ЛАХНО В.А., ГУСЄВ Б.С., КАСАТКІН Д.Ю.

КОМП'ЮТЕРНА ЛОГІКА

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

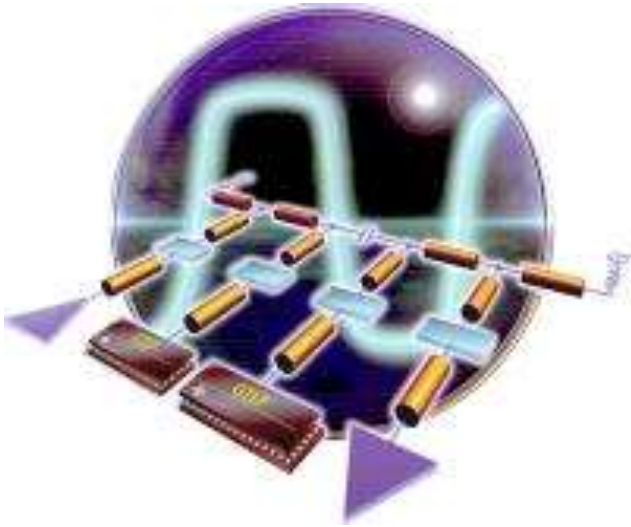
КОМП'ЮТЕРНА ЛОГІКА

Київ
2018

НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Лахно В.А., Гусєв Б.С., Касаткін Д.Ю.

КОМП'ЮТЕРНА ЛОГІКА



КИЇВ – 2018

УДК 004.312.22(004.222)
ББК 32.973.
Л 297

Рекомендовано Вченою радою Національного університету біоресурсів і природокористування України
(протокол 2 від 26.09.2018 р.)

Р е ц е н з е н т и :

Смірнов О.А. - д. т. н., проф., завідувач кафедри кібербезпеки та програмного забезпечення Центрального українського національного технічного університету

Казмірчук С.В. д.т.н., доцент, завідувач кафедри комп'ютеризованих систем захисту інформації, професор кафедри безпеки інформаційних технологій, ННІПДС

Малюков В.П. д. ф.-м. н. , доцент, професор кафедри інформаційних систем та математичних дисциплін ПВНЗ «Європейський університет»

Л 297

Комп'ютерна логіка: навчальний посібник / Лахно В.А., Гусєв Б.С., Касаткін Д.Ю. – Київ, вид-во: КОМПРІНТ, 2018. – 422.

ISBN

Дисципліна «Компютерна логіка» є однією з базових у системі підготовки бакалаврів за галуззю знань «Інформаційні технології» з спеціальностей «Компютерна інженерія» та «Компютерні науки». Пропонований навчальний посібник знайомить читача з методами подання чисел в інформаційних системах, алгоритмами виконання основних арифметичних і логічних операцій з числами в різних системах числення, основами математичної логіки, аналізу, синтезу цифрових операційних і управляючих автоматів. У навчальний посібнику викладено теоретичні й практичні аспекти проектування й застосування цифрових автоматів. На конкретних прикладах описано методику проектування логічних схем, зокрема для систем захисту інформації з обмеженим доступом.

Навчальний посібник також буде корисний студентам, аспірантам які вивчають курси «Комп'ютерна логіка», «Комп'ютерна схемотехніка», «Архітектура інформаційних систем» спеціальності «Комп'ютерні системи та мережі».

З М І С Т

В С Т У П

Цифрові автомати є типовим і найбільш поширеним видом інформаційних систем, які призначені для обробки цифрової інформації. Для успішного вивчення загальних принципів обробки цифрової інформації доцільно відволіктися від реального апаратного забезпечення інформаційної системи й розглядати її як певний абстрактний цифровий автомат, призначений для обробки інформації, представленої у цифровій формі. Знання з прикладної теорії таких автоматів необхідні для успішного пошуку нових принципів побудови інформаційних систем, удосконалення вже відомих алгоритмів обробки цифрової інформації, експлуатації обчислювальної техніки й розробки різноманітного програмного забезпечення, при цьому необхідні чіткі знання арифметичних і логічних основ цифрових автоматів, принципів їх аналізу й синтезу.

