узагальнена операція алгебраїчного ділення двійкових послідовностей, згідно із розглянутою теорією циклічних кодів, є базовою для виконання наступних комплексних дій.

1. Декодування кодових послідовностей несистематичних кодів. Відповідний алгоритм декодування наведений на рис. 2.16. Крім операції ділення цей алгоритм передбачає виконання циклічного зсуву кодових комбінацій ліворуч та праворуч. Для виконання цих елементарних операцій над двійковими послідовностями із модуля **CRC** викликається процедура **CSHRL** (абревіатура англійського словосполучення Cyclic SHift Right and Left – циклічний зсув ліворуч та праворуч). Напрямок зсуву для цієї процедури задається значенням параметра виклику **nop**: значення 1 відповідає зсуву ліворуч, а значення 0 – зсуву праворуч.

2. Формування кодових послідовностей систематичних циклічних кодів через ділення вхідного інформаційного слова із відповідною кількістю дописаних нулів на твірний поліном та додавання остачі, відповідний алгоритм є досить простим та був описаний у підрозділі 2.5.5.

3. Формування синдрому помилки для систематичного циклічного коду через ділення вектора помилки у останньому розряді на твірний поліном.

4. Пошук помилкового розряду у послідовностях систематичних циклічних кодів через ділення спотвореної кодової комбінації на твірний поліном та додавання відповідної кількості нулів. Відповідний алгоритм пошуку помилки через визначений синдром наведений на рис. 2.37.

Вхідними параметрами для виклику функції **CRC** є наступні.

vin – вхідний вектор двійкової послідовності, яку необхідно кодувати або декодувати. Всі елементи вектора повинні мати значення 0 або 1. Діапазон довжини вхідного вектора у разі виконання операції кодування становить від 4 до 26 елементарних символів, а у разі виконання операції

кодування – від 7 до 31 елементарних символів.

nop – тип операції, яку необхідно виконати. Можливі значення цього параметру: 1 – кодування інформаційного слова, 2 – декодування кодової послідовності.

TOC – тип циклічного коду. Можливі значення цього параметру: 1 – несистематичний код, 2 – систематичний код.

Оскільки алгоритми формування векторів кодових послідовностей циклічних кодів та їхнього декодування у цьому підрозділі були розглянуті досить досконало, а комп'ютерна реалізація цих алгоритмів з використанням засобів програмування системи MatLab є простою та не має суттєвих лінгвістичних особливостей, код програми **CRC** є зрозумілим та не потребує подальшого ретельного аналізу. Програма написана з використанням стандартних методів структурного та функціонального програмування, функцій MatLab для роботи з векторами, а також матричних макроопераций системи MatLab, зокрема множинної індексації структурованих даних [13, 14, 48].

На відміну від цього, алгоритми ітераційного множення та ділення двійкових послідовностей реалізовані нестандартно через функції обробки рядків матриць, тому розглянемо їх окремо. Ці алгоритми загалом базуються на двох елементарних операціях – на дописуванні нулів до двійкових послідовностей ліворуч та праворуч та на сумуванні цих послідовностей за модулем 2. В основу цих алгоритмів покладені ітераційні дії, які цілком аналогічні множенню та діленню чисел в стовпчик в арифметиці. Такі дії є загальновідомими та неодноразово використовувались як у цьому підрозділі, так і у другій частині посібника [48] для розв'язування практичних задач теорії кодування. Проте ручний та автоматизований способи реалізації алгоритмів множення та ділення двійкових послідовностей дещо відрізняються. Автоматизований спосіб реалізації цих алгоритмів передбачає структуризацію двійкових послідовностей, упорядкованість та індексацію всіх їхніх елементів та формалізацію дій над цими елементами. Оскільки система MatLab загалом орієнтована на роботу з матрицями, таку

формалізацію алгоритмів множення та ділення зручніше за все реалізувати через формування рядків відповідної матриці. Розглянемо окремо алгоритми формування матриць множення та ділення двійкових послідовностей.

Матриця множення формується наступним чином.

1. Визначається кількість елементарних елементів для обох множників, або їх довжина.

2. Той множник, який має більшу довжину k, береться як базовий, а множник меншої довжини l вважається допоміжним.

3. Формується вектор $\mathbf{V_1}$ із довжиною $k + l - 1$, спосіб формування якого залежить від останніх елементів допоміжної комбінації. Якщо останнім елементом допоміжної комбінації є 1, початковими елементами вектора $\mathbf{V_1}$ є k нулів, за якими слідує базова комбінація. Наприклад, якщо базова кодова комбінація **B** = [1, 0, 1, 1, 0], а допоміжна – **S** = [1, 0, 1], вектор $\mathbf{V_1}$ записується як $\mathbf{V_1}$ = [0, 0, 0, 1, 0, 1, 1, 0]. Якщо останніми елементами допоміжної комбінації є s нулів, вони дописуються до базової двійкової комбінації праворуч, а ліворуч ставляться $k - s - 1$ нулів. Наприклад, для базової комбінації **B** = [1, 0, 1, 1, 0] та допоміжної **S** = [1, 0, 0] вектор $\mathbf{V_1}$ складає $\mathbf{V_1}$ = [1, 0, 1, 1, 0, 0, 0].

4. Формується вектор $\mathbf{V_2}$ із довжиною $k + l - 1$, спосіб формування якого також залежить від елементів допоміжної комбінації. Якщо після одиниці у допоміжній комбінації стоїть s нулів, останніми елементами вектора $\mathbf{V_2}$ буде $s + 1$ нуль. Ці нулі дописуються до базової комбінації праворуч, а ліворуч ставляться $k - s - 1$ нулів.

5. Аналогічно формується вектор $\mathbf{V_i}$ на ітерації з номером i. Праворуч до базової комбінації приписуються $s + i$ нулів, а ліворуч – $k - s - i - 1$ нулів.

6. Елементи всіх векторів записуються до матриці множення **M** із розмірністю r x t, де $t = k + l - 1$, r – кількість ітерацій, яка відповідає кількості одиниць у допоміжній кодовій комбінації.

7. Всі рядки матриці **M** сумуються за модулем два.

Блок-схема описаного матричного алгоритму множення двійкових

чисел наведена на рис. 2.53.

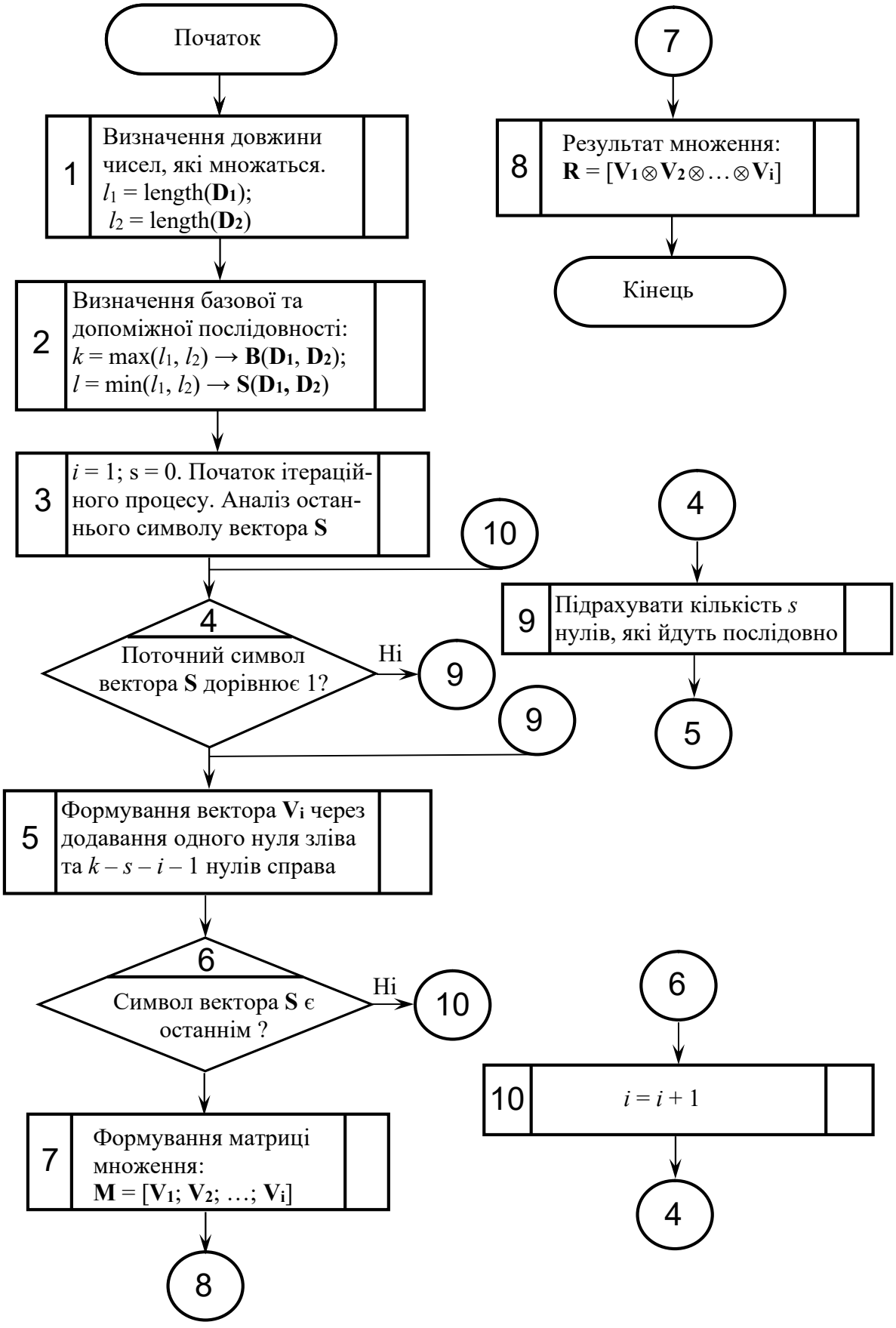


Рис. 2.53 Блок-схема матричного алгоритму множення двійкових чисел

Розглянемо приклад множення двійкових чисел за матричним алгоритмом, блок-схема якого наведена на рис. 2.53.

Приклад 2.28. З використанням описаного алгоритму матричного множення знайти добуток двійкових чисел 100101 та 1101.

Будемо розв'язувати поставлену задачу ітераційно, згідно з описаним алгоритмом, послідовно записуючи вектори $\mathbf{V_i}$, сформовані на основі базової кодової комбінації, до матриці множення **M**. Зрозуміло, що для даного прикладу базовою двійковою послідовністю є вектор **B** = [1, 0, 0, 1, 0, 1], а допоміжною – вектор **S** = [1, 1, 0, 1]. Тобто, $k = 6$, $l = 4$, $t = k + l - 1 = 9$.

Для прикладу, який розглядається, з урахуванням визначених умов, ітераційний процес формування матриці множення має наступний вигляд.

1. Оскільки останнім елементом допоміжного вектору є число 1, на першій ітерації необхідно до базової кодової послідовності дописати зліва 3 нулі. Тобто, $\mathbf{V_1}$ = [0, 0, 0, 1, 0, 0, 1, 0, 1].

2. Другий елемент допоміжного вектору дорівнює 0, а третій – одиниці. Тому на другій ітерації дописуємо до базової кодової комбінації два нулі праворуч та один нуль ліворуч. Тобто, $\mathbf{V_2}$ = [0, 1, 0, 0, 1, 0, 1, 0, 0].

3. Четвертий, останній елемент допоміжного вектора також дорівнює 1. Тому останній, третій вектор формується через дописування до базової кодової комбінації трьох нулів праворуч. Тобто, $\mathbf{V_3}$ = [1, 0, 0, 1, 0, 1, 0, 0, 0].

В результаті аналізу останнього біту допоміжної двійкової послідовності завершується ітераційний процес формування векторів. Матриця множення має вигляд:

$$\mathbf{M} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

Тепер, сумуючи рядки матриці множення **M** за модулем два, отримуємо остаточний результат множення двох двійкових чисел. Виконуючи цю операцію поелементно, маємо наступний результат:

$R_1 = 0 \otimes 0 \otimes 1 = 1$; $R_2 = 0 \otimes 1 \otimes 0 = 1$; $R_3 = 0 \otimes 0 \otimes 0 = 0$; $R_4 = 1 \otimes 0 \otimes 1 = 0$;

$R_5 = 0 \otimes 1 \otimes 0 = 1$; $R_6 = 0 \otimes 0 \otimes 1 = 1$; $R_7 = 1 \otimes 1 \otimes 0 = 0$; $R_8 = 0 \otimes 1 \otimes 0 = 0$;

$R_9 = 1 \otimes 0 \otimes 0 = 1$.

Тобто, остаточний вектор результату множення: **R** = [1, 1, 0, 0, 1, 1, 0, 0, 1].

Алгоритм формування матриці ділення двійкових чисел можна записати у вигляді тезових формулювань наступним чином.

1. Визначається кількість біт у діленому k та у дільнику l. За умови $k < l$ вважається, що частка дорівнює 0, а ділене є остачею. На першій ітерації вважається $s_1 = 0$.

2. Першим рядком матриці ділення **D** на першій ітерації є ділене, а на наступних ітераціях першим рядком вважається попередній результат.

3. Другий рядок матриці ділення формується на кожній ітерації через дописування до дільника $k - l - s$ нулів ліворуч та s нулів праворуч. Зрозуміло, що довжина цього вектора $\mathbf{W_i}$, якій формується на кожній ітерації, дорівнює k.

4. Третій рядок матриці ділення формується на кожній ітерації як результат сумування за модулем два першого та другого рядка.

5. На поточній ітерації підраховується кількість нулів s_i, які стоять зліва. Якщо $d = s_i - s_{i-1} > 1$, до частки **Q** дописується d нулів.

6. До частки **Q** записується значення 1.

7. Закінчення поточної ітерації та перехід до наступної за умови $k - s_i \geq l$. У противному випадку процес ділення вважається закінченим. Результатом ділення є отримана частка **Q** та остача **R**, який відповідає отриманий результат сумування.

8. Третій рядок попередньої ітерації вважається першим рядком наступної та містить зліва s_i нулів. Оскільки $k - s_i \geq l$, вважається, що $i = i + 1$ та пункти алгоритму 2 – 6 повторюються. У противному випадку процес ділення закінчується за пунктом 7.

Блок-схема описаного алгоритму ділення наведена на рис. 2.54.

Розглянемо приклад формування матриці ділення.

Приклад 2.29. З використанням описаного алгоритму матричного ділення знайти частку та остачу від ділення двійкових чисел 10010101 та 1101. Результат записати у вигляді матриці, першим рядком якої є частка, а другим – остача.

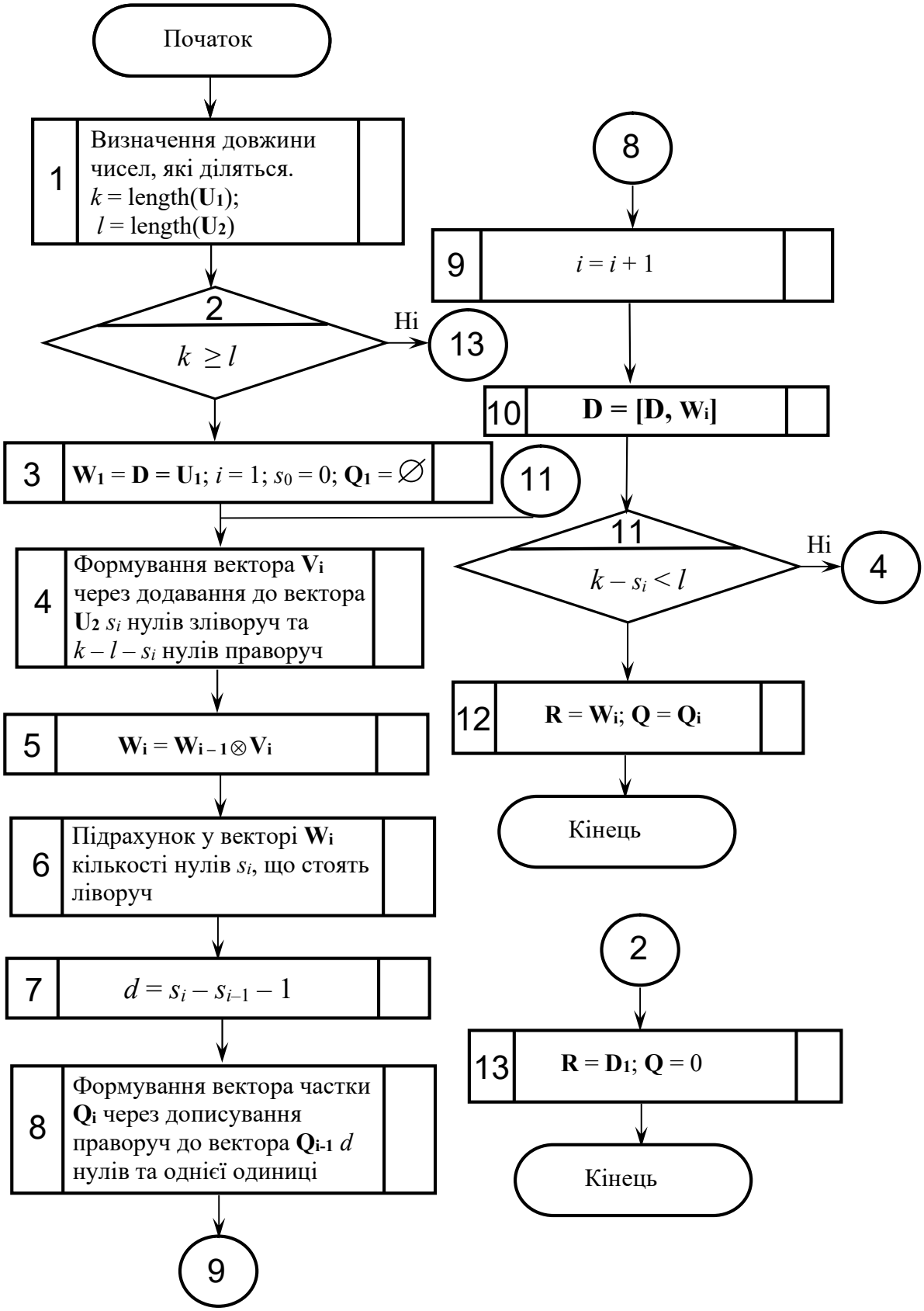


Рис. 2.54 Блок-схема матричного алгоритму ділення двійкових чисел

1. Першим рядком матриці ділення **D** буде вектор діленого, тобто $\mathbf{D_1} = \mathbf{U_1} = [1, 0, 0, 1, 0, 1, 0, 1]$. У другий рядок записується вектор дільника $\mathbf{U_2}$ із дописаними праворуч чотирма нулями, тобто $\mathbf{D_2} = \mathbf{V_1} = [1, 1, 0, 1, 0, 0, 0, 0]$. Сумуючи за модулем два перший та другий рядки матриці ділення **D**, отримуємо третій рядок цієї матриці:

$$\mathbf{D}_3 = \mathbf{D_1} \oplus \mathbf{D_2} =$$

$$= \left[1 \oplus 1, \quad 0 \oplus 1, \quad 0 \oplus 0, \quad 1 \oplus 1, \quad 0 \oplus 0, \quad 1 \oplus 0, \quad 0 \oplus 0, \quad 1 \oplus 0\right] =$$

$$= \left[0, \quad 1, \quad 0, \quad 0, \quad 0, \quad 1, \quad 0, \quad 1\right].$$

Оскільки у третьому рядку матриці зліва стоїть лише 1 нуль, у вектор частки **Q** вписується на цій ітерації лише одна одиниця.

2. Четвертим рядком матриці **D** буде вектор дільника $\mathbf{U_2}$ із дописаними праворуч трьома нулями та із одним дописаним нулем ліворуч, тобто $\mathbf{D_4} = \mathbf{V_2} = [0, 1, 1, 0, 1, 0, 0, 0]$. А п'ятий рядок матриці ділення **D** формується як сума за модулем два третього та четвертого рядків наступним чином:

$$\mathbf{D}_5 = \mathbf{D}_3 \oplus \mathbf{D}_4 =$$

$$= \left[0 \oplus 0, \quad 1 \oplus 1, \quad 0 \oplus 1, \quad 0 \oplus 0, \quad 0 \oplus 1, \quad 1 \oplus 0, \quad 0 \oplus 0, \quad 1 \oplus 0\right] =$$

$$= \left[0, \quad 0, \quad 1, \quad 0, \quad 1, \quad 1, \quad 0, \quad 1\right].$$

У п'ятому рядку матриці зліва стоять два нулі, що на 1 нуль більше, ніж було у другому. Тому і на цій ітерації до частки необхідно дописати лише одну одиницю. Тобто, вектор частки на цій ітерації дорівнює $\mathbf{Q} = [1, 1]$.

3. Для формування шостого рядка матриці **D** необхідно до вектора дільника $\mathbf{U_2}$ дописати по два нуля праворуч та ліворуч. Тобто, $\mathbf{D_6} = \mathbf{V_3} = [0, 0, 1, 1, 0, 1, 0, 0]$. Тоді сьомий рядок цієї матриці обчислюється наступним чином:

$$\mathbf{D}_7 = \mathbf{D}_5 \oplus \mathbf{D}_6 =$$

$$= \left[0 \oplus 0, \quad 0 \oplus 0, \quad 1 \oplus 1, \quad 0 \oplus 1, \quad 1 \oplus 0, \quad 1 \oplus 1, \quad 0 \oplus 0, \quad 1 \oplus 0\right] =$$

$$= \left[0, \quad 0, \quad 0, \quad 1, \quad 1, \quad 0, \quad 0, \quad 1\right].$$

Обчислений сьомий рядок матриці **D** містить три нулі, на один більше, ніж п'ятий, тому і на цій ітерації дописуємо до частки лише одну одиницю.

На цій ітерації вектор частки становить **Q** = [1, 1, 1].

4. Для формування восьмого рядка матриці **D** необхідно до вектора дільника $\mathbf{U_2}$ дописати три нулі зліва та один нуль справа. Таким чином, $\mathbf{D_8} = \mathbf{V_4} = [0, 0, 0, 1, 1, 0, 1, 0]$. Тоді дев'ятим рядком матриці буде наступна послідовність:

$$\mathbf{D}_9 = \mathbf{D}_7 \oplus \mathbf{D}_8 =$$

$$= [0 \oplus 0, \quad 0 \oplus 0, \quad 0 \oplus 0, \quad 1 \oplus 1, \quad 1 \oplus 1, \quad 0 \oplus 0, \quad 0 \oplus 1, \quad 1 \oplus 0] =$$

$$= [0, \quad 0, \quad 0, \quad 0, \quad 0, \quad 0, \quad 1, \quad 1].$$

Оскільки у отриманому дев'ятому рядку матриці ділення зліва стоїть шість нулів, тобто, виконується умова $k - s_i < l$, процес ділення можна вважати закінченим. Остаточний результат розрахунків: частка – **Q** = [1, 1, 1, 1], остача – **R** = [1, 1]. Матриця ділення **D** має вісім стовпчиків та дев'ять рядків та записується у вигляді:

$$\mathbf{D} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

Результат розрахунків, проведених з використанням матриці ділення, можна записати у матричній формі наступним чином:

$$\mathbf{Res} = \begin{bmatrix} \mathbf{Q} \\ \mathbf{R} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}.$$

У програмі **`Bin_Division`** результат ділення двійкових послідовностей також записується у матричній формі, що спрощує структуризацію даних та обмін даними між процедурами [14, 15]. Зрозуміло, що після передавання даних до процедури формування циклічних кодів **`CRC`** зайві нулі, що стоять зліва, видаляються. Такі зайві нулі у старших розрядах можуть стояти як у частці, так і в остачі, залежно від кількості розрядів у діленому та дільнику, оскільки для подання частки та остачі у матричній

формі кількість озрядів в них має співпадати.

Використання матричних макрооперацій системи науково-технічних розрахунків MatLab для формування циклічних кодів та декодування їхніх кодових послідовностей дозволяє у значній мірі спростити програмний код та організувати ефективну обробку числових даних. Розроблені програмі засоби є досить універсальними та можуть бути використані в складних програмних комплексах, призначених для моделювання та проектування сучасних кодувальних електронних систем [75 – 77].

Контрольні питання та завдання до розділу 2

1. Поясніть сутність теореми Шеннона 2.1 та зворотної до неї теореми 2.2.

2. Наведіть доведення теореми 2.1.

3. Поясніть сутність співвідношень (2.1) – (2.14).

4. Наведіть доведення теореми Котельникова – Найквіста.

5. Поясніть сутність співвідношень (2.15) – (2.26).

6. Що являють собою блокові коди? Наведіть приклади таких кодів.

7. Що являють собою неперервні коди? Наведіть приклади таких кодів.

8. Що являють собою рівномірні коди? Наведіть приклади таких кодів.

9. Що являють собою роздільні та нероздільні блокові коди? Наведіть приклади таких кодів.

10. Що являють собою лінійні коди? Наведіть приклади таких кодів.

11. Що являють собою циклічні коди? Наведіть приклади таких кодів.

12. Що являють собою неперервні, або деревоподібні коди та як вони формуються?

13. Поясніть головний принцип побудови блокових завадостійких кодів, описаний у підрозділі 2.2.2.

14. Поясніть співвідношення (2.27) – (2.33).

15. Поясніть принцип побудови блокових завадостійких кодів з точку зори відстані між кодовими комбінаціями, а також рис. 2.1.

16. Поясніть визначення 2.1 та 2.2.

17. Поясніть співвідношення (2.34).

18. Поясніть співвідношення (2.35) – (2.37) для мінімальної кодової відстані завадостійких кодів. Наведіть приклади використання цих співвідношень.

19. Поясніть геометричну інтерпретацію принципу роботи завадостійких кодів із виправленням помилок, яка показана на рис. 2.2.

20. Поясніть геометричну інтерпретацію кодових відстаней між двійковими сигналами у вигляді гіперкубів, яка наведена на рис. 2.3.

21. Що являє собою надлишковість завадостійкого коду та які способи розрахунку цього параметру Вам відомі?

22. Поясніть співвідношення (2.38), (2.39).

29. Що являє собою оптимальний код? Наведіть приклади таких кодів.

30. Поясніть співвідношення (2.40) – (2.42).

31. Поясніть співвідношення, наведені у таблиці 2.1.

32. Що являє собою щільноупакований код? Наведіть приклади таких кодів.

33. Які стаціонарні параметри систематичних блокових завадостійких кодів Вам відомі? Наведіть власні приклади таких параметрів.

34. Чому мінімальна кодова відстань $d_{\min}$ вважається стаціонарним параметром завадостійкого коду? Свою відповідь обґрунтуйте.

35. Які параметри окремих кодових комбінацій для систематичних блокових завадостійких кодів Вам відомі? Наведіть власні приклади таких параметрів.

36. Які імовірнісні параметри систематичних блокових завадостійких кодів Вам відомі? Наведіть власні приклади таких параметрів.

37. Чому імовірність спотворення одного біту $p_б$ вважається основним імовірнісним параметром завадостійкого коду? Свою відповідь обґрунтуйте.

38. Що являє собою параметр завадостійкості коду та як він розраховується? Поясніть співвідношення (2.44).

39. Що являє собою коефіцієнт виявлення помилок та як він

розраховується? Поясніть співвідношення (2.44).

40. Що являє собою коефіцієнт виправлення помилок та як він розраховується? Поясніть співвідношення (2.45).

41. Які параметри надлишковості систематичних блокових завадостійких кодів Вам відомі? Наведіть власні приклади таких параметрів.

42. Які інформаційні параметри систематичних блокових завадостійких кодів вам відомі? Наведіть власні приклади таких параметрів.

43. Як інформаційні параметри завадостійких кодів пов’язані із параметрами надлишковості? Свою відповідь обґрунтуйте.

44. Яким чином поняття ентропії використовується для визначення інформаційних параметрів завадостійкого коду? Наведіть власні приклади такого використання.

45. Поясніть співвідношення (2.47) – (2.51).

46. Що являє собою границя Варшамова – Гільберта у теорії завадостійкого кодування а як вона пов’язана із границею Хеммінга? Свою відповідь обґрунтуйте.

47. Поясніть класифікаційну діаграму параметрів завадостійких кодів, яка наведена на рис. 2.4.

48. Що являє собою код із перевіркою на парність та яку він має коректувальну здатність? Наведіть власні приклади формування коду із перевіркою на парність.

49. Поясніть приклад 2.1.

50. Що являє собою код із повторенням елементів та яку він має коректувальну здатність? Наведіть власні приклади формування коду із повторенням елементів.

51. Поясніть приклад 2.2.

52. Які параметри надлишковості має код із повторенням елементів? Поясніть співвідношення (2.52). Як вони пов’язані із співвідношеннями (2.38), (2.39)?

53. Поясніть приклад 2.3.

54. Яким чином код із повторенням елементів пов’язаний із манчестерськими кодами, розглянутими у підрозділі 1.2.4? Свою відповідь обґрунтуйте та поясніть, чи можуть бути пов’язані між собою способи фізичного та логічного кодування сигналів?

55. Чи відповідає зв’язок між фізичним та логічним кодуванням сигналів еталонній моделі OSI, яка була розглянута у першій частині посібника? Свою відповідь обґрунтуйте.

56. Як пов’язані спектри натуральних кодів із спектрами завадостійких кодів із виявленням помилок, які відповідають цим натуральним кодам? Які з цих спектрів мають мінімальну, а які – максимальну енергію, і чому?

57. Поясніть графічні залежності, які наведені на рис. 2.5.

58. Що являє собою прямокутний код та як він формується? Яку він має коректувальну здатність та які параметри надлишковості? Наведіть приклади формування прямокутного коду.

59. Поясніть співвідношення (2.53). Як вони пов’язані із співвідношеннями (2.38), (2.39)?

60. Поясніть приклад 2.4.

61. Як код із перевіркою на парність та код із повторенням елементів використовуються в алгоритмах побудови лінійних та циклічних кодів та декодування їхніх послідовностей? Свою відповідь обґрунтуйте та наведіть приклади такого використання.

62. Для визначених кодових послідовностей побудувати код із перевіркою на парність та код із повторенням елементів із визначеною кількістю повторень. Початкові дані для розв’язування цього завдання наведені у таблиці 2.20.

63. Для кодових послідовностей, заданих у таблиці 2.20, побудувати прямокутний код. Порядок слідування послідовностей відповідає порядку варіантів, наведеному у четвертому стовпчику таблиці 2.20.

64. Що являє собою інверсний код і який спосіб його формування?

65. Поясніть принцип роботи електричних схем, наведених на рис. 2.6, а, б.

66. Поясніть приклад 2.5 та рис. 2.7.

67. Для кодових послідовностей, визначених у другому стовпчику таблиці 2.20, побудувати інверсний код.

Таблиця 2.20 – Початкові дані для завдань 62 та 63

Номер варіанту	Число, яке кодується	Кількість повторень	Порядок слідування варіантів для прямокутного коду
1.	10110101	2	1, 10, 5, 13, 2
2.	10001000	3	2, 6, 4, 10, 1
3.	10101010	2	3, 4, 12, 5, 1
4.	11110000	4	9, 15, 3, 4, 5
5.	10111110	3	2, 8, 7, 3, 2
6.	11111110	5	6, 4, 9, 14, 4
7.	10010010	4	7, 14, 13, 12, 3
8.	11001100	3	6, 10, 11, 7, 15
9.	10100101	2	15, 3, 5, 8, 7
10.	10000000	4	14, 9, 10, 12, 1
11.	10000001	5	11, 10, 9, 8, 7
12.	10000101	3	12, 5, 3, 12, 1
13.	10011100	2	7, 3, 4, 2, 1
14.	11000111	4	3, 15, 12, 3, 2
15.	10000111	5	10, 1, 7, 3, 4

68. Що являють собою коди Хеммінга та які вони мають переваги над іншими лінійними завадостійкими кодами? Свою відповідь обґрунтуйте та наведіть доречні приклади.

69. З використанням яких співвідношень визначається розрядність коду Хеммінга?

70. Поясніть графічну залежність, наведену на рис. 2.8.

71. Поясніть спосіб формування коду Хеммінга за допомогою відповідних питань та відповідей на них, як це зроблено у підрозділі 2.4.1. Для виконання цього завдання розіграйте із своїми одногрупниками відповідну коротку сценку, в якій один студент задає другому питання про спосіб формування коду Хеммінга, а другий студент відповідає на них.

72. Поясніть властивості 2.1 та 2.2.

73. Поясніть порядок розташування інформаційних та контрольних розрядів у коді Хеммінга (12, 8) згідно із таблицями 2.2 та 2.4.

74. Поясніть спосіб формування контрольних сум для коду Хеммінга згідно із таблицею 2.3.

75. Поясніть співвідношення (2.54) – (2.57).

76. Що являє собою рівномірно захищений код? Наведіть власні приклади таких кодів.

77. Поясніть приклад 2.6, таблиці 2.5, 2.6 та 2.7, а також співвідношення (2.58) – (2.62).

78. Поясніть приклад 2.7, таблицю 2.8 та рис. 2.9.

79. Що являє собою модифікований код Хеммінга та як він формується?

80. Поясніть приклад 2.8, співвідношення (2.63), (2.64) та таблицю 2.9.

81. Поясніть приклад 2.9. Як він пов’язаний із прикладом 2.7?

82. Наведіть власні приклади побудови модифікованого коду Хеммінга.

83. Розіграйте із своїми одногрупниками коротку сценку, в якій один студент задає другому питання про спосіб формування модифікованого коду Хеммінга, а другий студент відповідає на них.

84. У чому полягає головна особливість побудови лінійних кодів, які дозволяють виправляти подвійні помилки? Наведіть власні приклади таких кодів.

85. Що являють собою синдроми одиночної та подвійної помилки та як вони визначаються?

86. Поясніть властивості 2.3 – 2.6.

87. Поясніть синдроми помилок, які наведені у таблиці 2.10.

88. Поясніть приклад 2.10, співвідношення (2.66) та таблицю 2.11.

89. Поясніть приклад 2.11 та таблицю 2.12.

90. Розіграйте із своїми одногрупниками коротку сценку, в якій один студент задає другому питання про спосіб формування лінійних кодів із виправленням подвійних помилок, а другий студент відповідає на них.

91. Які цифрові електронні пристрої, призначені для формування кодів Хеммінга, Вам відомі? Опишіть головні принципи роботи цих пристроїв.

92. Які цифрові електронні пристрої, призначені для декодування послідовностей кодів Хеммінга, Вам відомі? Опишіть принципи роботи цих пристроїв.

93. Поясніть електронні схеми, наведені на рис. 2.10 – 2.13, з точки зору логічних операцій двійкової арифметики та з точки зору теорії скінченних автоматів.

94. Розіграйте із своїми одногрупниками короткі сценки, в яких один студент задає другому питання про принципи роботи електронних схем, наведених на рис. 2.10 – 2.13, а другий студент відповідає на ці питання.

95. Що являє собою принцип мажоритарного декодування та як він використовується у декодувальній електронній апаратурі? Наведіть власні приклади використання принципу мажоритарного декодування для дешифрування послідовностей кодів Хеммінга.

96. Як принцип мажоритарного декодування пов’язаний із життєвими ситуаціями, наприклад, із обранням лідера групи або із вирішенням суперечок між членами групи? Для ілюстрації цієї аналогії між мажоритарним принципом роботи електронних кодувальних пристроїв та відповідними життєвими ситуаціями наведіть доречні приклади.

97. Розіграйте із своїми одногрупниками коротку сценку, у якій один студент поясняє другому принцип мажоритарного декодування на прикладі життєвих ситуацій, а другий студент вважає, що такі аналогії є недоречними та наводить власні аргументи.

98. Чи завжди можна розв’язати суперечливі життєві ситуації через принцип, аналогічний принципу мажоритарного декодування? Свою відповідь обґрунтуйте та наведіть доречні приклади.

99. Поясніть приклад 2.12.

100. У чому полягає особливість подання лінійних кодів через матричне перетворення? Як матричне подання лінійних кодів пов’язане із обчисленням контрольних сум? Свою відповідь обґрунтуйте та наведіть доречні приклади.

101. Що являє собою породжувальна матриця лінійного коду та який спосіб її формування? Наведіть власні приклади формування породжувальної матриці лінійного коду.

102. Що являє собою перевірочна матриця лінійного коду та який спосіб її формування? Наведіть власні приклади формування перевірочної матриці лінійного коду.

103. Поясніть співвідношення (2.68) – (2.72).

104. Поясніть приклади 2.13, 2.14 та 2.15.

105. Розіграйте із своїми одногрупниками коротку сценку, у якій один студент пояснює іншому принцип формування породжувальної та перевірочної матриці лінійного коду.

106. Поясніть головні принципи побудови комп’ютерних програм, призначених для побудови коду Хеммінга та декодування його кодових послідовностей.