У навчальний посібнику послідовно розглянуто найбільш загальні поняття з інформаційних основ цифрових автоматів та використання цифрових пристроїв у системах захисту інформації з обмеженим доступом:

- поширені системи числення й форми подання чисел у цифрових автоматах;

- принципи здійснення арифметичних дій з двійковими й двійково-десятковими числами та організації контролю роботи цифрового автомата;

– основи алгебри логіки, методи спрощення й мінімізації логічних функцій;

– методи аналізу й синтезу логічних електронних схем;

– загальні принципи розробки програмних алгоритмів й алгоритмів апаратної реалізації арифметичних дій у цифрових автоматах.

У навчальний посібнику зроблено акцент на прикладний аспект теорії цифрових автоматів, наведені приклади використання цифрових пристроїв у системах захисту інформації з обмеженим доступом, розглянуті принципи перетворення й обробки цифрової інформації, сучасні методи аналізу й синтезу цифрових автоматів. Навчальний матеріал супроводжено численними прикладами реального застосування прикладної теорії цифрових автоматів для розв’язання конкретних задач, пов’язаних з методами аналізу й синтезу логічних електронних схем, принципів розробки програмних алгоритмів у цифрових автоматах.

Розділ 1. ІНФОРМАЦІЙНІ ОСНОВИ ЦИФРОВИХ АВТОМАТІВ

1.1. Інформація та її характеристики

Обчислювальна машина – це фізична система, призначена для автоматизації процесу алгоритмічної обробки інформації. Таким чином, поняття *„обчислювальна машина”* найтісніше пов’язане з поняттями *„інформація”* та *„алгоритмічна обробка”* [1,6,7,8,10, 14, 21, 24, 27 та ін.].

Об’єкт передачі й перетворення в обчислювальних системах (машинах) – інформація. У цьому розумінні обчислювальну машину (систему) можна назвати інформаційною.

Інформація – це сукупність відомостей, отриманих від навколишнього середовища (вхідна інформація), виданих у навколишнє середовище (вихідна інформація) або збережених усередині певної системи (внутрішня інформація).

Інформацію подають у вигляді креслень, рисунків, тексту, звукових і світлових сигналів, енергетичних імпульсів і т. ін. і передають сигналами різної фізичної природи через лінії зв’язку джерела з приймачем.

Алгоритмічна обробка – це обробка інформації відповідно до заздалегідь розробленого алгоритму.

Алгоритм – сформульоване певною мовою правило (правила), що визначає дії, послідовне виконання яких приводить від початкових відомостей до певного результату.

Отже, специфіка інформаційних процесів полягає не лише в передачі інформаційних повідомлень через задане фізичне середовище, але й у перетворенні, обробки й зберіганні інформації.

Наявність інформації обумовлює дію ряду процесів в інформаційних системах. У найзагальній формі процес розв’язання задачі у інформаційних системах має наступні етапи:

1) введення інформації або установка початкових даних;
2) обробка або перетворення введеної інформації;
3) визначення результатів і виведення обробленої інформації.

Важливе питання теорії передачі й перетворення інформації – визначення міри, кількості та якості інформації.

Інформаційні міри зазвичай розглядають у трьох аспектах: структурному, статистичному й семантичному.

Сучасні інформаційні системи можуть розв'язувати широке коло різнопланових задач. Для цього лише потрібно за допомогою програми „навчити" інформаційну систему алгоритму розв'язання тієї або іншої задачі й ввести в неї вихідні відомості. Програму ж записують алгоритмічною мовою (наприклад, Паскаль, С++ або Бейсик), яка є достатньо близькою до природної мови (особливо англійської).

Однак інформаційна система не розуміє не лише природної мови, але й алгоритмічної. Для розшифровки тексту програми, написаною мовою Паскаль або С++, у обчислювальній машині повинна міститися спеціальна програма – компілятор (транслятор), яка перекладає текст вихідної програми з С++ або Паскаля мовою обчислювальної машини. Таким чином, *обчислювальна машина* – це технічний пристрій, у якому інформація про початкові відомості розв'язуваної задачі, правила її розв'язання (алгоритм) і результати обчислень повинна задаватися у вигляді зміни якихось фізичних величин:

– намагніченості матеріалу (наприклад, для відтворення мелодії за допомогою магнітофону або збереження відомостей про атестацію студентів з дисциплін навчального плану на магнітному диску);

– освітленості екрана (дисплея) та ін.

Раніше, коли було невідомо про електричні та магнітні явища, найбільш доступною і зручною була механічна форма подання інформації в обчислювальних приладах. В арифмометрах операції над числами виконувалися за допомогою коліс, які при додаванні одиниці поверталися на колі з кроком 36° і за допомогою шрифту приводили в рух наступне за старшинством колесо кожного разу, коли цифра 9 переходила до цифри 0 (накопичувався десяток). Однак механічні пристрої громіздкі, дорогі й інерційні (з їх допомогою не можна побудувати універсальні й швидкодіючі обчислювальні машини). Тому зараз у всіх обчислювальних машинах основною формою подання інформації слугують електричні сигнали (найчастіше – напруги постійного струму). Обчислювальні машини винайшли достатньо давно, у ті далекі часи про електроніку навіть не згадувалося. Перші електронні обчислювальні машини були ламповими й займали дуже багато місця. Однак саме тоді було закладено основні принципи їх роботи, які діють до цього часу. Суть їх полягає в наступному. Відомості передаються за допомогою якогось сигналу методом „є сигнал або немає" або, по-іншому, „ввімкнений чи вимкнений". Вісім біт об'єднуються в байт, один байт дорівнює восьми бітам. Чому саме вісім? Тому, що перші обчислювальні машини були восьмирозрядними й могли працювати одночасно тільки з вісьмома бітами, наприклад, 010000111. Усі перші нулі можна видаляти, тому число 010000111 можна записати як 10000111. Це те саме, що й у звичній для нас десятковій системі числення, де кожен розряд може приймати значення від 0 до 9. Тут також ніхто не писатиме число 6743 як 00006743.

В 1 байт можна записати будь-яке число від 0 до 255. Указаний діапазон чисел доволі малий. Тому частіше використовують більші градації:

– два байти = слово;

– два слова = подвійне слово.

1.2. Структурна міра інформації

У цьому підрозділі буде розглянуто будову масивів інформації та їх зміну простим підрахунком інформаційних елементів або комбінаторним методом.

Інформацію завжди подають у вигляді повідомлення. Елементарна одиниця повідомлення – символ. Символи, зібрані в групу, – слова.

Повідомлення, оформлене у вигляді слів або окремих символів, завжди передається в матеріально-енергетичній формі (електричний сигнал, світловий та ін.).

Розрізняють інформацію неперервну й дискретну.

При структурному підході розрізняють *геометричну*, *комбінаторну* й *адитивну* міру інформації.

Геометрична міра передбачає вимір параметра геометричної моделі інформаційного повідомлення (довжина, площа, об'єм) у дискретних одиницях.

Максимально можлива кількість інформації в заданих структурах визначає інформаційну ємність моделі (системи), яка визначається як сума дискретних значень за всіма вимірами (координатами). У *комбінаторній* мірі кількість інформації визначається як кількість комбінацій (елементів). Можлива кількість інформації збігається з кількістю можливих поєднань, перестановок і розміщень елементів. Відповідно до *адитивної* міри (міри Хартлі) кількість інформації вимірюють у двійкових одиницях – бітах. Вводяться поняття глибини *„q"* числа й довжини *„n"* числа. *Глибина числа „q"* – кількість символів (елементів), прийнятих для подання інформації (це основа системи числення). У кожен момент часу реалізується один якийсь символ. *Довжина числа „n"* – кількість позицій, необхідних і достатніх для подання чисел заданої величини. При заданих *„q"* і *„n"* кількість різноманітних відображуваних станів $N=q^n$. Величина N не зручна для оцінки інформаційної ємності. Тому вводиться логарифмічну міру, яка дозволяє обчислювати кількість інформації [13, 21, 23, 34, 35]:

$$I(q)=\log N=n\log q.$$

Отже, 1 біт інформації відповідає одній елементарній події, яка може відбутися чи не відбутися. Це дозволяє оперувати мірою як числом.