107. Чому для написання програм, призначених для побудови коду Хеммінга та декодування його кодових послідовностей, зручно використовувати засоби програмування системи науково-технічних розрахунків MatLab? Для обґрунтування своєї відповіді оберіть будь-яку із операцій, пов’язаних із формуванням коду Хеммінга, та напишіть відповідний програмний код для виконання цієї операції на мові програмування системи MatLab та на мові програмування C.

108. Поясніть, чому мова програмування системи MatLab рідко

використовується на практиці для вирішення задач програмованого кодування та декодування цифрових сигналів в електронній апаратурі?

109. Організуйте диспут, під час якого студенти обговорюють можливості використання засобів програмування системи MatLab для розв'язування завдань кодування сигналів. Для цього розділіть групу на дві частини. Перша підгрупа має приводити аргументи щодо необхідності та ефективності використання такого підходу, а друга підгрупа вважає, що такий підхід не є ефективним і для розв'язування задач кодування сигналів в електроніці слід використовувати такі мови програмування, як C та Assembler. Під час проведення диспуту студенти повинні використовувати заздалегідь підготовлені комп'ютерні презентації.

110. Чи може для розв'язування проблеми диспуту, описаної у завданні 109, бути використаний принцип, аналогічний принципу мажоритарного декодування? Свою відповідь обґрунтуйте.

111. Поясніть співвідношення (2.73) – (2.75) та таблицю 2.13.

112. Поясніть, чому під час написання програм, призначених для розв'язування завдань кодування та декодування двійкових послідовностей, вкрай важливо слідкувати за порядком запису бітів числа? Із чим пов'язане використання різних способів запису чисел у двійковій арифметиці та у програмуванні?

113. Поясніть блок-схеми алгоритмів, наведених на рис. 2.14 та 2.15.

114. Поясніть принцип роботи програми, наведеної у додатку З. Чому ця програма написана за модульним принципом?

115. Написати з використанням засобів програмування системи MatLab програму, призначену для формування лінійного коду, якій виправляє подвійні помилки.

116. Написати з використанням засобів програмування системи MatLab програму, призначену для формування кодів Хеммінга через породжувальну матрицю та декодування послідовностей кодів Хеммінга через перевірочну матрицю. Для розв'язування цієї задачі використати матричні функції

системи MatLab, описані у другій частині посібника [48] та у навчальних посібниках [13, 14].

117. Написати програму, призначену для формування кодів Хеммінга та декодування їхніх кодових послідовностей з використанням засобів матричного програмування, розглянутих у підрозділі 1.2.2.

118. Написати з використанням засобів програмування системи MatLab програму, призначену для мажоритарного декодування кодів Хеммінга.

119. Побудувати звичайний та модифікований коди Хеммінга для кодових послідовностей, заданих у таблиці 2.21. Перевірити роботу сформованих кодів Хеммінга, вважаючи, що одиночні та подвійні помилки виникли у заданих розрядах. Результат розрахунків перевірити з використанням програми, наведеної у додатку З.

120. Знайти помилки у кодах Хеммінга, побудованих у завданні 119, з використанням методу мажоритарного декодування.

121. Для звичайних та модифікованих кодів Хеммінга, побудованих у завданні 119, розрахувати параметри надлишковості та побудувати спектри відповідних цифрових сигналів за умови використання коду AMI та манчестерського коду.

122. Що являють собою циклічні коди і які принципи їхнього формування Вам відомі? Наведіть власні приклади формування циклічного коду.

123. Як способи формування циклічних кодів пов’язані із теорією чисел, яка була розглянута у першому розділі другої частини посібника, та із теорією поліномів, розглянутою у третьому розділі другої частини посібника? Свою відповідь обґрунтуйте та наведіть власні приклади таких зв’язків.

124. Організувати диспут, під час якого студенти обговорюють доречність та необхідність використання інженерами теорії чисел та теорії поліномів для розв’язування практичних задач завадостійкого кодування під час проектування електронних систем. Для цього розділіть групу на дві

частини. Перша підгрупа має приводити аргументи щодо необхідності та ефективності використання методів сучасної дискретної математики для побудови завадостійких кодів. А друга підгрупа вважає, що у цьому немає необхідності. Аргументи цієї підгрупи полягають у тому, що сьогодні існують розвинені електронні та програмні засоби для кодування та декодування сигналів і інженеру взагалі не потрібно знати принципи формування кодів, які реалізовані у цих засобах, для розробки сучасного електронного обладнання. Під час проведення диспуту студенти повинні використовувати заздалегідь підготовлені комп'ютерні презентації.

Таблиця 2.21 – Початкові дані для завдань 119, 193 та 194

Номер варіанту	Число, яке кодується	Номери розрядів, у яких виникла одиночна помилка	Номери розрядів, у яких виникла подвійна помилка
1.	10111101	3	3, 12
2.	10000001	10	7, 10
3.	10101110	4	3, 4
4.	11110010	2	2, 13
5.	10110110	11	7, 11
6.	11011010	9	6, 9
7.	10110010	3	2, 3
8.	11011101	7	4, 7
9.	10101101	5	5, 11
10.	10010001	12	7, 12
11.	10010101	8	8, 13
12.	10100101	1	1, 3
13.	10011101	11	10, 11
14.	11010111	11	9, 11
15.	10110011	13	3, 13

125. Що являє собою твірна матриця циклічного коду **G**? Наведіть приклади формування цієї матриці.

126. Поясніть властивість циклічних кодів 2.7.

127. Як пов’язані числове та поліноміальне подання циклічного коду? Поясніть визначення 2.7 та співвідношення (2.76).

128. Які головні алгебраїчні операції над двійковими поліномами Вам відомі? Наведіть приклади виконання цих операцій.

129. Розіграйте із своїми одногрупниками коротку сценку, у якій один студент поясняє іншому правила виконання алгебраїчних операцій над двійковими поліномами.

130. Поясніть правила множення двійкових поліномів та наведіть приклади виконання цієї алгебраїчної операції.

131. Поясніть правила ділення двійкових поліномів та наведіть приклади виконання цієї алгебраїчної операції.

132. Поясніть співвідношення (2.77) – (2.79).

133. Що являють собою незвідні поліноми та як вони формуються?

134. Поясніть визначення 2.8 та 2.9 та таблицю 2.14, а також таблицю незвідних поліномів, наведену у додатку І.

135. Як визначається кількість розрядів у завадостійкому циклічному коді? Поясніть співвідношення (2.80) та таблицю 2.15.

136. Поясніть спосіб формування несистематичних циклічних кодів та алгоритм їхнього декодування, який наведений на рис. 2.16.

137. Розіграйте із своїми одногрупниками коротку сценку, у якій один студент поясняє іншому алгоритм виправлення помилок у несистематичному циклічному коді.

138. Поясніть властивість 2.8 та визначення 2.10, 2.11.

139. Що являє собою ідеал у теорії поліномів і як він визначається? Наведіть приклади ідеалів.

140. Що являє собою породжувальний поліном ідеалу та як він

визначається? Наведіть приклади породжувальних поліномів.

141. Поясніть теореми 2.3 – 2.5.

142. Поясніть приклад 2.16 та рис. 2.17 – 2.24.

143. Які існують головні правила вибору незвідних твірних поліномів для формування завадостійких кодів? Наведіть власні приклади обрання твірного поліному.

144. Поясніть співвідношення (2.83), (2.84).

145. Як формується циклічний код, призначений для виявлення одиночної помилки? Поясніть приклад 2.17 та рис. 2.25, 2.26.

146. Наведіть власний приклад формування циклічного коду, призначеного для виявлення одиночних помилок.

147. Поясніть співвідношення (2.85), (2.86), та числові дані, які наведені у таблиці 2.16.

148. Поясніть властивості 2.9 – 2.12.

149. Поясніть приклад 2.18 та рис. 2.27 – 2.29.

150. Поясніть приклад 2.19, співвідношення (2.87) – (2.89) та рис. 2.30.

151. Поясніть приклад 2.20 та рис. 2.31.

152. Поясніть визначення 2.12, властивість 2.13 та таблицю 2.17.

153. Що являють собою належні поліноми та як вони визначаються?

154. Розіграйте із своїми одногрупниками коротку сценку, у якій один студент поясняє іншому спосіб визначення належних поліномів.

155. Поясніть співвідношення (2.90), (2.91).

156. Поясніть приклад 2.21 та рис. 2.32.

157. Поясніть приклад 2.22, співвідношення (2.94) – (2.96) та рис. 2.33, 2.34.

158. Поясніть визначення 2.13, 2.14, теорему 2.6 та співвідношення (2.96) – (2.99).

159. У чому полягає головний недолік несистематичних циклічних кодів, через який вони рідко використовуються у сучасних кодувальних електронних системах? Свою відповідь обґрунтуйте та підтвердить доречними прикладами.

160. Поясніть спосіб формування систематичного циклічного коду та наведіть власні приклади формування таких кодів.

161. Поясніть приклад 2.23 та рис. 2.35, 2.36.

162. Які алгоритми пошуку помилки у систематичному циклічному коді Вам відомі? Поясніть блок-схему алгоритму, наведену на рис. 2.37.

163. Поясніть приклад 2.24 та рис. 2.38.

164. Поясніть спосіб формування систематичного циклічного коду, оснований на визначені контрольних сум з використанням співвідношень (2.100), (2.101).

165. Поясніть приклад 2.25, рис. 2.39, співвідношення (2.102) та таблицю 2.18.

166. Організувати диспут, у якому студенти обговорюють переваги та недоліки систематичних та несистематичних циклічних кодів. Для цього розділіть групу на дві частини. Перша підгрупа має приводити аргументи щодо переваг використання несистематичних кодів, а друга – систематичних. Під час проведення диспуту студенти повинні використовувати заздалегідь підготовлені комп'ютерні презентації.

167. У чому полягає особливість матричного подання циклічних кодів? Наведіть власні приклади такого подання.

168. Що являє собою матриця доповнення та як вона формується?

169. Поясніть співвідношення (2.103).

170. Що являє собою твірна матриця та як вона формується?

171. Що являють собою скорочені циклічні коди та як вони формуються?

172. Поясніть визначення 2.15 та властивості 2.14, 2.15.

173. Розіграйте із своїми одногрупниками коротку сценку, у якій один студент поясняє іншому спосіб формування циклічних кодів через твірну матрицю.

174. Розіграйте із своїми одногрупниками коротку сценку, у якій один студент поясняє іншому спосіб формування скорочених циклічних кодів.

175. Організувати диспут, під час якого студенти обговорюють

переваги та недоліки циклічних та лінійних кодів. Для цього розділіть групу на дві частини. Перша підгрупа має приводити аргументи щодо переваг циклічних кодів, а друга – лінійних. Під час проведення диспуту студенти повинні використовувати заздалегідь підготовлені комп'ютерні презентації.

176. Поясніть принципи роботи схем множення поліномів, які наведені на рис. 2.41, а, б. У чому полягає відмінність цих двох схем?

177. Поясніть принцип роботи схеми, наведеної на рис. 2.42.

178. Поясніть принципи роботи схем поліноміального ділення, наведених на рис. 2.43 та 2.44.

179. У чому полягає головна відмінність кодувальних електронних схем, призначених для формування несистематичних та систематичних циклічних кодів? Свою відповідь обґрунтуйте та наведіть приклади таких схем.

180. Поясніть принципи роботи кодувальних схем, наведених на рис. 2.45 та 2.46.

181. Поясніть узагальнену структурну схему декодера циклічних кодів, яка наведена на рис. 2.47.

182. Поясніть блок-схему алгоритму роботи декодера циклічних кодів, яка наведена на рис. 2.48.

183. Поясніть приклад 2.26, рис. 2.49, 2.50 та таблицю 2.18.

184. Як в електронних схемах, призначених для декодування циклічних кодів, використовується принцип мажоритарного декодування? Наведіть власні приклади такого використання.

185. Поясніть приклад 2.27, співвідношення (2.105), (2.106), рис. 2.51, 2.52 та таблицю 2.19.

186 Розіграйте із своїми одногрупниками коротку сценку, у якій один студент поясняє іншому особливості використання принципу мажоритарного декодування для дешифрування послідовностей циклічних кодів.

187. Поясніть електронні схеми, наведені на рис. 2.41 – 2.52, з точки зору логічних операцій двійкової арифметики та з точки зору теорії

скінченних автоматів.

188. Розіграйте із своїми одногрупниками короткі сценки, в яких один студент задає другому питання про принципи роботи електронних схем, наведених на рис. 2.41 – 2.52, а другий студент відповідає на ці питання.

189. Поясніть структуру програми, призначеної для формування циклічних кодів та декодування їхніх кодових послідовностей, яка написана на мові програмування системи MatLab та наведена у додатку К.

190. Поясніть алгоритм формування матриці множення, який наведений на рис. 2.53 та приклад 2.28.

191. Поясніть алгоритм формування матриці ділення, який наведений на рис. 2.54 та приклад 2.29.

192. Організувати диспут, під час якого студенти обговорюють переваги та недоліки матричних алгоритмів множення та ділення двійкових послідовностей. Для цього розділіть групу на дві частини. Перша підгрупа має приводити аргументи щодо переваг використання матричних алгоритмів, а представники другої вважають, що зручніше виконувати ці операції безпосередньо над числами, без використання матриць. Під час проведення диспуту студенти повинні використовувати заздалегідь підготовлені комп’ютерні презентації.

193. Побудувати систематичні та несистематичні циклічні коди із виправленням однієї помилки для інформаційних слів, наведених у таблиці 2.21. Внести помилку у відповідний розряд та перевірити коректність роботи побудованого коду. Результати розрахунків перевірити з використанням програми, наведеної у додатку К.

194. Для інформаційних слів, наведених у таблиці 2.21, побудувати систематичний циклічний код, який виправляє одиночні та виявляє подвійні помилки. Перевірити коректність роботи коду для випадків виникнення одиночної та подвійної помилки.

195. Для систематичних та несистематичних циклічних кодів, побудованих у завданнях 193 та 194, розрахувати параметри надлишковості

та побудувати спектри відповідних цифрових сигналів за умови використання коду AMI та манчестерського коду.

196. З використанням мови програмування системи MatLab написати комп'ютерну програму, призначену для формування та декодування циклічних кодів, які виправляють одиночні та виявляють подвійні помилки. За допомогою написаної програми перевірити результати розрахунків, виконаних у завданні 194.

197. З використанням мови програмування системи MatLab написати комп'ютерну програму, призначену для формування та декодування циклічних кодів через рекурентні співвідношення (2.100) та (2.101).

198. З використанням мови програмування системи MatLab написати комп'ютерну програму, призначену для декодування послідовностей циклічних кодів за допомогою принципу мажоритарного декодування.

199. З використанням мови програмування системи MatLab написати комп'ютерну програму з графічним інтерфейсом користувача, який дозволяв би для заданого вхідного інформаційного слова або кодової послідовності у спливаючому вікні обирати відповідні твірні поліноми з першого до восьмого порядку. Програма повинна формувати циклічні коди для інформаційних слів довжиною від 3 до 247 бітів та розв'язувати задачу декодування циклічних кодів для кодових послідовностей довжиною від 7 до 255 бітів. Способи кодування та деконування також повинні обиратися користувачем через спливаючому вікно інтерфейсу.

Перелік посилань

1. Денбновецький С.В., Мельник І.В., Писаренко Л.Д. Кодування сигналів в електронних системах. Частина 1. Параметри сигналів та каналів зв'язку та методи їхнього оцінювання. Навчальний посібник для студентів-бакалаврів напрямку підготовки 6.0508 «Електроніка» – К.: Кафедра. – 524 с.

2. Кузьмин И.В., Кедрус В.А. Основы теории информации и кодирования. – К.: Вища школа, 1977. – 238 с.

3. Пеннин П.И., Филиппов Л.И. Радиотехнические системы передачи информации. – М.: Радио и связь, 1984. – 256 с.

4. Цымбыл В.П. Основы теории информации и кодирования. – К.: Вища школа, 1992. – 294 с.

5. Дмитриев В.И. Прикладная теория информации. – М.: Высшая школа, 1989. – 320 с.

6. Панин В.В. Основы теории информации. – М.: БИНОМ, 2007. – 436 с.

7. Шеннон К. Работы по теории информации и кибернетике. – М: Издательство иностранной литературы, 1963. – 830 с.

8. Денбновецький С.В., Лещишин О.В. Електронні системи: навчальний посібник. – К.: НТУУ „КПІ", 2011. – 288 с.

9. Ширяев А.Н. Вероятность. Учебное пособие для студентов вузов. – М.: Наука, 1989. – 640 с.

10. Орлов В.А., Филиппов Л.И. Теория информации в упражнениях и задачах. – М.: Высшая школа, 1976. – 136 с.

11. Самарский А.А., Гулин А.В. Численные методы. – М.: Наука, 1989. – 432 с.

12. Денбновецкий С.В., Писаренко Л.Д., Резниченко В.К. Основы автоматизированного проектирования электронных приборов. К.: Выща школа, 1987. – 335 с.

13. Мельник І.В. Система науково-технічних розрахунків MatLab та її використання для розв'язання задач із електроніки: навчальний посібник у 2-х томах. Т. 1. Основи роботи та функції системи. – К.: Університет «Україна», 2009. – 507 с.

14. Мельник І.В. Система науково-технічних розрахунків MatLab та її використання для розв'язання задач із електроніки: навчальний посібник у 2-х томах. Т. 2. Основи програмування та розв'язання прикладних задач. – К.: Університет «Україна», 2009. – 327 с.

15. Лунтовський А.О., Мельник І.В. Проектування та дослідження комп'ютерних мереж: навчальний посібник. – К.: Університет «Україна», 2010. – 361 с.

16. Лунтовський А.О., Мельник І.В. Основи проектування безпроводових комп'ютерних мереж: навч. посібник. – К.: Освіта України, 2011. – 362 с.

17. Мельник І.В. Інформаційні комп'ютерні мережі: Навчальний посібник для дистанційного навчання. – К.: Університет «Україна», 2006. – 250 с.

18. Мельник И.В., Гадимов Г.И. Основы построения компьютерных сетей. – Баку, Издательство «Элм», 2013. – 480 с.

19. Компьютерные сети. Принципы, технологи, протоколы. Учебник для ВУЗов. / В.Г. Олифер, Н.А. Олифер. – СПб.: Питер, 2004. – 864 с.

20. Таненбаум Э. Компьютерные сети. – BHV, СПб, 2003. – 992 с.

21. Лунтовський А.О. Технології розподілених програмних додатків. Монографія. К.: Державний університет інформаційно-комунікаційних технологій ДУІКТ, 2010. – 452 с.