Кількість інформації при цьому еквівалентна кількості двійкових символів „0” або „1”.

За наявності кількох джерел інформації загальна кількість інформації [13, 14, 23, 34, 35]:

$$I(q, q, ..., q\ ) = I(q\ ) + I(q\ ) + ... + I(q\ ).$$

Процес передачі інформації є випадковим процесом, тому що зміст повідомлення, яке передається, заздалегідь не відомий. Випадковий характер мають і перешкоди, що впливають на процес передачі інформації. Тому вивчення закономірностей передачі й перетворення інформації здійснюється методами теорії ймовірності та математичної статистики, а теорію інформації іноді називають статистичною теорією передачі повідомлень.

Разом з тим, теорію інформації часто розглядають і більш вузько – як теорію міри кількості інформації й кодування. Уведення чисельної міри кількості інформації дозволяє:

– порівнювати різноманітні повідомлення за їх змістовністю;

– оцінювати швидкість передачі інформації в різноманітних системах;

– порівнювати ці системи за ефективністю;

– визначати граничну кількість інформації, яка може бути передана в конкретних умовах.

Потреба в передачі інформації виникає лише тоді, коли досліджувана система може мати кілька випадкових станів. Передбачено, що ймовірності перебування системи в цих станах відомі апріорно.

Про перебування системи в тому чи іншому стані визначається повідомленням, що надійшло. Повідомлення про подію, поява якої заздалегідь точно відомо, не містить інформації.

З двох повідомлень, одне з яких містить результати двох можливих наслідків (кидок монети), а інше – результати шести можливих наслідків (кидання гральної кості), останнє містить більше інформації.

У першому випадку невизначеність наслідку була порівняно малою, тому ще до отримання повідомлення про те, що, наприклад, випав „герб” з ймовірністю $P=1/2$ можна передбачити появу наслідку.

У другому випадку $P=1/6$ невизначеність була значно більшою, і тому йдеться про отримання повідомлення, яке більшою мірою містить елемент несподіваності, новизни.

Таким чином, кількість інформації, наявної в повідомленні, може бути кількісно оцінена за ймовірністю появи цього повідомлення.

Такий критерій оцінки дозволяє встановити об’єктивну чисельну міру кількості інформації, що міститься в будь-яких можливих повідомленнях, незалежно від їх конкретного змісту.

Відповідно до викладеного вище вимоги, яким має відповідати міра кількості інформації, можна сформулювати таким чином:

1. Кількість інформації, що міститься в цьому повідомленні, має бути визначено не його конкретним змістом, а лише степенем невизначеності, який знімається після отримання цього повідомлення.

2. Кількість інформації повинна дорівнювати нулю, якщо подія, яка нас цікавить, має лише один наслідок.

3. Має бути дотриманий принцип адитивності, тобто кількість інформації, що міститься в цьому повідомленні, повинна бути пропорційною його довжині.

У випадку рівноймовірних подій (повідомлень) за чисельну міру кількості інформації приймають логарифм оберненої ймовірності кожного з них. Якщо скористатися двійковим логарифмом і отримати $N=2$, то $I=log2=1$.

За одиницю кількості інформації прийнято вважати таку кількість інформації, яка знімає невизначеність у виборі одного з двох рівноймовірних наслідків. Ця одиниця називається двійковою одиницею або бітом. У загальному випадку отримання нерівноймовірних повідомлень невизначеність появи конкретного (i-го) повідомлення характеризується його ймовірністю P_i.

Інформацію, оброблювану інформаційною системою, зображують відповідними символами. Інформація має різний характер – від чисел до музики.

Семантичними мірами інформації прийнято вважати:

– змістовність;

– логічну кількість;

– доцільність.

Змістовність події i через функцію міри $m(i)$ – змістовність її заперечення.

Логічні функції істинності $m(i)$ й хибності $m(i)=1-m(i)$ мають формальну подібність з функціями ймовірності подій $P(i)$ і $q(i)=1-P(i)$. Логічну кількість інформації I обчислюють за формулою:

$$I=log(1/m(i))=-log\ m(i).$$

Відмінність статистичної оцінки від логічної полягає в тому, що в першому випадку враховують імовірності реалізації тих або інших подій, що наближає до оцінки змісту інформації. Міру доцільності інформації визначають як зміну ймовірності досягнення мети після отримання додаткової інформації.

$$I_{цел}=logP_н-logP_к=logP_н/P_к.$$

де $P_н$, $P_к$ – початкові й кінцеві ймовірності досягнення мети.

1.3. Абстрактні автомати

Класичними прикладами абстрактних автоматів є машини Тьюрінга й Поста [1, 8, 18, 22, 31, 32].

Ідея машини Поста базується на таких постулатах:

1) алгоритмічні процеси є послідовністю достатньо простих „механічних" операцій;

2) на результат не впливає, хто виконує алгоритм і скільки разів він повторюється, – результат завжди той самий;

3) таким чином, алгоритмічні процеси може виконувати машина.

У гіпотетичній машині Поста (1936) інформацію подають у двійковому алфавіті *A={0,1}*. Машина має інформаційну стрічку необмеженої довжини – пам'ять машини. У кожному гнізді може розташуватися 0 або 1. Машина має „зчитувальну головку" (спеціальний чутливий елемент), яка визначає вміст гнізда *(j)*, див. рис. 1.1.

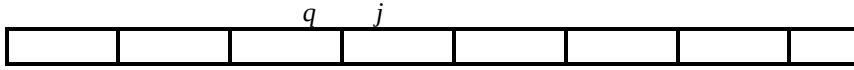


Рис. 1.1. Схема машини Поста

Інформаційна стрічка може переміщатися в обидва боки, таким чином, що в кожен момент часу головка знаходиться навпроти певного гнізда.

Машина має керуючий пристрій, який у кожен момент часу перебуває в певному стані – *q*. Стрічка переміщається дискретно так, щоб навпроти головки зупинялося гніздо.

Набір команд абстрактного автомата:

1 – головку пересунути вправо;

2 – головку пересунути вліво;

3 – записати мітку;

4 – стерти мітку;

5 – передати управління;

6 – стоп.

Для складання програми необхідно скласти список послідовно виконуваних команд. Це і є програма. Кожна команда має власний номер – *i*. Стрілка вказує напрям руху (головки чи стрічки). Друге число, яке стоїть у кінці команди, називається відсилкою – *u*. Тому програма абстрактного автомата повинна мати дві властивості:

1. На першому місці в списку команд завжди стоїть команда з номером „1", на другому – з номером „2" і т. д.

2. Проводиться запуск (початок дії) будь-якої з команд, яка міститься у програмі в списку команд. Після пересування стрічки (головки)

вліво чи вправо головка зчитує стан секції (гніздо порожнє або записана мітка).

Інформація про те, які секції порожні, а які відмічені, утворює стан стрічки або стан автомата. Таким чином, маючи вказаний вище набір команд, автомат може здійснювати певні дії, які задаватиме програма.

Програмою абстрактного автомата будемо називати скінчений не порожній список команд. Для роботи абстрактного автомата необхідно задати програму й початковий стан, тобто положення головки й стан гнізд стрічки.

Наприклад, програма для машини Поста має такий вигляд:

Рухається головка	**Номер команди,** *i*	**Відсилка,** *u*	
вправо 1 крок	1	3	виконати команду № 3
вправо 1 крок	2	4	виконати команду № 4
запис мітки	3	2	виконати команду № 2
Команда передачі управління	4	5	
Стоп			

Після цього автомат переходить до виконання команди № 1. Усі секції (гнізда) нумеруються в певному порядку. Порядок нумерації може збігатися з порядком, у якому розташовано натуральні цілі числа. Кожна команда виконується за один крок, після чого починається виконання команди, номер якої вказано у запуску. Якщо ця команда має два запуска, то команда з номером верхнього запуску виконується, якщо під головкою знаходиться порожнє гніздо. Якщо ж під головкою знаходиться гніздо з міткою, то виконується команда з номером нижнього запуску. Виконання команди передачі управління не змінює станів автомата (жодна з міток не знищується й не ставиться, і стрічка залишається нерухомою).