22. Лунтовський А.О., Климаш М.М., Семенко А.І. Розподілені сервіси телекомунікаційних мереж та повсякденний комп'ютінг і Cloud-технології. Монографія. – Львів: Національний університет «Львівська політехніка», 2012. – 368 с.

23. Лунтовський А.О., Климаш М.М. Інформаційна безпека

розподілених систем. Монографія. – Львів: Національний університет «Львівська політехніка», 2014. – 444 с.

24. Бабак В.П., Наритник Т.М., Куц Ю.В., Казмиренко В.Я. Обробка сигналів у радіоканалах цифрових систем передавання інформації. / За загальною редакцією член.-кор. НАН України В.П. Бабака. – К.: Книжкове видавництво НАНУ, 2005. – 476 с.

25. Сигорский В.П. Математический аппарат инженера. – К.: Тэхника, 1977. – 768 с.

26. Денисенко А.Н. Сигналы. Теоретическая радиотехника. Справочное пособие. – М.: Горячая линия – Телеком, 2006. – 704 с.

27. Фигурин В.А., Оболонкин В.В. Теория вероятностей и математическая статистика. Учебное пособие. – Минск: ООО «Новое знание», 2000. – 208 с.

28. Ипатов В. Широкополосные системы и кодовое разделение сигналов. Принципы и приложения. – М.: Техносфера, 2007. – 488 с.

29. Вишневский В.М., Ляхов А.И., Портной С.Л., Шахнович И.В. Широкополосные беспроводные сети передачи информации. – М.: Техносфера, 2005. – 592 с.

30. Ирвин Дж., Харль Д. Передача данных в сетях: инженерный подход: Перевод с английского. – СПБ: БХВ, 2003. – 448 с.

31. Тарасенко В.П., Маламан А.Ю., Черниченко Ю.П., Корнійчук В.І. Надійність комп'ютерних систем. – К: Корнійчук, 2007. – 256 с.

32. Домбругов Р.М. Телевидение. – К.: Вища школа, 1998. – 214 с.

33. Скляр Б. Цифровая связь. Теоретические основы и практическое применение. – М.: Издательский дом «Вильямс», 2003. – 1104 с.

34. Багатоканальний зв'язок та телекомунікаційні технології: Підручник для студентів вищих навчальних закладів / За редакцією Поповського В.В. – Харків: Компаня «Сміт», 2003. – 512 с.

35. Гельдман М.М. Аналого-цифровые преобразователи для информационно-измерительных систем. - М.: Изд-во стандартов. 1989. – 320 с.

36. Федорков Б.Г., Телец В.А. "Микросхемы ЦАП и АЦП: функционирование, параметры, применение". – М.: Энергоатомиздат, 1990. – 320 с.

37. Тимченко О.В. Методи різницевого кодування форми сигналів в системах передачі мовної інформації. – Львів: Видавництво Української академії друкарства, 2006. – 320 с.

38. Банкет В.Л., Иващенко П.В., Геер А.Э. Цифровые методы передачи информации в спутниковых системах связи. Учебное пособие. – Одесса, УГАС, 1996. – 180 с.

39. Джордейн Р. Справочник программиста персональных компьютеров типа IBM PC, XT и AT. – М.: Финансы и статистика, 1992. – 544 с.

40. Абель П. Язык ассемблера для IBM PC и программирования. – М.: Высшая школа, 1992. - 447 с.

41. http://www.cisco.com/

42. http://www.dsplib.ru/content/bpsk/bpsk.html

43. Солодов А.В. Теория информации и её применение в задачах оптимального контроля. – М.: Наука, 1967. – 534 с.

44. Каган Б.М. Электронные вычислительные машины и системы. Учебное пособие для вузов. – М.: Энергоатомиздат, 1991. – 592 с.

45. Корнев В.В. Параллельные вычислительные системы. – М.: Нолидж, 1999. – 320 с.

46. Корнейчук В.И., Тарасенко В.П. Основы компьютерной арифметики. – Киев, «Корнійчук», 2003. – 176 с.

47. Столингс В. Беспроводные линии связи и сети. – М.: Издательский дом «Вильямс», 2003. – 640 с.

48. Денбновецький С.В., Мельник І.В., Писаренко Л.Д. Кодування сигналів в електронних системах. Частина 2. Математичні основи теорії кодування. Навчальний посібник для студентів-бакалаврів напрямку підготовки 6.0508 «Електроніка». Книга 1. – К.: Кафедра. – 724 с.

49. Денбновецький С.В., Мельник І.В., Писаренко Л.Д. Кодування сигналів в електронних системах. Частина 2. Математичні основи теорії кодування. Навчальний посібник для студентів-бакалаврів напрямку підготовки 6.0508 «Електроніка». Книга 2. – К.: Кафедра. – 432 с.

50. Денбновецький С.В., Мельник І.В., Писаренко Л.Д. Кодування сигналів в електронних системах. Частина 2. Математичні основи теорії кодування. Навчальний посібник для студентів-бакалаврів напрямку підготовки 6.0508 «Електроніка». Книга 3. – К.: Кафедра. – 368 с.

51. Бронштейн И.Н., Семендяев К.А. Справочник по математике. Для инженеров и учащихся втузов. – М.: «Наука», Главная редакция физико-математической литературы, 1986. – 723 с.

52. Берлекэмп Э. Алгебраическая теория кодирования. – М.: Мир, 1971. – 480 с.

53. Биркгоф Г., Барти Т. Современная прикладная алгебра. – М.: Мир, 1976. – 400 с.

54. Андерсон Дж. Дискретная математика и комбинаторика. – М.: Издательский дом «Вильямс», 2004. – 960 с.

55. Вступ до алгебраїчної теорії перешкодостійкого кодування. / Волкович С.Л., Геранін В.О., Мовчан Т.В., Писаренко Л.Д., Розарінов Г.М., Хотяїнцев С.М.: Науково-методичне видання. – Київ, ВПФ УкрІНТЕІ, 2002. – 236 с.

56. Олексенко П.Ф., Коваль В.В., Розарінов Г.М., Сукач Г.О. Теоретичні основи завадостійкого кодування. Частина І. Підручник для вищих навчальних закладів за редакцією академіка НАН України В.Ф. Мачуліна. – К.: Наукова думка, 2010. – 192 с.

57. Олексенко П.Ф., Коваль В.В., Розарінов Г.М., Сукач Г.О. Теоретичні основи завадостійкого кодування. Частина ІІ. Підручник для вищих навчальних закладів за редакцією академіка НАН України В.Ф. Мачуліна. – К.: Наукова думка, 2012. – 212 с.

58. Основи теорії телекомунікацій: підручник / О.В. Корнейко, О.В. Кувшинов, О.П. Лежнюк [та ін.]; за заг. ред. М.Ю. Ільченка. – К.: Видавництво ІСЗЗІ НТУУ «КПІ», 2010. – 786 с.

59. Кнут Д. Искусство программирования. Том 3. Сортировка и поиск. – М.: «Вильямс», 2007. — 824 с.

60. Варакин Л.Е. Системы связи с шумоподобными сигналами. – М.: радио и связь, 1985. – 380 с.

61. https://ua-mova.livejournal.com/885374.html#/885374.html#

62. Питерсон У., Уэлдон Э. Коды, исправляющие ошибки. – М.: Советское радио, 1986. – 593 с.

63. Блейхут Р. Теория и практика кодов, контролирующих ошибки: пер. с англ. – М.: Мир, 1986. – 576 с.

64. Фирсов В.А. Теория информации. – Самара, Издательство самарского государственного аэрокосмического университета, 2011. – 128 с.

65. Прохоров В.С. Теория информации. Лекции. – Електронний ресурс. Режим доступу: http://profbeckman.narod.ru/Informat.files/Teorinf.pdf

66. Сукачев Э.А., Бухан Д.Ю. Корреляционный анализ детерминированных сигналов. Компьютеризированый курс. Монография. – Одесса, 2014. – 134 с.

67. Бородин Л.Ф. Введение в теорию помехоустойчивого кодирования. – М.: Советское радио, 1968. – 408 с.

68. Теория электрической связи: учебное пособие / К.К. Васильев, В.А. Глушков, А.В. Дормидонтов, А.Г. Нестеренко. Под общ. ред. К.К. Васильева. – Ульяновск: УлГТУ, 2008. – 452 с.

69. Финк Л.М. Теория передачи дискретных сигналов. – М.: "Советское радио", 1970. – 728 с.

70. Вернер М. Основы кодирования. — М.: Техносфера, 2004. – 288 с.

70. http://informkod.narod.ru/5_5item.htm

71. http://yourtutor.narod.ru/cyclic/CyclicCodes.htm

72. http://dvo.sut.ru/libr/opds/i285vino/2.htm

73. http://book.itep.ru/2/28/corec_28.htm

74. http://stor.boom.ru/uch/din/1/15/153.htm

75. Мельник І.В. Лабораторний парктикум з курсу «Проектування інформаційних електронних систем та мереж». Електронний навчальний посібник для студенів-спеціалістів V курсу факультета електроніки спеціальності електронні прилади та пристрої. – 166 с.

76. Дьяконов В.П. Simulink 5/6/7: Самоучитель. – М.: ДМК – Пресс. – 2008. – 784 с. с.

77. Дьяконов В.П., Пеньков А.А. MatLab и Simulink в электроэнергетике. Справочник. – М.: Горячая линия – Телеком. – 2009. – 816

ДОДАТОК А. Програма для розрахунку імовірності помилки сигналу із фазовою модуляцією

```
function ErPhase
    EbN0=0.001:0.01:10;
    p=(1-erf(sqrt(2)*EbN0))/2;
    plot(log10(EbN0),p);
    grid on;
return
```

ДОДАТОК Б. Програма для формування диференціальних кодів та їхнього декодування

```
%Fuctioon for calculation the sequences
%of difference coding and for decoding
%of difference codes. Input favameters:
%vin - binary sequence. Only value 0 and 1
%for vector vin is allows.
%nop - number of necessary operation.
%Possible values of this parameter:
%1 - forming of difference code.
%In this case output vector is coded
%coded bynary sequence.
%2 - decoding of coded sequence.
%In this case output vector is
%decoded binary sequence.
%Example of using this function
%>> c=diffcode([0,1,1,1,0,0,1,0,1],1)
%c =
%     0     1     0     1     1     1     0     0     1
%>> d=diffcode(c,2)
%d =
%     0     1     1     1     0     0     1     0     1
%>>
function vout=diffcode (vin,nop)
    n=length(vin);
    dd1='Wrong input vector! Bits of vector have t';
    dd2='o be equal 0 or 1. Check input data!';
    ddstr=[dd1,dd2];
    for ii=1:n
```

```
        if(~((vin(ii)==1)|(vin(ii)==0)))
           error (ddstr); end;
        end;
    rtr1='Wrong number of operation nop. This valu';
    rtr2='e can be equal 1 for coding task or 2 f';
    rtr3='or decoding task';
    ddstr2=[rtr1,rtr2,rtr3];
    if(~((nop==1)|(nop==2))) error (ddstr2); end;
    if(nop==1)
        ffh1='(ii==1).*v(1)+(ii==2).*v(2)';
        ffh2='+(ii>2).*xor(w(ii-1),v(ii))';
        ffh=[ffh1,ffh2];
        vdiff=recvectbin(2,n,vin,ffh);
        vout=vdiff;
    end;
    if(nop==2)
        ffhh1='(ii==1).*v(1)+(ii==2).*xor(v(1),v(2))+';
        ffhh2='(ii>2).*xor(v(ii-1),v(ii))';
        ffhh=[ffhh1,ffhh2];
        vdescr=recvectbin(2,n,vin,ffhh);
        vout=vdescr;
    end;
return

%Function recvectbin(nr,n,v,ff) calculated
%the elements of vector by the recurrent formula,
%defined by expression:
%w(i)=ff(w(i-1),...,w(i-nr),i).
%Function parameters:
%nr - number of pervious recurrent elements;
%n - maximum necessary number of vector's elements;
```

```
%v - vector of initial values, w(1), w(2)...w(j)
%ff - function, defined by string.
%You can use both complete logical expression with
%brackets or the name of external
%function, like '(ii==5).*(w(ii-1))' or 'ssd',
%where 'ssd' is the name of function.
%If you use only name of function, please keep such
%rules:
%1. Maximum number of pervious recurrent
%elements nr is 10.
%2. Required sequence of function parameters is:
%w(ii-1),...,w(ii-nr),ii
%When you defined string expression, please keep the
%following
%rules for variables designation:
%v - input vector;
%w - output vector;
%ii - index counter.
%Some examples of using recvectbin function.
%ffhh1='(ii==1).*v(1)+(ii==2).*v(2)+(ii==3).*v(3)+';
%ffhh2='(ii==4).*mod3(v(4),v(1),v(1))+(ii==5).*';
%ffhh3='mod3(v(5),v(2),v(2))+(ii>5).*mod32(v(ii),';
%ffhh4='v(ii-3),v(ii-5))';
%ffhh=[ffhh1,ffhh2,ffhh3,ffhh4];
%vdescr=recvectbin(5,n,vin,ffhh);
%See also recvect, recmat
%Written by Igor Melnyk, National Technical University
%of Ukraine "Igor Sikorskiy KPI", Electron Devices
%Dept., November, 2017
function w=recvectbin (nr,n,v,ff)
if (nr<0) error ('Error, number of recurrent...
             elements nr can not be negative'), end;
if (n<0) error ('Error, number of iterations n can...
```

```
            not be negative'), end;
if (nr>n) error ('Error, number of recurrent...
        elements nr is smaller, than maximum...
            numbers of vectors elements n '), end;
if (nr>length(v)) error ('Error, number of elements...
        of input vector v is smaller,...
        then nr - number of recurrent elements. ...
        Please check input vector v'), end;
if (n<length(v)) error ('Error, number of ...
            elements of input vector v ...
            is larger, than maximum necessary...
            number of vector elements...
            n. Please check input vector v'), end;
nn=1; c1=0; c2=0; s1=0; s2=0; ch=0;
for iii=1:length(ff)
   if (ff(iii)=='(' ) nn=2; c1=c1+1;   end;
   if (ff(iii)==')' ) c2=c2+1;   end;
   if (ff(iii)=='[' ) nn=2; s1=s1+1; ch=1;   end;
   if (ff(iii)==']' ) s2=s2+1;   end;
end;
if((nn==2)&(c1~=c2)) error ('Check...
    expression. Numbers...
        of round brackets are unequal!'), end;
if((nn==2)&(s1~=s2)) error ('Check expression. ...
    Numbers of square brackets are unequal!'), end;
if (nn==2)
   for ii=1:n
      if (ii<=nr)
         fg=ff; lff=length(fg);
         ggstr=num2str(ii+1);
         if (ii<nr) qw=['+(ii==',ggstr,')'];
                shift=7;
        end;
```

```
   if (ii==nr) qw=['+(ii>',ggstr-1,')'];
            shift=6;
  end;
   for gg=8:1ff
     if (qw==fg(gg-shift:gg))
       fg=fg(1:gg-shift-1);
       break;
    end;
   end;
 end;
 if (ii>nr) fg=ff; end;
 if (ch==0) w(ii)=eval(fg); end;
 if ((ch==1)&(nr==1))
       res=eval(fg); w(ii)=res(2);
       w(ii-1)=res(1);
end;
 if ((ch==1)&(nr==2))
       res=eval(fg); w(ii)=res(3);
       w(ii-1)=res(2); w(ii-2)=res(1);
end;
 if ((ch==1)&(nr==3))
       res=eval(fg); w(ii)=res(4);
       w(ii-1)=res(3); w(ii-2)=res(2);
       w(ii-3)=res(1);
end;
 if ((ch==1)&(nr==4))
       res=eval(fg); w(ii)=res(5);
       w(ii-1)=res(4); w(ii-2)=res(3);
       w(ii-3)=res(2); w(ii-4)=res(1);
end;
 if ((ch==1)&(nr==5))
  res=eval(fg); w(ii)=res(6);
  w(ii-1)=res(5); w(ii-2)=res(4);
  w(ii-3)=res(3); w(ii-4)=res(2);
```

```
        w(ii-5)=res(1);
        end;
    end;
end;
return;
```

Результати тестування програми

```
>> c=diffcode([0,1,1,1,0,0,1,0,1],1)
c =
     0     1     0     1     1     1     0     0     1
>> c=diffcode(c,2)
c =
     0     1     1     1     0     0     1     0     1
>>
```

ДОДАТОК В. Програма для моделювання часових залежностей двійкових сигналів з різною формою кодування та аналізу їхніх спектрів

```
%Function for forming of signals
%with different methods of channel coding
%by defined binary sequence. Different
%time dependences of signals with unipolar,
%bipolar and pulse coding can be obtained
%and analyzed. With using additional function
%SpectrFourier analyzing of signals frequency
%band is also possible. The time and frequency
%dependences are presented in graphic form.
% Input parameters for this function.
% vin - input vector of bytes sequence.
% ns - integer value for describing the type
%of signal in channel coding. Such values of
%this parameters are possible.
%1 - Bipolar code with short-time
% synchronized impulses.
%2 - Unipolar code without return to zero
%3 - Bipolar Code without return to zero with inversion
%4 - Manchester code
%5 - Code 2B/1Q
%Examples of using this function.
%>> digspectrum([1,0,0,0,1],3)
%>> digspectrum([1,0,0,0,1],4)
%See also SpectrFourier
function digspectrum(vin,ns)
tdig=1e-6; tpause=2e-7; timp=1e-7;
```

```
    stept=tdig/50;
    Tout=[]; Uout=[]; n=length(vin);
    dd1='Wrong input vector! Bits of vector have t';
    dd2='o be equal 0 or 1. Check input data!';
    ddstr=[dd1,dd2];
    for ii=1:n
        if(~((vin(ii)==1)|(vin(ii)==0)))
           error (ddstr); end;
        end;
    rtr1='Wrong number of operation nop. This valu';
    rtr2='e can be equal 1, 2, 3, 4 or 5.';
    ddstr2=[rtr1,rtr2];
    if(~((ns==1)|(ns==2)|(ns==3)|(ns==4)|(ns==5)))
        error (ddstr2); end;
if(ns==1)
    U(2)=1; U(1)=0;
    for ii=1:n
        tau1=(ii-1).*tdig:stept:ii.*tdig-tpause;
        rt1=length(tau1);
        Uout1(1:rt1)=U(vin(ii)+1);
        tau2=ii.*tdig-tpause+stept:stept:ii.*tdig-timp;
        rt2=length(tau2);
        Uout2(1:rt2)=0;
        tau3=ii.*tdig-timp+stept:stept:ii.*tdig-stept;
        rt3=length(tau3);
        Uout3(1:rt3)=-1;
        Tper=[tau1,tau2,tau3];
        Uper=[Uout1,Uout2,Uout3];
        Tout=[Tout,Tper];
        Uout=[Uout,Uper];
```

```
    end; % End of for  ii=1:n
end; % End of if (ns==1)
if(ns==2)
    U(2)=1; U(1)=0;
    for ii=1:n
        tau1=(ii-1).*tdig:stept:ii.*tdig-stept;
        rt1=length(tau1);
        Uout1(1:rt1)=U(vin(ii)+1);
        Tout=[Tout,tau1];
        Uout=[Uout,Uout1];
    end; % End of for  ii=1:n
end; % End of if (ns==2)
if(ns==3)
    U(2)=-1; U(1)=0;
    for ii=1:n
        tau1=(ii-1).*tdig:stept:ii.*tdig-stept;
        rt1=length(tau1);
        if(vin(ii)==1)U(2)=-U(2); end;
        Uout1(1:rt1)=U(vin(ii)+1);
        Tout=[Tout,tau1];
        Uout=[Uout,Uout1];
    end; % End of for  ii=1:n
end; % End of if (ns==3)
if(ns==4)
    U(2)=1; U(1)=-1;
    stept=stept./2;
    trt=[];
    for kk=1:n
        if (vin(kk)==1) dr=[0,1]; trt=[trt,dr]; end
        if (vin(kk)==0) dr=[1,0]; trt=[trt,dr]; end
```

```
    end; % End of for  ii=1:n
    for ii=1:2.*n
        tau1=(ii-1).*tdig:stept:ii.*tdig-stept;
        rt1=length(tau1);
        Uout1(1:rt1)=U(trt(ii)+1);
        Tout=[Tout,tau1];
        Uout=[Uout,Uout1];
    end; % End of for  ii=1:n
end; % End of if (ns==4)
if(ns==5)
    U(1)=-2.5; U(2)=-0.833;
    U(3)=2.5;  U(4)=0.833;
    trt1=[]; trt2=[];
    if (mod(n,2)==1)
        trt1=[vin(1:n-1),0,vin(n)];
        n=n+1;
      else trt1=vin;
    end;
    stept=2.*stept;
    for kk=1:2:n-1
        if (trt1(kk)==0)&(trt1(kk+1)==0) dr=1; end;
        if (trt1(kk)==0)&(trt1(kk+1)==1) dr=2; end;
        if (trt1(kk)==1)&(trt1(kk+1)==0) dr=3; end;
        if (trt1(kk)==1)&(trt1(kk+1)==1) dr=4; end;
        trt2=[trt2,dr];
    end; % End of for  ii=1:n
    for ii=1:n/2
        tau1=(ii-1).*tdig:stept:ii.*tdig-stept;
        rt1=length(tau1);
        Uout1(1:rt1)=U(trt2(ii));
```

```
        Tout=[Tout,tau1];
        Uout=[Uout,Uout1];
    end; % End of for  ii=1:n
end; % End of if (ns==5)
ch1=plot(Tout,Uout);
grid on;
figure
SpectrFourier(Uout,Tout);
return

%Function for calculation of frequency band of
%digital signals with using analytical solution
%for Fourier integral. Input parameters:
%Trange - discrete vector of fixed  time points.
%Amp - vector of signal's amplitude in fixed points.
%Range of thtis vectors have to be same for
%providing correct calculations.
%For obtinnig dependence of digital signals
%amplitude verse the time function digspectrum
%can me used.
%See also digspectrum
function SpectrFourier(Amp,Trange)
    w=10:100:2e7;
    ninp=length(Trange);
    ninp2=length(Amp);
    drr1='Wrong input data. Range of amplitude and ti';
    drr2='me vectors is not equal. Check input data.';
    drr=[drr1,drr2];
    if (ninp~=ninp2) error (drr); end;
    Us(1)=Amp(1); Ts(1)=0; nchg=2;
```

```
    for ii=2:ninp
        if(Amp(ii)~=Amp(ii-1))
            Us(nchg)=Amp(ii);
            Ts(nchg)=Trange(ii);
            nchg=nchg+1;
        end;
    end;
    nfi=length(Ts);
    nw=length(w); Acos=zeros(1,nw); Asin=zeros(1,nw);
    for kk=2:nfi
        Acos=Acos+(Us(kk-1)).*(sin(w.*Ts(kk))-...
                    sin(w.*Ts(kk-1)));
        Asin=Asin+(Us(kk-1)).*(cos(w.*Ts(kk-1))-...
                    cos(w.*Ts(kk)));
    end;
    Acos=Acos./w; Asin=Asin./w;
    A=sqrt((Acos).^2+(Asin).^2);
    ch2=plot(w,A);
    grid on;
    hold on;
return
```

ДОДАТОК Г. Таблиця кодів ASCII

Таблиця Д1

DEC	HEX	Символ	DEC	HEX	Символ	DEC	HEX	Символ	DEC	HEX	Символ
000	00H	Нуль	032	20H	sp	064	40H	@	096	60H	`
001	01H	Початок заголовку	033	21H	!	065	41H	A	097	61H	A
002	02H	Початок тексту	034	22H	«	066	42H	B	098	62H	B
003	03H	Кінець тексту	035	23H	#	067	43H	C	099	63H	C
004	04H	Кінець передавання даних	036	24H	$	068	44H	D	100	64H	D
005	05H	Запит	037	25H	%	069	45H	E	101	65H	e
006	06H	Так	038	26H	&	070	46H	F	102	66H	f
007	07H	Дзвоник	039	27H	‘	071	47H	G	103	67H	g
008	08H	Повернення на крок	040	28H	(	072	48H	H	104	68H	h
009	09H	Горизонтальна табуляція	041	29H	)	073	49H	I	105	69H	i
010	0AH	Переведення рядку	042	2AH	*	074	4AH	J	106	6AH	j
011	0BH	Вертикальна табуляція	043	2BH	+	075	4BH	K	107	6BH	k
012	0CH	Переведення сторінки	044	2CH	,	076	4CH	L	108	6CH	l
013	0DH	Повернення каретки	045	2DH	-	077	4DH	M	109	6DH	m
014	0EH	Shift out	046	2EH	.	078	4EH	N	110	6EH	n
015	0FH	Shift in	047	2FH	/	079	4FH	O	111	6FH	o
016	10H	Data line ESC	048	30H	0	080	50H	P	112	70H	p
017	11H	Керування 1	049	31H	1	081	51H	Q	113	71H	q
018	12H	Керування 2	050	32H	2	082	52H	R	114	72H	r
019	13H	Керування 3	051	33H	3	083	53H	S	115	73H	s
020	14H	Керування 4	052	34H	4	084	54H	T	116	74H	t
021	15H	Ні	053	35H	5	085	55H	U	117	75H	U
022	16H	Синхронизація	054	36H	6	086	56H	V	118	76H	V
023	17H	Кінець блоку	055	37H	7	087	57H	W	119	77H	W
024	18H	Анулювання	056	38H	8	088	58H	X	120	78H	X
025	19H	End of medium	057	39H	9	089	59H	Y	121	79H	Y
026	1AH	Заміна	058	3AH	:	090	5AH	Z	122	7AH	Z
027	1BH	Escape	059	3BH	;	091	5BH	[	123	7BH	{
028	1CH	Розділення файлу	060	3CH	<	092	5CH	\	124	7CH	|
029	1DH	Розділення групи	061	3DH	=	093	5DH	]	125	7DH	}
030	1EH	Розділення запису	062	3EH	>	094	5EH	^	126	7EH	~
031	1FH	Розділення одиниці	063	3FH	?	095	5FH	_	127	7FH	Backspace

Позначення: DEC – десятковий код символу, HEX – шістнадцятковий код символу.

ДОДАТОК Д. Програма для формування коду Грея та зворотного перетворення коду Грея до двійкового числа

```
%Fuctioon for calculation the sequences
%of Grey code and for decoding
%of Grey codes. Input pavameters:
%vin - binary sequence. Only value 0 and 1
%for vector vin is allows.
%nop - number of necessary operation.
%Possible values of this parameter:
%1 - forming of Gray code.
%In this case output vector is coded
%Grey code  sequence.
%2 - decoding of Grey code sequence.
%In this case output vector is
%decoded binary sequence.
%Example of using this function
%>> c=greycode([0,1,0,0],1)
%c =
%     0     1     1     0
%>> d=greycode(c,2)
%d =
%     0     1     0     0
%>>
function vout=greycode (vin,nop)
    n=length(vin);
    dd1='Wrong input vector! Bits of vector have t';
    dd2='o be equal 0 or 1. Check input data!';
    ddstr=[dd1,dd2];
    for ii=1:n
        if(~((vin(ii)==1)|(vin(ii)==0)))
           error (ddstr); end;
        end;
```

```
    rtr1='Wrong number of operation nop. This valu';
    rtr2='e can be equal 1 for coding task or 2 f';
    rtr3='or decoding task';
    ddstr2=[rtr1,rtr2,rtr3];
    if(~((nop==1)|(nop==2))) error (ddstr2); end;
    if(nop==1)
        ffh='(ii==1).*v(1)+(ii>1).*xor(v(ii),v(ii-1))';
        vgrey=recvectbin(1,n,vin,ffh);
        vout=vgrey;
    end;
    if(nop==2)
        ffhh='(ii==1).*v(1)+(ii>1).*xor(w(ii-
1),v(ii))';
        vdescr=recvectbin(1,n,vin,ffhh);
        vout=vdescr;
    end;
return
```

Результати тестування програми

```
>> c=greycode([0,1,0,0],1)
c =
     0     1     1     0
>> d=greycode(c,2)
d =
     0     1     0     0
>>
```

ДОДАТОК Е. Програма для формування коду 4B/5B та декодування послідовностей цього коду

```
%Fuctioon for calculation the sequences
%of 4B/5B coding and for decoding
%of this codes. Input pavameters:
%vin - binary sequence. Only value 0 and 1
%for vector vin is allows. Number of bites
%of coded sequence is unconditioned If amount of
%bits not multiple of 4, additional bites of zeros
%are added in coded vector automatically.
%nop - number of necessary operation.
%Possible values of this parameter:
%1 - forming of 4B/5B code.
%In this case output vector is coded
%coded bynary sequence.
%2 - decoding of 4B/5B coded sequence.
%In this case output vector is
%decoded binary sequence.
%Number of bites of decoded sequence should to
%be multiple of 5.
%Example of using this function
%>> c=code4b5b([1,0,1,0,0,1],1)
%c =
%   1   0   1   1   0   0   1   0   0   1
%>> d=code4b5b(c,2)
%d =
%   1   0   1   0   0   0   0   1
%>>
function vout=code4b5b(vin,nop)
    n=length(vin);
    dd1='Wrong input vector! Bits of vector have t';
    dd2='o be equal 0 or 1. Check input data!';
```

```
ddstr=[dd1,dd2];
for ii=1:n
    if(~((vin(ii)==1)|(vin(ii)==0)))
       error (ddstr); end;
    end;
rtr1='Wrong number of operation nop. This valu';
rtr2='e can be equal 1 for coding task or 2 f';
rtr3='or decoding task';
ddstr2=[rtr1,rtr2,rtr3];
if(~((nop==1)|(nop==2))) error (ddstr2); end;
CodeTable=[1,1,1,1,0;0,1,0,0,1;1,...
    0,1,0,0;1,0,1,0,1;...
    0,1,0,1,0;0,1,0,1,1;0,1,1,1,0;...
    0,1,1,1,1;1,0,0,1,0;1,0,0,1,1;1,0,1,1,0;...
    1,0,1,1,1;1,1,0,1,0;1,1,0,1,1;...
    1,1,1,0,0;1,1,1,0,1];
CTR=[0,0,0,0;0,0,0,1;0,0,1,0;0,0,1,1;...
    0,1,0,0;0,1,0,1;0,1,1,0;...
    0,1,1,1;1,0,0,0;1,0,0,1;1,0,1,0;...
    1,0,1,1;1,1,0,0;1,1,0,1;1,1,1,0;1,1,1,1];
vcode=vin;
if(nop==1)
    if (mod(n,4)==1)
        vf=vin(1:n-1);
        vr=[0,0,0,vin(n)];
        vcode=[vf,vr];
    end; %End of if (mod(n,4)==1)
    if (mod(n,4)==2)
        vf=vin(1:n-2);
        vr=[0,0,vin(n-1),vin(n)];
        vcode=[vf,vr];
    end; %End of if (mod(n,4)==2)
    if (mod(n,4)==3)
```

```
            vf=vin(1:n-3);
            vr=[0,vin(n-2),vin(n-1),vin(n)];
            vcode=[vf,vr];
        end; %End of if (mod(n,4)==3)
        vout=[]; nn=length(vcode);
        for ll=0:(nn/4)-1;
            vg=vcode(4.*ll+1:4.*ll+4);
            numrow=8.*vg(1)+4.*vg(2)+2.*vg(3)+vg(4)+1;
            vr=CodeTable(numrow,:);
            vout=[vout,vr];
        end; %End of for ll=0:(n/4)-1;
    end;  %End of if (nop==1)
    if(nop==2)
        if (mod(n,5)~=0)
            dr1='Wrong bites sequence for decodin';
            dr2='g! Amount of bites should be multipl';
            dr3='e of 5. Check input data.';
            dr=[dr1,dr1,dr3];
            error (ddstr);
        end; %End of if (mod(n,5)~=0)
        vout=[]; nn2=length(vin);
        for kk=0:(nn2/5)-1;
            vg=vin(5.*kk+1:5.*kk+5); FC=0;
            for ii=1:16
                if (vg==CodeTable(ii,:))
                    vt=CTR(ii,:);
                    FC=1;
                end %End of if (vg==CodeTable(ii,:))
            end;%End of for ii=1:16
            if(FC==0)
             dft1='Wrong bites sequence for decodin';
             dft2='g! This sequence doesn''t fin';
             dft3='d in code table. Check inpu';
```

```
            dft4='t data.'
            dft=[dft1,dft2,dft3,dft4];
               error (dft);
            end; %End of if(FC==0)
            vout=[vout,vt];
        end; %End of for kk=0:(n/5)-1;
    end;  %End of if (nop==2)
return
```

Результати тестування програми

```
>> c=code4b5b([1,0,1,0,0,1],1)
c =
     1   0   1   1   0   0   1   0   0   1
>> d=code4b5b(c,2)
d =
     1   0   1   0   0   0   0   1
>> d=code4b5b([1,1,0,0,0,0,0,0,0,1],2)
??? Error using ==> code4b5b
Wrong bites sequence for decoding! This sequence
doesn't find in code table. Check input data.
>> c=code4b5b([0,0,0,0,0],1)
c =
     1   1   1   1   0   1   1   1   1   0
>> d=code4b5b(c,2)
d =
     0   0   0   0   0   0   0   0
>>
>> e=code4b5b([1,0,0,0,0,0,0,1],1)
e =
    1    0    0    1    0    0    1    0    0    1
>> digspectrum(e,3)
>> digspectrum([1,0,0,1,0,0,1,0,0,0],3)
>> digspectrum(e,3)
```

ДОДАТОК Є. Програма для скрамблювання та дескрамблювання двійкових кодових послідовностей

```
%Fuctioon for calculation the sequences
%of scrambling coding and for decoding
%of scrambling codes. Input parameters:
%vin - binary sequence. Only value 0 and 1
%for vector vin is allows.
%nop - number of necessary operation.
%Possible values of this parameter:
%1 - forming of scrambling code.
%In this case output vector is coded
%coded bynary sequence.
%2 - decoding of scrambling sequence.
%In this case output vector is
%decoded binary sequence.
%Example of using this function
%>> v=[1,1,0,1,1,0,0,0,0,0,0,1];
%>> c=scrambling(v,1)
%c =
%   1   1   0   0   0   1   1   0   1   1   1   0
%>> d=scrambling(c,2)
%d =
%   1   1   0   1   1   0   0   0   0   0   0   1
%>>
function vout=scrambling (vin,nop)
    n=length(vin);
    dd1='Wrong input vector! Bits of vector have t';
    dd2='o be equal 0 or 1. Check input data!';
    ddstr=[dd1,dd2];
    for ii=1:n
        if(~((vin(ii)==1)|(vin(ii)==0)))
           error (ddstr); end;
```

```
        end;
    rtr1='Wrong number of operation nop. This valu';
    rtr2='e can be equal 1 for scrambling task or 2 f';
    rtr3='or descrambling task';
    ddstr2=[rtr1,rtr2,rtr3];
    if(~((nop==1)|(nop==2))) error (ddstr2); end;
    if(nop==1)
        ffh1='(ii==1).*v(1)+(ii==2).*v(2)';
        ffh2='+(ii==3).*v(3)+(ii==4)';
        ffh3='.*mod2(v(ii),w(ii-3))';
        ffh4='+(ii==5).*mod2(v(ii),w(ii-3))+';
        ffh5='(ii>5).*mod32(v(ii),w(ii-3),w(ii-5))';
        ffh=[ffh1,ffh2,ffh3,ffh4,ffh5];
        vscr=recvectbin(5,n,vin,ffh,0.5,0.5);
        vout=vscr;
    end;
if(nop==2)
        ffhh1='(ii==1).*v(1)+(ii==2).*v(2)+...
                                    (ii==3).*v(3)+';
        ffhh2='(ii==4).*mod3(v(4),v(1),v(1))+(ii==5).*'
        ffhh3='mod32(v(5),v(2),v(2))+(i>5).*...
                                        mod32(v(ii),'
        ffhh4='v(ii-3),v(ii-5))';
        ffhh=[ffhh1,ffhh2,ffhh3,ffhh4];
        vdescr=recvectbin(5,n,vin,ffhh,0.5,0.5);
        vout=vdescr;
    end;
return

function out=mod2(m1,m2)
    ddr1='Error! All 3 parameters of fucntion...
                        xor3 have t';
    ddr2='o be equal 0 or 1! Check input data.';
```

```
    ddr=[ddr1,ddr2];
    if(~((m1==1)|(m1==0)))|...
        (~((m2==1)|(m2==0))) error (ddr); end;
    out=(m1==1).*(m2==1).*0+(m1==0).*(m2==0).*0+...
        (m1==1).*(m2==0).*1+(m1==0).*(m2==1).*1;
return

function out=mod32(m1,m2,m3)
    ddr1='Error! All 3 parameters of fucntion...
                          xor3 have t';
    ddr2='o be equal 0 or 1! Check input data.';
    ddr=[ddr1,ddr2];
    if(~((m1==1)|(m1==0)))|...
        (~((m2==1)|(m2==0)))|...
        (~((m3==1)|(m3==0))) error (ddr); end;
    out=(m1==1).*(m2==1).*(m3==1).*1+...
        (m1==0).*(m2==0).*(m3==0).*0+...
        (m1==1).*(m2==0).*(m3==0).*1+...
        (m1==0).*(m2==1).*(m3==0).*1+...
        (m1==0).*(m2==0).*(m3==1).*1+...
        (m1==1).*(m2==1).*(m3==0).*0+...
        (m1==0).*(m2==1).*(m3==1).*0+...
        (m1==1).*(m2==0).*(m3==1).*0;
return
```

Результати тестування програми

```
>> v=[1,1,0,1,1,0,0,0,0,0,0,1];
>> c=scrambling(v,1)
c =
     1    1    0    0    0    1    1    0    1    1    1    1
>> d=scrambling(c,2)
d =
     1    1    0    1    1    0    0    0    0    0    0    1
```

ДОДАТОК Ж. Програма для побудови коду Хаффмана та декодування його кодових послідовностей

```
%Function for calculation of Huffman code and
%for decoding of bites sequences of this code.
%Input parameters:
%1. vh - vector of probabilities of coded symbols.
%Sum of probobilities in this vector should be equal 1.
%2. ns - amount of coded symbols from the sequence,
%defined by vector vh.
%Value 1, 2 or 3 is possible for this parameter.
%For example, if ns=3, sequences of 3 symbols are
coded.
%3. vc - vector of sequence of bites, which should be
%decoding with using formed code. If this task is not
%necessary, this vector may be empty.
%4. nop - number of necessary operation.
%Possible values of this parameter:
%1 - only forming of code is required;
%2 - decoding of input vector of bites vc is also
necessary.
%Output parameters:
%1. HUFF - matrix with the Huffman code of elements.
%Bites are written in corresponded rows.
%For unfilled bites digit 5 is used.
%2. Eff - Efficiency of formed code, calculated
%by estimation the entrophy of considered seguence of
%symbols.
%3. CodeOut - Decoding seguence of symbols.
%If the task of decoding not solved, this vector is
%always empty.
%Examples of using this function:
%v=[0.4,0.2,0.2,0.1,0.05,0.05];
```

```
%[MH,E,C]=huffman(v,1,[],1);
%vcode=[1,1,1,1,1,0,1,1,1,0,0,0,1,0,0];
%[MH,E,C]=huffman(v,1,vcode,2);
function [HUFF,Eff,CodeOut]=huffman(vs,ns,vc,nop)
    %Control of input data
    ty1='Wrong value of necessary operations nop. ';
    ty2=' This parameter sould be 1 or 2.';
    ty=[ty1,ty2];
    if (~((nop==1)|(nop==2))) error (ty); end;
    dr1='Wrong amount of coded element. This ...
                                    parameter ';
    dr2='must be integer value, smaller, than 4';
    dr3='Please, check input data';
    drt=[dr1,dr2,dr3];
    if (~((ns==1)|(ns==2)|(ns==3))) error (dr1); end;
%Defining the sorting vector with taking into account
%amount of coded elements
    nvs=length(vs);
    if (ns==1) vh=vs; end;
    if (ns==2)
        ttt=0;
        for iii=1:nvs
            for jjj=1:nvs
                ttt=ttt+1;
                vh(ttt)=vs(iii)*vs(jjj);
            end; % End of for jjj=1:nvs
        end; % End of for iii=1:nvs
    end; % End of if (ns==2)
    if (ns==3)
        ttt=0;
        for iii=1:nvs
            for jjj=1:nvs
                for kkk=1:nvs
```

```
                    ttt=ttt+1;
                    vh(ttt)=vs(iii)*vs(jjj)*vs(kkk);
                end; % End of for kkk=1:nvs
            end; % End of for jjj=1:nvs
        end;  % End of for iii=1:nvs
    end; % End of if (ns==3)
%Checking the probabilities in coded vactor of symbols.
%The sum of probabilities have to be equal 1.
    cc1='Wrong probability in value number ';
    cc2='! Please, chek data!';
    n=length(vh);
    for ii=1:n
        cc=num2str(ii);
        cct=[cc1,cc,cc2];
        if((vh(ii)>1)|(vh(ii)<0)) error (cct); end;
    end; % End of for ii=1:n
    cf1='Wrong probabilities! ';
    cf2='Sum of probabilities is not equal to 1!';
    cf3='Please check input data!';
    ccr=[cf1,cf2,cf3];
    if (abs(sum(vh)-1.0)>1e-8) error (ccr); end;
    %Sorting of probabilities by increasing
    n=length(vh);
    vsort=vh; cp=1:n;
    for kkk=1:n-1
        for jjj=kkk:n
            if (vsort(kkk)<vsort(jjj))
                sd=vsort(kkk);
                vsort(kkk)=vsort(jjj);
                vsort(jjj)=sd;
                cf=cp(kkk);
                cp(kkk)=cp(jjj);
                cp(jjj)=cf;
```

```
        end; % End of if (vsort(kkk)...
    end; % End of for jjj=kkk:n
end; % End of for kkk=1:n-1
%Initial values, or patterns for vectors
%and matrices
V1=vsort; VZ=[0]; VZN=[]; MH=vsort; DH=1:n;
HUFF=5*ones(n,n-1); HOUT=[];
DHM=1:n; sh=0;
%Main cycle for treatment of matrixes
for ii=1:n-1
  %Finding sum of 2 probabilities
  VZN=[VZN,VZ];
  if (ii==1)
     SN=sum(vsort(n-ii:n));
     Spr=vsort(1:n-ii-1);
  end; % End of if (ii==1)
  if (ii>1)
     SN=sum(VN(n-ii:n));
     Spr=VN(1:n-ii-1);
  end; % End of if (ii>1)
  %Defining of probability vector
  VN=[Spr,SN,VZN];
  %Sorting of probability vector
  sh=0;
  for ll=n-2:-1:1
%Conditions of sorting
    if ((VN(ll+1)>0)&(VN(ll)<VN(ll+1)))
     %Fixation of sorting of
     %elemants in the rows of matrixes
        dhf=DH(ll+1); DH(ll+1)=DH(ll); DH(ll)=dhf;
        sdf=VN(ll+1); VN(ll+1)=VN(ll); VN(ll)=sdf;
%Indication of passing of sorting procedure
        sh=1;
```

```
        end; % End of if ((VN(ll+1)>0)&(VN(ll)...
      end; % End of for ll=n-2:-1:1
     % Defining the matrix of sums and
    % the matrix of elementes ordering
      MH=[MH;VN];
      DHM=[DHM;DH];
  end; % End of for ii=1:n-1
% Defining the matrix of numbers
% of summing elements
  SEL=[];
  for ii=1:n-1
      DFG=[DHM(ii,n-ii);DHM(ii,n-ii+1)];
      SEL=[SEL,DFG];
  end; % End of for ii=1:n-1
  % Calculation the matrix for forming of Huffman code
  MCE=zeros(n,n); ConnElem=n; MCE(SEL(1,1),1)=SEL(2,1);
  for ii=1:n-1
      % 1. Filling the bytes of current
      % elements in sum
      if (ii==1) HUFF(n-1,n-1)=1; HUFF(n,n-1)=0; end;
      if ((ii>=2))
         csel1=0;csel0=0;
         for kk=1:n-2
            if (HUFF(SEL(1,ii),kk)==5)& ...
                    (HUFF(SEL(1,ii), ...
                kk+1)~=5) HUFF(SEL(1,ii),kk)=1;
                csel1=1;
             end; % End of if (HUFF(SEL(1,ii),kk)...
            if (HUFF(SEL(2,ii),kk)==5)& ...
                    (HUFF(SEL(2,ii), ...
                    kk+1)~=5) HUFF(SEL(2,ii), ...
                    kk)=0;
                    csel0=1;
```

```
            end; % End of if (HUFF(SEL(2,ii),kk)...
        end; % End of for kk=1:n-2
        if (csel1==0) HUFF(SEL(1,ii),n-1)=1; end;
        if (csel0==0) HUFF(SEL(2,ii),n-1)=0; end;
        % Initialization the matrix of
       % connected elements
       CopyUp=SEL(1,ii); CopyDown=SEL(2,ii); Wrt=0;
       ConnElUp=MCE(CopyUp,:);
       ConnElDown=MCE(CopyDown,:);
       ConnElem=SEL(2,ii);
  %2. Writing the bites 0 and 1 in the codes
  % of connected elements
  rr=1;
  while (ConnElUp(rr)>0)
      for kk1=1:n-2
           if (HUFF(ConnElUp(rr),kk1)==5)&...
              (HUFF(ConnElUp(rr),kk1+1)~=5)...
               HUFF(ConnElUp(rr),kk1)=1;
           end; % End of if (HUFF(ConnElUp(rr)
           (HUFF(ConnElUp(rr),kk1)==5)&
        end; % End of for kk1=1:n-2
       rr=rr+1;
    end; % End of while (ConnElUp(rr)>0)
dd=1;
while (ConnElDown(dd)>0)
      for kk2=1:n-2
          if (HUFF(ConnElDown(dd),kk2)==5)&...
                (HUFF(ConnElDown(dd),kk2+1)~=5)...
                HUFF(ConnElDown(dd),kk2)=0;
          end; % End of if (HUFF(ConnElDown(dd),...
          (HUFF(ConnElDown(dd),kk2)==5)&
        end; % End of for kk2=1:n-2
       dd=dd+1;
```

```
  end; % End of while (ConnElDown(dd)>0)
        %3. Fixing of new connection by changing
        %the matrix of connected elements
        if (MCE(SEL(1,ii),1)==0)
                MCE(SEL(1,ii),1)=ConnElem;
                Wrt=1; if (MCE(SEL(1,ii),1)==0)
       end; % End of if (MCE(SEL(1,ii),1)==0)
        if (MCE(ConnElem,1)~=0)& ...
                (MCE(SEL(1,ii),1)==ConnElem)
            for tt=1:n-1
               if (MCE(ConnElem,tt)~=0)
                  MCE(CopyUp,tt+1)=MCE(ConnElem,tt);
                  MCE(ConnElem,tt)=0; Wrt=1;
               end; %End of if (MCE(ConnElem,tt)~=0)
           end %End of for tt=1:n-1
        end; %End of if (MCE(ConnElem,1)~=0)
       if (MCE(CopyUp,1)~=0)&(Wrt==0)
          for kk=1:n-1
            if (MCE(CopyUp,kk)==0)&(Wrt==0)
              MCE(CopyUp,kk)=ConnElem; Wrt=1;
               for qq=1:n-1
                 if (MCE(ConnElem,qq)~=0)
                        MCE(CopyUp,qq+kk)= ...
                        MCE(ConnElem,qq);
                         MCE(ConnElem,qq)=0;
                end; %End of if (MCE(ConnElem,qq)~=0)
               end %End of for qq=1:n-1
        end; %End of if (MCE(CopyUp,kk)~=0)
       end %End of for kk=1:n-1
    end; %End of if (MCE(CopyUp,1)~=0)
  end; %End of if (ii>=2)
 end; %End of for ii=1:n-1
% Changing the order of elements corresponding to
```

```
% input vector
 for kkk=1:n-1
     for jjj=kkk:n
            if (cp(kkk)>cp(jjj))
                sd=cp(kkk);
                cp(kkk)=cp(jjj);
                cp(jjj)=sd;
                FFF=HUFF(kkk,:);
                HUFF(kkk,:)=HUFF(jjj,:);
                HUFF(jjj,:)=FFF;
            end; %End of if (cp(kkk)>cp(jjj))
     end; %End of for jjj=kkk:n
 end; %End of for kkk=1:n-1
        HH=0; TH=0;
        for ii=1:n
         %Defining the entrophy
            HH=HH-vsort(ii)*log2(vsort(ii));
            LS=HUFF(ii,:);
            LSH(ii)=n-1;
 %Defining the length of each code element
            for jj=1:n-1
                if (LS(jj)==5)&(LS(jj+1)~=5)
                 LSH(ii)=n-jj-1;
                 break;
               end; %End of if (LS(jj)==5)&(LS(jj+1)~=5)
            end; %End of for jj=1:n-1
%Defining the avarenge time of one element transfering
            TH=TH+vsort(ii)*LSH(ii)*1e-6;
        end; %End of for ii=1:n
  Eff=(HH/TH)*1e-6
  if (nop==1) CodeOut=[]; return; end;
  % Decoding of elementa of input binary
  % vector vc coresponding to forming Huffman code.
```

```
% This part of program executed only
% for input parameter nop==2
if (nop==2)
       % Defining the manimal and maximal
       % lehgth of binary symbols in formed
       % Huffman code.
     LMAX=max(LSH); LMIN=min(LSH);
     nel=1; nelold=1; CodeOut=[];
% Control of elements of input vector vc.
   msg1='Error! Wrong elements of code vec';
   msg2='tor. All elements should be equa'
   msg3='l 0 or 1! Please, check input data!'
   msg=[msg1,msg2,msg3];
    for ll=1:length(vc)
         if (~((vc(ll)~=0)|(vc(ll)~=1)))
            error (msg);
            return;
        end; %End of if (~((vc(ll)~=0)|(vc(ll)~=1)))
    end; %End of for ll=1:length(vc)
%Cyclic structure for comparising the sequences of
%binary elements of input vector with the finded
%sequences of binary elements for Huffman code.
     msgchk1='Wrong input vector Huffamn co';
     msgchk2='de sequence! Check input data.';
     msgchk=[msgchk1,msgchk2];
     while(nel<=length(vc))
        for ww=LMIN:LMAX
            nelold=nel;
            for kk=1:n
               if (nel+ww-1>length(vc))
                     error (msgchk);
                     break;
             end; %End of if (nel+ww-1>length(vc))
```

```
                    LENGTHEL=length(HUFF(kk,...
                                  n-LSH(kk):n-1));
                    LENGTHINP=length(vc(nel:nel+ww-1));
                    if (LENGTHEL==LENGTHINP)&...
                     (HUFF(kk,n-LSH(kk):n-1...
                     )==vc(nel:nel+ww-1))
                              nelold=nel;
                              nel=nel+ww;
                              CodeOut=[CodeOut,kk];
                              break;
                    end; %End of if (LENGTHEL=...
                 end; %End of for kk=1:n
                 if (nel~=nelold) break; end;
            end; %End of for ww=LMIN:LMAX
           if(nel==nelold) error ('msgchk'); break; end;
      end; %End of while(nel<=length(vc))
   end; %End of if (nop==2)
return
```

Результати тестування програми

```
>> v=[0.22,0.2,0.16,0.16,0.1,0.1,0.04,0.02];
>> vcode=[];
>> [MH, E,C]=huffman(v,1,vcode,1);
>> MH
MH =
     5     5     5     5     5     0     1
     5     5     5     5     5     0     0
     5     5     5     5     1     1     1
     5     5     5     5     1     1     0
     5     5     5     5     1     0     0
     5     5     5     1     0     1     1
     5     5     1     0     1     0     1
     5     5     1     0     1     0     0
>>
```

```
>> E
E =
    0.9836
>> C
C =
     []
>> v=[0.4,0.2,0.2,0.1,0.05,0.05];
>> vcode=[1,1,1,1,1,1,1,1,1,1,0,0,1,0,0];
>> haffman (v,1,vcode,2);
>> [MH, E,C]=huffman(v,1,vcode,1);
>> C
C =
     3     3     3     2     1     2     1
>>
>> MH
MH =
     5     5     5     5     0
     5     5     5     1     0
     5     5     1     1     1
     5     1     1     0     1
     1     1     0     0     1
     1     1     0     0     0
>>
>> vcode=[1,1,1,1,1,0,0,0,1,0,1,1,0,1];
>> [MH,E,C]=huffman(v,1,vcode,2);
>> C
C =
     3     6     2     4
>> vcode=[1,1,1,1,1];
??? Error using ==> huffman
Wrong input vector for Huffman code sequence! Check
input data
>> v=[0.4,0.25,0.2,0.15];
```

```
>> vcode=[];
>> v=[0.4,0.25,0.2,0.15];
>> [MH,E,C]=huffman(v,1,vcode,1);
MH =
     5     5     0
     5     1     0
     1     1     1
     1     1     0
>> E
E =
    0.9763
>> help huffman
 Function for calculation of Huffman code and
 for decoding of bites sequences of this code.
 Input parameters:
 1. vh - vector of probabilities of coded symbols.
 Sum of probabilities in this vector should be equal 1.
 2. ns - amount of coded symbols from the sequence,
 defined by vector vh.
 Value 1, 2 or 3 is possible for this parameter.
 For example, if ns=3, sequences of 3 symbols are
coded.
 3. vc - vector of sequence of bites, which should be
decoding
 with using formed code. If this task is not necessary,
 this vector may be empty.
 4. nop - number of necessary operation.
 Possible values of this parameter:
 1 - only forming of code is required;
```

```
 2 - decoding of input vector of bites vc is also
necessary.
 Output parameters:
 1. HUFF - matrix with the Huffman code of elements.
 Bites are written in corresponded rows.
 For unfilled bites digit 5 is used.
 2. Eff - Efficiency of formed code, calculated by
estimation
 the entropy of considered sequence of symbols.
 3. CodeOut - Decoding sequence of symbols. If the task
of
 decoding not solved, this vector is always empty.
 Examples of using this function:
 v=[0.4,0.2,0.2,0.1,0.05,0.05];
 [MH,E,C]=huffman(v,1,[],1);
 vcode=[1,1,1,1,1,0,1,1,1,0,0,0,1,0,0];
>> v=[0.4,0.25,0.2,0.15];
>> [MH,E,C]=huffman(v,1,[],1);
>> MH
MH =
     5     5     0
     5     1     0
     1     1     1
     1     1     0
>> E
E =
    0.9763
>>
>> v=[0.7,0.2,0.1];
>> [MH,E,C]=huffman(v,1,[],1);
```

```
>> MH
MH =
     5     1
     0     1
     0     0
>> E
E =
    0.8898
>> [MH,E,C]=huffman(v,2,[],1);
>> MH
MH =
     5     5     5     5     5     5     5     0
     5     5     5     5     5     1     1     1
     5     5     5     5     1     0     1     1
     5     5     5     5     5     1     1     0
     5     5     5     5     1     0     0     0
     5     5     5     1     0     0     1     0
     5     5     5     5     1     0     1     0
     5     5     1     0     0     1     1     1
     5     5     1     0     0     1     1     0
>> E
E =
    0.9560
>>
>> v=[0.25,0.25,0.25,0.25];
>> [MH,E,C]=huffman(v,1,[],1);
>> MH
MH =
     5     0     1
     5     0     0
```

```
     5     1     1
     5     1     0
>> E
E =
     1
>>
>> v=[0.3,0.25,0.25,0.2];
>> [MH,E,C]=huffman(v,1,[],1);
>> MH
MH =
     5     1     1
     5     1     0
     5     0     1
     5     0     0
>> E
E =
    0.9927
>>
```

ДОДАТОК 3. Програма для побудови коду Хеммінга та декодування його кодових послідовностей

```
%Fuction for calculation the sequences of Hamming
%coding and for decoding of these codes. Input
%parameters:
%vin - binary sequence. Only value 0 and 1 for vector
%vin is allows. Number of bites of coded sequence is
%from 4 to 26, and for decoded sequence - from 7 to 31.
%Another function parameters are:
%nop - number of necessary operation.
%Possible values of this parameter:
%1 - forming of Hamming code.
%In this case output vector is coded coded binary
%sequence.
%2 - decoding of Hamming coded sequence.
%In this case output vector is decoded binary sequence.
%Error position is also noted.
%TOC - type of code.
%Possible values of this parameter:
%1 - Classic variant of Hamming code for correction of
%single errors.
%2 - Modified variant of Hamming code for correction of
%single errors and detection of double errors.
%Distignition feature of this function is presentation
%the input and output vector of bit sequence. Since in
%mathematic right to left order of digits is usually
%used such order also has been choosing for input and
%output sequences. For example, for presentation the
%sequence 0010, vector [0,0,1,0] is used.
```

```
%Output vector [1,1,0,1,0,1,0] corresponded to sequence
%1101010.
%See also Hamming_Summing
%Examples of using this function
%>> c=Hamming_Coding([1,0,1,0,0,1],1,1)
%c =
%   1   0   1   1   0   0   1   1   1   0
%>> d=Hamming_Coding([1,0,1,0,0,1],1,2)
%d =
%   0   1   0   1   1   0   0   1   1   1   0
%>>
%>> f=Hamming_Coding(c,2,1)
%The Hemmeing Code Sequence is correct. No errors
%f =
%   1   0   1   0   0   1
%>>
%>> c(4)=0;
%>> f=Hamming_Coding(c,2,1)
%>>The Hemmeing Code Sequence is incorrect. Error in
%the 7-th digit detected.
%>>f =
%>>  1   0   1   0   0   1
%>>
function Vout=Hamming_Coding(vin,nop,TOC)
    nv=length(vin);
    dd1='Wrong input vector! Bits of vector have t';
    dd2='o be equal 0 or 1. Check input data!';
    ddstr=[dd1,dd2];
    for ii=1:nv
        if(~((vin(ii)==1)|(vin(ii)==0)))
```

```
        error (ddstr); end;
    end;
rtr1='Wrong number of operation nop. This valu';
rtr2='e can be equal 1 for coding task or 2 f';
rtr3='or decoding task';
ddstr2=[rtr1,rtr2,rtr3];
if(~((nop==1)|(nop==2))) error (ddstr2); end;
ftr1='Wrong number of type of code TOC. This valu';
ftr2='e can be equal 1 for classic Hamming code or 2 f';
ftr3='or modified Hamming code';
fddstr2=[ftr1,ftr2,ftr3];
if(~((TOC==1)|(TOC==2))) error (fddstr2); end;
etr1='Wrong amount of bites in input vector. Coded sequenc';
etr2='e can inclued not less then 4 bites but not mor';
etr3='e then 26 bytes';
estr=[etr1,etr2,etr3];
if (((nv<4)|(nv>26))&(nop==1)) error (estr); end;
ttr1='Wrong amount of bites in input vector. De';
ttr2='coded sequenc';
ttr3='e can inclued not less then 7 bites ';
ttr4='but not more then 31 bytes ';
tstr=[ttr1,ttr2,ttr3, ttr4];
vininv=vin(nv:-1:1); zerom(1:31)=0;
if ((nv<7)|(nv>31))&(nop==2) error (tstr); end;
if (nop==1) k=nv; end; if (nop==2) n=nv; end;
if (nop==1)
    if (k>=4)
        OutVec1=[0,0,vininv];
        S1=OutVec1(1:3); S2=OutVec1(4:nv+2);
        OutVec2=[S1,0,S2];
```

```
        SOUT=length(OutVec2);
        z=zerom(8:31);
        if (k==4)
            OutVec3=[OutVec2,z]; NOUT=k+3;
         end; % if(k==4)
    end; %if (k>=4)
    if ((SOUT>7)&(SOUT<=14))
        z=zerom(SOUT:32-3);
        S3=OutVec2(1:7); S4=OutVec2(8:SOUT);
        OutVec3=[S3,0,S4,z];
        NOUT=k+4;
    end; %if((SOUT>7)&(SOUT<=14))
    if ((SOUT>=15)&(SOUT<=31))
        z=zerom(SOUT:32-4);
        S5=OutVec2(1:7); S6=OutVec2(8:14);
        S7=OutVec2(15:SOUT);
        OutVec3=[S5,0,S6,0,S7,z];
        NOUT=k+5;
    end; %if((SOUT>=15)&(SOUT<=31))
    R=Hamming_Summing(OutVec3);
    OutVec3(1)=R(1); OutVec3(2)=R(2);
    OutVec3(4)=R(3); OutVec3(8)=R(4);
    OutVec3(16)=R(5);
    OutVec=OutVec3(1:NOUT);
    VOutInv=OutVec(NOUT:-1:1);
end;%if(nop==1)
if ((nop==1)&(TOC==1)) Vout=VOutInv; end;
if (nop==1)&(TOC==2)
   r6=mod(sum(VOutInv),2);
   Vout=[r6,VOutInv];
```

```
end;% if (nop==1)&(TOC==2)
ERROUTTEXT1='The Hemmeing Code Sequence is in'
ERROUTTEXT2='correct. Error in the ';
ERROUTTEXT3='-th digit detected.';
if ((nop==2)&(TOC==1))
    z=zerom(nv:31);
    Vc=[vininv,z];
    R=Hamming_Summing(Vc);
    SND=16.*R(5)+8.*R(4)+4.*R(3)+2.*R(2)+R(1);
    if (SND~=0)
        if (Vc(SND)==1) vininv(SND)=0; end;
        if (Vc(SND)==0) vininv(SND)=1; end;
    end;% if(SND~=0)
    if (nv==7) Vout2=[vininv(3),...
                vininv(5:7)]; end;
     if ((nv>7)&(nv<=15))
            Vout2=[vininv(3),vininv(5:7),...
                          vininv(9:nv)]; end;
    if ((nv>=16)&(nv<=31))
            Vout2=[vininv(3),vininv(5:7),...
            vininv(9:15),vininv(17:nv)]; end;
    ghy1='The Hemmeing Code Sequence is c'
     ghy2='orrect. No errors'
     ghy=[ghy1, ghy2];
     if (SND==0) disp (ghy); end;
    ERP=Num2Str(SND);
    ERROUTTEXT=[ERROUTTEXT1, ERROUTTEXT2,...
         ERP,ERROUTTEXT3];
    if (SND~=0) disp (ERROUTTEXT); end;
    lout=length(Vout2);
```

```
    Vout=Vout2(lout:-1:1);
end; %if((nop==2)&(TOC==1))
if ((nop==2)&(TOC==2))
    z=zerom(nv-1:31); Vc2=[vininv(1:nv-1),z];
    Vc=[vininv,z];
    R=Hamming_Summing(Vc2);
    SND2=16.*R(5)+8.*R(4)+4.*R(3)+2.*R(2)+R(1);
    ER2=mod(sum(vininv),2);
    if (SND2~=0)&(ER2==1)
        if (Vc(SND2)==1) vininv(SND2)=0; end;
        if (Vc(SND2)==0) vininv(SND2)=1; end;
    end;
    if ((SND2==0)&(ER2==0)|(SND2==0)&(ER2==1)...
                  |(SND2~=0)&(ER2==1))
        if (nv==8) Vout3=[vininv(3),...
                  vininv(5:7)]; end;
        if ((nv>8)&(nv<=16))
             Vout3=[vininv(3),vininv(5:7),...
             vininv(9:nv-1)]; end;
        if ((nv>=17)&(nv<=31))
             Vout3=[vininv(3),vininv(5:7),
             vininv(9:15),vininv(17:nv-1)]; end;
    end; %if((SND2==0)&(ER2==0)
    dfg1='Double error. The Modifie
     dfg2='d Hemmeing Code Sequence i'sn''t c;
    dfg3='orrect and can''nt be decoded.';
    dfg=[dfg1,dfg2,dfg3];
    if (SND2~=0)&(ER2==0)
         Vout3=[]; disp (dfg);
     end; %if(SND2~=0)&(ER2==0)
```

```
        gks1='The Modified Hemmeing Code S';
        gks2='equence is correct. No errors';
        gks=gks1+gks2;
        if ((SND2==0)&(ER2==0)) disp (gks); end;
        if (SND2==0)&(ER2==1) ERP2=Num2Str(nv);
                else ERP2=Num2Str(SND2); end;
        ERROUTTEXT=[ERROUTTEXT1,ERP2,ERROUTTEXT2];
        if ((SND2~=0)&(ER2==1)) disp (ERROUTTEXT); end;
        if (SND2==0)&(ER2==1) disp (ERROUTTEXT); end;
        lout3=length(Vout3);
        Vout=Vout3(lout3:-1:1);
    end;
return

%Fuction for calculation the Checksums
%of Hamming Codes. See also Hamming_Coding
function SH=Hamming_Summing(VS)
    Sum1=VS(1)+VS(3)+VS(5)+VS(7)+VS(9)+...
            VS(11)+VS(13)+VS(15)+VS(17)+...
            VS(19)+VS(21)+VS(23)+VS(25)+...
            VS(27)+VS(29)+VS(31);
    Sum2=VS(2)+VS(3)+VS(6)+VS(7)+...
            VS(10)+VS(11)+VS(14)+VS(15)+...
            VS(18)+VS(19)+VS(22)+VS(23)+...
            VS(26)+VS(27)+VS(30)+VS(31);
    Sum3=VS(4)+VS(5)+VS(6)+VS(7)+...
            VS(12)+VS(13)+VS(14)+VS(15)+...
            VS(20)+VS(21)+VS(22)+VS(23)+...
            VS(28)+VS(29)+VS(30)+VS(31);
    Sum4=VS(8)+VS(9)+VS(10)+VS(11)+...
```

```
            VS(12)+VS(13)+VS(14)+VS(15)+...
            VS(24)+VS(25)+VS(26)+VS(27)+...
            VS(28)+VS(29)+VS(30)+VS(31);
    Sum5=VS(16)+VS(17)+VS(18)+VS(19)+...
            VS(20)+VS(21)+VS(22)+VS(23)+...
            VS(24)+VS(25)+VS(26)+VS(27)+...
            VS(28)+VS(29)+VS(30)+VS(31);
    SH=mod([Sum1,Sum2,Sum3,Sum4,Sum5],2);
return
```

Результати тестування програми

```
>> d=Hamming_Coding([1,1,1,0],1,1)
d =
     1   1   1   1   0   0   0
>> f=Hamming_Coding(d,2,1)
The Hemmeing Code Sequence is correct. No errors
f =
     1   1   1   0
>> d(3)=0;
>> g=Hamming_Coding(c,2,1)
The Hemmeing Code Sequence is incorrect. Error in the
5-th digit detected.
g =
     1   1   1   0
>> d(1)=0;
>> h=Hamming_Coding(c,2,1)
The Hemmeing Code Sequence is incorrect. Error in the
2-th digit detected.
h =
     0   1   0   0
>> c=Hamming_Coding([1,1,1,0],1,2)
```

```
c =
     0   1   1   1   1   0   0   0
>> d=Hamming_Coding(c,2,2)
The Modified Hemmeing Code Sequence is correct. No
errors
d =
     1   1   1   0
>> c(3)=0;
>> d=Hamming_Coding(c,2,2)
The Hemmeing Code Sequence is incorrect. Error in the
6-th digit detected.
d =
     1   1   1   0
>> c(5)=0;
>> d=Hamming_Coding(c,2,2)
Double error. The Modified Hemmeing Code Sequence isn't
correct and can'nt be decoded.
d =
   Empty matrix: 1-by-0
>>
>> c=Hamming_Coding([1,0,1,0,0,1],1,1)
c =
     1   0   1   1   0   0   1   1   1   0
>> d=Hamming_Coding(c,2,1)
The Hemmeing Code Sequence is correct. No errors
d =
     1   0   1   0   0   1
>>
```

ДОДАТОК І. Незвідні поліноми з першого до восьмого порядків

Таблиця Д2

№ з/п	Степінь	Поліноміальне подання	Числове подання
1.	1	$x + 1$	11
2.	2	$x^2 + x + 1$	111
3.	3	$x^3 + x + 1$	1011
4.	3	$x^3 + x^2 + 1$	1101
5.	4	$x^4 + x + 1$	10011
6.	4	$x^4 + x^3 + 1$	11001
7.	4	$x^4 + x^3 + x^2 + x + 1$	11111
8.	5	$x^5 + x^2 + 1$	100101
9.	5	$x^5 + x^3 + 1$	101001
10.	5	$x^5 + x^3 + x^2 + x + 1$	101111
11.	5	$x^5 + x^4 + x^2 + x + 1$	110111
12.	5	$x^5 + x^4 + x^3 + x + 1$	111011
13.	5	$x^5 + x^4 + x^3 + x^2 + 1$	111101
14.	6	$x^6 + x + 1$	1000011
15.	6	$x^6 + x^3 + 1$	1001001
16.	6	$x^6 + x^4 + x^2 + x + 1$	1010111
17.	6	$x^6 + x^4 + x^3 + x + 1$	1011011
18.	6	$x^6 + x^5 + 1$	1100001
19.	6	$x^6 + x^5 + x^2 + x + 1$	1100111
20.	6	$x^6 + x^5 + x^3 + x^2 + 1$	1101101
21.	6	$x^6 + x^5 + x^4 + x + 1$	1110011
22.	6	$x^6 + x^5 + x^4 + x^2 + 1$	1110101
23.	7	$x^7 + x + 1$	10000011
24.	7	$x^7 + x^3 + 1$	10001001
25.	7	$x^7 + x^3 + x^2 + x + 1$	10001111
26.	7	$x^7 + x^4 + 1$	10010001
27.	7	$x^7 + x^4 + x^3 + x^2 + 1$	10011101

Таблиця Д2, продовження

№ з/п	Степінь	Поліноміальне подання	Числове подання
28.	7	$x^7 + x^5 + x^2 + x + 1$	10100111
29.	7	$x^7 + x^5 + x^3 + x + 1$	10101011
30.	7	$x^7 + x^5 + x^4 + x^3 + 1$	10111001
31.	7	$x^7 + x^5 + x^4 + x^3 + x^2 + x + 1$	10111111
32.	7	$x^7 + x^6 + 1$	11000001
33.	7	$x^7 + x^6 + x^3 + x + 1$	11001011
34.	7	$x^7 + x^6 + x^4 + x + 1$	11010011
35.	7	$x^7 + x^6 + x^4 + x^2 + 1$	11010101
36.	7	$x^7 + x^6 + x^5 + x^2 + 1$	11100101
37.	7	$x^7 + x^6 + x^5 + x^3 + x^2 + x + 1$	11101111
38	7	$x^7 + x^6 + x^5 + x^4 + 1$	11110001
39.	7	$x^7 + x^6 + x^5 + x^4 + x^2 + x + 1$	11110111
40.	7	$x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + 1$	11111101
41.	8	$x^8 + x^4 + x^3 + x + 1$	100011011
42.	8	$x^8 + x^4 + x^3 + x^2 + 1$	100011101
43.	8	$x^8 + x^5 + x^3 + x + 1$	100101011
44.	8	$x^8 + x^5 + x^3 + x^2 + 1$	100101101
45.	8	$x^8 + x^5 + x^4 + x^3 + 1$	100111001
46.	8	$x^8 + x^5 + x^4 + x^3 + x^2 + x + 1$	100111111
47.	8	$x^8 + x^6 + x^3 + x^2 + 1$	101001101
48.	8	$x^8 + x^6 + x^4 + x^3 + x^2 + x + 1$	101011111
49.	8	$x^8 + x^6 + x^5 + x + 1$	101100011
50.	8	$x^8 + x^6 + x^5 + x^2 + 1$	101100101
51.	8	$x^8 + x^6 + x^5 + x^3 + 1$	101101001
52.	8	$x^8 + x^6 + x^5 + x^4 + 1$	101110001
53.	8	$x^8 + x^6 + x^5 + x^4 + x^2 + x + 1$	101110111
54.	8	$x^8 + x^6 + x^5 + x^4 + x^3 + x + 1$	101111011
55.	8	$x^8 + x^7 + x^2 + x + 1$	110000111

Таблиця Д2, закінчення

№ з/п	Степінь	Поліноміальне подання	Числове подання
56.	8	$x^8 + x^7 + x^3 + x + 1$	110001011
57.	8	$x^8 + x^7 + x^3 + x^2 + 1$	110001101
58.	8	$x^8 + x^7 + x^4 + x^3 + x^2 + x + 1$	110011111
59.	8	$x^8 + x^7 + x^5 + x + 1$	110100011
60.	8	$x^8 + x^7 + x^5 + x^3 + 1$	110101001
61.	8	$x^8 + x^7 + x^5 + x^4 + 1$	110110001
62.	8	$x^8 + x^7 + x^5 + x^4 + x^3 + x^2 + 1$	110111101
63.	8	$x^8 + x^7 + x^6 + x + 1$	111000011
64.	8	$x^8 + x^7 + x^6 + x^3 + x^2 + 1$	111001101
65.	8	$x^8 + x^7 + x^6 + x^4 + x^2 + x + 1$	111010111
66.	8	$x^8 + x^7 + x^6 + x^4 + x^3 + x^2 + 1$	111011101
67.	8	$x^8 + x^7 + x^6 + x^5 + x^2 + x + 1$	111100111
68.	8	$x^8 + x^7 + x^6 + x^5 + x^4 + x + 1$	111110011
69.	8	$x^8 + x^7 + x^6 + x^5 + x^4 + x^2 + 1$	111110101
70.	8	$x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + 1$	111111001

ДОДАТОК К. Програма для формування циклічних кодів та декодування їхніх кодових послідовностей, основана на алгоритмах множення та ділення двійкових чисел

```
%Function for finding production of 2
%binary vectors by modulus 2. All elements
%of multiplied vectors should be equil 0
%or 1. Example of using this function:
%>> R=Bin_Product([1,0,1,0],[1,0,1,1,0,0])
%R =
%   1   0   0   1   1   1   0   0   0
%>>
%See also Bin_Division and CRC
function Res=Bin_Product(VS1,VS2)
    n1=length(VS1); n2=length(VS2);
    dd1='Wrong input vector V1! Bits of vector have t';
    dd2='o be equal 0 or 1. Check input data!';
    ddstr=[dd1,dd2];
    for ii=1:n1
        if(~((VS1(ii)==1)|(VS1(ii)==0)))
           error (ddstr); end;
    end;
    dds1='Wrong input vector V2! Bits of vector have t';
    ddstr2=[dds1,dd2];
    for ii=1:n2
        if(~((VS2(ii)==1)|(VS2(ii)==0)))
           error (ddstr); end;
    end;
    if(n1>n2) P1=VS1;  P2=VS2; else P1=VS2; P2=VS1;
end;
```

```
    nrows=length(P2);  ncoll=nrows+length(P1);
    MProd=zeros(1,ncoll-1);
    for iii=nrows:-1:1
        if(P2(iii)==1)
           ZL=zeros(1,iii-1); ZR=zeros(1,nrows-iii);
           SV=[ZL,P1,ZR]; MProd=[MProd;SV];
        end;
    end;
    SC=sum(MProd); Res=mod(SC,2);
return

%Function for finding devision of 2
%binary vectors by modulus 2. All elements
%of devided vectors should be equil 0
%or 1. Input parameters:
%First vector V1 - the dividend.
%Second vector V2 - the divisor.
%Length of V1 hane to be grater,
%than length of V2. Since the preorder of
%digit location is used, therefore when
%the fisrt elements of vectors is equil 0,
%they are ignored.
%The result of using this function is
%the quotient, which is arranged at the first
%row of matrix of result RES, and the remainder,
%which is located at the second row of it.
%Example of using this function:
%>> R=Bin_Division([1,0,1,0,1,0,1],[1,0,1,1,])
%R =
%R =
```

```
%     1     0     0     1
%     0     1     1     0
%>>
%See also Bin_Product and CRC
function RES=Bin_Division(VS1,VS2)
    n1=length(VS1); n2=length(VS2);
    dd1='Wrong input vector V1! Bits of vector have t';
    dd2='o be equal 0 or 1. Check input data!';
    ddstr=[dd1,dd2];
    for ii=1:n1
        if(~((VS1(ii)==1)|(VS1(ii)==0)))
           error (ddstr); end;
    end;
    dds1='Wrong input vector V2! Bits of vector have t';
    ddstr2=[dds1,dd2];
    for ii=1:n2
        if(~((VS2(ii)==1)|(VS2(ii)==0)))
           error (ddstr); end;
    end;
    ii=1;
    while (VS1(ii)==0) ii=ii+1; end;
    VD1=VS1(ii:n1);
    jj=1;
    while (VS2(jj)==0)jj=jj+1; end;
    VD2=VS2(jj:n2);
    wstr1='Both input binary vectors can not b';
    wstr2='e zero vectors. Check input data!';
    wstr=[wstr1,wstr2];
    if ((ii==n1)|(jj==n2)) error (wstr); end;
    qqq1='Wrong input data. Length of first vector, t';
```

```
qqq2='he dividend, have to be grater, than length o';
qqq3='f second vector, the divisor. First zeros in b';
qqq4='its sequences not take into account Check inpu';
qqq5='t data!';
qstr=[qqq1, qqq2, qqq3, qqq4, qqq5];
nd1=length(VD1); nd2=length(VD2);
if (nd1<=nd2) error (qstr); end;
MD=VD1;
Rem_Length=nd1; iter=1; NLZ(1)=0;
while (Rem_Length>=nd2)
    if(iter==1)
        NRZ=nd1-nd2;
        RZ=zeros(1,NRZ);
        Div=[VD2,RZ];
        Rem=XOR(VD1,Div);
        MD=[MD;Div;Rem];
        ResVect=[1];
    end;  %if(iter==1)
    ir=1;
 while (MD(2*(iter-1)+3,ir)==0)&(ir<nd1)
         ir=ir+1;
 end;
    if (ir==nd1) break; end;
    iter=iter+1;
    NLZ(iter)=ir-1;
    Rem_Length=nd1-NLZ(iter);
    if(iter>1)&(Rem_Length>=nd2)
        LZ=zeros(1,NLZ(iter));
        RZ=zeros(1,nd1-NLZ(iter)-nd2);
        Div=[LZ,VD2,RZ];
```

```
            NZRes=NLZ(iter)-NLZ(iter-1)-1;
            NZResVect=zeros(1,NZRes);
            ResVect=[ResVect,NZResVect,[1]];
            SMDC=size(MD);
            Rem=XOR(MD(SMDC(1),:),Div);
            MD=[MD;Div;Rem];
        end; %if(iter>1)
    end; %while (Rem_Length>=nd2)
    ResVect=[ResVect,RZ];
    SMD=size(MD); LR=SMD(1);
    Remaind=MD(LR,:);
    irs=1;
    while (Remaind(irs)==0)&(irs<nd1)
         irs=irs+1;
    end;
    if ((irs-1)==nd1) Rem1=[0]; else
         Rem1=Remaind(irs:nd1); end;
    NQ=length(ResVect); NR=length(Rem1);
    if (NQ==NR) RES=[ResVect;Rem1]; end;
    if (NQ>NR)
        NSQ=NQ-NR; ZSQ=zeros(1,NSQ);
        Rem2=[ZSQ,Rem1];
        RES=[ResVect;Rem2];
    end;
    if (NQ<NR)
        NSR=NR-NQ; ZSR=zeros(1,NSR);
        ResVect2=[ZSR,ResVect];
        RES=[ResVect2;Rem1];
    end;
return
```

```
% Function for left and right cyclic shifting of
% vectors elements.
% Input parameters:
% vin - input vector.
% nop - number of necessary operation.
% Possible values of this parameter:
% 1 - left cyclic shifting.
% 2 - right cyclic shifting.
% This operation is used in computer arifmatic for
% forming of Cyclic Recurrent Codes.
% Examles of using this function:
% >> CSHRL([1,0,1,1,1,0],1)
% ans =
%     0   1   1   1   0   1
% >> CSHRL([1,0,1,1,1,0],2)
% ans =
%     0   1   0   1   1   1
% >>
% See also CRC
function Vout=CSHRL(vin,nop)
    nvin=length(vin);
    if (nop==1) Vout=[vin(2:nvin),vin(1)]; end;
    if (nop==2) Vout=[vin(nvin),vin(1:nvin-1)]; end;
return

%Fuction for calculation the sequences of Cyclic
%Recurrent Codes (CRC) and for decoding
%of these codes. Input parameters:
%vin - binary sequence. Only value 0 and 1
```

```
%for vector vin is allows. Number of
%bites of coded sequence is from 4 to 26,
%and for decoded sequence - from 7 to 31.
%Another function parameters are:
%nop - number of necessary operation.
%Possible values of this parameter:
%1 - forming of CRC.
%In this case output vector is coded coded binary
%sequence.
%2 - decoding of CRC sequence.
%In this case output vector is decoded binary sequence.
%Error position is also noted.
%TOC - type of CRC.
%Possible values of this parameter:
%1 - Non-Systematic CRC for correction of single
%errors.
%2 - Systematic CRC for correction of single errors.
%Standard creation polynomial fumction are used for
%forming CRC.
%Examples of using this function
%>> c=CRC([1,0,1,0,0,1],1,1)
%c =
%   1   0   1   1   1   0   1   0   1   1
%>> f=CRC(c,2,1)
%CRC is correct. Coded sequence is vout=1 0 1 0 0 1
%f =
%   1   0   1   0   0   1
%>> c(10)=0
%c =
%   1   0   1   1   1   0   1   0   1   0
```

```
%>> f=CRC(c,2,1)
%CRC is incorrect. Error in the 1-th digit detected.
%Corrected code sequence is vin=1 0 1 1 1 0 1 0 1 1
%Coded sequence is vout=1  0  1  0  0  1
%f =
%   1   0   1   0   0   1
%See also: Bin_Division, Bin_Product and CSHRL
function Vout=CRC(vin,nop,TOC)
    nv=length(vin);
    dd1='Wrong input vector! Bits of vector have t';
    dd2='o be equal 0 or 1. Check input data!';
    ddstr=[dd1,dd2];
    for ii=1:nv
        if(~((vin(ii)==1)|(vin(ii)==0)))
            error (ddstr); end; %if(~((vin(ii)==1)
    end; %for ii=1:nv
    rtr1='Wrong number of operation nop. This valu';
    rtr2='e can be equal 1 for coding task or 2 f';
    rtr3='or decoding task'; ddstr2=[rtr1,rtr2,rtr3];
    if(~((nop==1)|(nop==2))) error (ddstr2); end;
    ftr1='Wrong number of type of code TOC. This valu';
    ftr2='e can be equal 1 for non-systematic CRC code or 2 f';
    ftr3='or systematic CRC code';
    fddstr2=[ftr1,ftr2,ftr3];
    if(~((TOC==1)|(TOC==2))) error (fddstr2); end;
    etr1='Wrong amount of bites in input vector. Coded sequenc';
    etr2='e can inclued not less then 4 bites but not mor';
    etr3='e then 26 bytes'; estr=[etr1,etr2,etr3];
    if (((nv<4)|(nv>26))&(nop==1)) error (estr); end;
    ttr1='Wrong amount of bites in input vector';
```

```
ttr2='. Decoded sequence can inclued not less th';
ttr3='en 7 bites but not mor e then 31 bytes';
tstr=[ttr1,ttr2,ttr3];
if ((nv<7)|(nv>31))&(nop==2) error (tstr); end;
if(nop==1)
    if (nv==4) TP=[1,0,1,1]; end;
    if (nv>=5)&(nv<=11) TP=[1,0,0,1,1]; end;
    if (nv>11)&(nv<=26) TP=[1,0,1,0,0,1]; end;
end; %if(nop==1)
if(nop==2)
    if (nv==7) TP=[1,0,1,1]; end;
    if (nv>=8)&(nv<=15) TP=[1,0,0,1,1]; end;
    if (nv>16)&(nv<=31) TP=[1,0,1,0,0,1]; end;
end; %if(nop==2)
if(nop==1)&(TOC==1)
    Vout=Bin_Product(vin,TP);
end; %if(nop==1)&(TOC==1)
if(nop==1)&(TOC==2)
    if (nv==4) VS=[vin,0,0,0]; end;
    if (nv>=5)&(nv<=11) VS=[vin,0,0,0,0]; end;
    if (nv>16)&(nv<=31) VS=[vin,0,0,0,0,0]; end;
    MC=Bin_Division(VS,TP); RC=MC(2,:);
    NVS=length(VS); NRC=length(RC);
    DIFL=NVS-NRC; ZL=zeros(1,DIFL);
    RCM=[ZL,RC];
    Vout=xor(RCM,VS);
end; %if(nop==1)&(TOC==2)
OUTEXT='CRC is correct. Coded sequence is vout=';
ERROUTTEXT1='CRC is incorrect. Error in the ';
ERROUTTEXT2='-th digit detected.';
```

```
ERROUTTEXT3='Corrected code sequence is vin=';
ERROUTTEXT4='Coded sequence is vout=';
if(nop==2)&(TOC==1)
    MC=Bin_Division(vin,TP);
    Res=MC(1,:); Rem=MC(2,:);
     LRem=length(Rem); jj=1;
    while (Rem(jj)==0)&(jj<=LRem)
        jj=jj+1; if (jj==LRem+1) break; end;
    end; %while (Rem(jj)==0)&(jj<=LRem)
    if(jj==LRem+1)
        Vout=MC(1,:);
        Vout=Res; CSS=Num2Str(Vout);
        OT1=[OUTEXT,CSS]; disp(OT1);
    end; %if(jj==LRem+1)
    if(jj~=LRem+1)
         NV=vin; nsh=0; dr=sum(Rem); NV=length(NV);
         if(dr==1)
            Lrem=length(Rem); DN=LNV-Lrem;
            ZL1=zeros(1,DN);  RemS=[ZL1,Rem];
            Vcorr=xor(RemS,NV);
            Out=Bin_Division(Vcorr,TP);
            Vout=Out(1,:);
            nn=length(RemS);
            while (RemS(nn)==0) nn=nn-1; end;
            nnn=length(RemS)-nn+1;
           ERP=Num2Str(nnn); CS=Num2Str(Vcorr);
            ETOUT1=[ERROUTTEXT1,ERP,ERROUTTEXT2];
            ETOUT2=[ERROUTTEXT3,CS];
            disp(ETOUT1); disp(ETOUT2);
            CS2=Num2Str(Vout);
```

```
            ETOUT3=[ERROUTTEXT4,CS2]; disp(ETOUT3);
        end; %if(dr==1)
        chng=0;
        while (dr>1)
            NV=CSHRL(NV,1);
            MCW=Bin_Division(NV,TP);
            RemW=MCW(2,:); dr=sum(RemW);
            nsh=nsh+1;      chng=1;
        end; %while (dr>1)
        if(chng==1)
            Lrem2=length(RemW); DN2=LNV-Lrem2;
            ZL2=zeros(1,DN2);    Rem2=[ZL2,RemW];
            NVSH=xor(NV,Rem2);
            for ii=nsh:-1:1 NVSH=CSHRL(NVSH,2); end;
            CD=Bin_Division(NVSH,TP);
            Vout=CD(1,:); ERP=Num2Str(nsh);
            CS=Num2Str(NVSH);
            ERRORTTEXTOUT1=[ERROUTTEXT1,...
                  ERP,ERROUTTEXT2];
            ERRORTTEXTOUT2=[ERROUTTEXT3,CS];
            disp(ERRORTTEXTOUT1);
            disp(ERRORTTEXTOUT2);
            CS2=Num2Str(Vout);
            ERRORTTEXTOUT3=[ERROUTTEXT4,CS2];
            disp(ERRORTTEXTOUT3);
        end; %if(chng==1)
    end;%if(jj~=LRem+1)
end; %if(nop==2)&(TOC==1)
if(nop==2)&(TOC==2)
    MCW2=Bin_Division(vin,TP);
```

```
RemW2=MCW2(2,:); LRem2=length(RemW2);
jj=1;
while (RemW2(jj)==0)&(jj<=LRem2)
    jj=jj+1; if (jj==LRem2+1) break; end;
end; %while (Rem(jj)==0)&(jj<=LRem)
if (nv==7) LSER=3; end;
if (nv>=8)&(nv<=15) LSER=4; end;
if (nv>16)&(nv<=31) LSER=5; end;
if(jj==LRem2+1)
    Res2=vin(1:nv-LSER);
    Vout=Res2; CSS2=Num2Str(Vout);
    OT1=[OUTEXT,CSS2];
    disp(OT1);
end; %if(jj==LRem+1)
if(jj<LRem2+1)
    ZR=zeros(1,nv-1); VCER=[1,ZR];
    SERM=Bin_Division(vin,TP); SER=SERM(2,:);
    LSR=length(SER);
    SERSH=SER(LSR-LSER+1:LSR);
    RSHC=Bin_Division(VCER,TP);
    SERSHC=RSHC(2,LSR-LSER+1:LSR);
    SHLSEQ=vin;
    ci=0; SERINT=1;
    while (SERINT~=0)
        MRESC=Bin_Division(SHLSEQ,TP);
        RESC=MRESC(2,:); Lnew=length(RESC);
        SERSH=RESC(Lnew-LSER+1:Lnew);
        SERINT=sum(xor(SERSHC,SERSH));
        SHLSEQ=[SHLSEQ,0];
        if (SERINT~=0) ci=ci+1; end;
```

```
            end; %while (RESC~=SER)
            if (SHLSEQ(ci+1)==1) SHLSEQ(ci+1)=0;
                else SHLSEQ(ci+1)=1; end;
            Vout=SHLSEQ(1:nv-LSER);
            ERP=Num2Str(nv-ci); SHOUT=SHLSEQ(1:nv);
            C3=Num2Str(SHOUT);
            ETO1=[ERROUTTEXT1,ERP,ERROUTTEXT2];
            ETO2=[ERROUTTEXT3,C3];
            disp(ETO1); disp(ETO2);
        end; %if(jj<LRem+1)
    end; %if(nop==2)&(TOC==1)
return
```

Результати тестування програми

```
>> f=CRC([1,0,0,0,1,1,0],2,1)
CRC is incorrect. Error in the 4-th digit detected.
Corrected code sequence is vin=1  0  0  1  1  1  0
Coded sequence is vout=1  0  1  0
f =
   1   0   1   0
>> d=CRC([1,0,1,0,0,1],1,2)
d =
   1   0   1   0   0   1   0   0   1   0
>> f=CRC(d,2,2)
CRC is correct. Coded sequence is vout=1  0  1  0  0  1
f =
   1   0   1   0   0   1
>> d(3)=0;
>> f=CRC(d,2,2)
CRC is incorrect. Error in the 8-th digit detected.
```

```
Corrected code sequence is vin=1 0 1 0 0 1 0 0 1 0
f =
     1   0   1   0   0   1
>>
>> g=CRC([1,0,1,0,0,1,1],2,1)
CRC is correct. Coded sequence is vout=1  0  0  1
g =
     1   0   0   1
>> d=CRC([1,0,0,0,1,1,0,1,1],1,2)
d =
     1  0  0  0  1  1  0  1  1  0  1  0  0
>> f=CRC(d,2,2)
CRC is correct. Coded sequence is vout=1 0 0 0 1 1 0 1 1
f =
     1   0   0   0   1   1   0   1   1
>> d(4)=1
d =
     1   0   0   1   1   1   0   1   1   0   1   0   0
>> f=CRC(d,2,2)
CRC is incorrect. Error in the 10-th digit detected.
Corrected code sequence is vin=1  0  0  0  1  1  0  1
1  0  1  0  0
f =
     1   0   0   0   1   1   0   1   1
>> d=CRC(CRC([1,0,0,0,1,1],1,2),2,2)
CRC is correct. Coded sequence is vout=1  0  0  0  1  1
d =
    1   0   0   0   1   1
>> f=CRC(CRC(CRC([1,0,0,0,1,1],1,2),2,2),1,2)
CRC is correct. Coded sequence is vout=1  0  0  0  1  1
f =
     1   0   0   0   1   1   1   1   1   1
>>
```