Під час запуску автомата може виникнути одна з таких ситуацій:

1. Автомат дійшов до команди, яку не можна виконати (запис мітки в зайняте гніздо, стирання мітки в порожньому гнізді). Виконання програми припиняється, автомат зупиняється, відбувається безрезультатна зупинка.

2. Автомат дійшов до команди „*стоп*”, програма вважається виконаною, відбувається результатна зупинка.

3. Автомат не доходить ні до результатної, ні до безрезультатної зупинки; відбувається нескінченна робота (автомат „завис”).

Машина Тьюрінга (1937) відрізняється від машини Поста тим, що алфавіт може мати більше двох символів. Крім того, розширено перелік операцій, наприклад, додати команди зміни змісту гнізда перед просуванням стрічки [32].

Припустімо, що алфавіт машини Тьюрінга [32]:

$$A=(S_0,S_1,\dots S_n),$$

де S_0 – відмінне від 0.

Стан машини – множина [1, 8, 24, 27]:

$$Q=\{q_0, q_1,\dots q_m\},$$

де q_0 – заключний стан;
A – зовнішній алфавіт машини;
Q – внутрішній алфавіт машини.

У кожен момент часу машина має певну послідовність станів гнізд і один стан управляючого пристрою, а зчитувальна головка знаходиться навпроти якогось гнізда. Сукупність усіх елементів, які визначають стан машини, називають її конфігурацією (див. рис. 1.2).

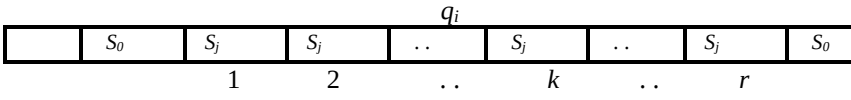


Рис. 1.2. Схема машини Тьюрінга

Конфігурація машини може мати такий вигляд:

$$\dots S_0S_{j1}S_{j2}S_{j3}\dots q_jS_{jk}\dots S_{jr-1}S_{jr}S_0\,,$$

де S_0 – порожнє гніздо;
S_{j1} – стан (вміст) першого не порожнього лівого гнізда;
S_{jk} – стан гнізда, яке розглядається в певний момент часу;
r – кількість зайнятих гнізд;
q_i – стан управляючого пристрою; $i=0,1\dots m$.

Якщо машина перебуває в стані q_i, має доступ до гнізда S_k, переходить у стан q_j, замінюючи символ S_k на S_m, а стрічка переміщається вліво на одне гніздо, це означає, що вона виконує команду

$$q_iS_k\rightarrow q_jS_mЛ \quad \text{или} \quad q_iS_kq_jS_mЛ.$$

Замість ***Л*** можуть стояти ***П*** (правий, тобто стрічка рухається вправо) або ***С*** (стоп, тобто стрічка зупиняється).

Таким чином, перехід від одної конфігурації до іншої здійснюється за допомогою команди, яка має такий загальний вигляд:

$$q_iS_k\rightarrow q_jS_mT,$$

де $T=Л, П, С$.

Сукупність усіх команд машини Тьюрінга називається її програмою.

Для того щоб задати машину Тьюрінга, необхідно задати її внутрішній і зовнішній алфавіт, програму, початкову конфігурацію, якими символами позначено порожнє гніздо й кінцевий стан управляючого пристрою.

Програму машини Тьюрінга можна задати у вигляді таблиці (див. табл. 1.1).

Машина Тьюрінга, представлена в табл. 1.1, здійснює перетворення слова 1*0 в слово 0*1. Наведемо алгоритм переносу 0 разом зі зміною конфігурації машини після виконання кожної з команд програми.

Таблиця 1.1

Приклад програми для машини Тьюрінга

Q	0	1	*	s_0
q_2	-	q_2 0 Л	q_1 * Л	Q_0 С
q_1	q_2 1 Л	-	-	-
q_0	Зупинка			

Припустімо, що початкова конфігурація машини має вигляд: $s_0q_2$1*0s_0.

Тоді

q_1 1→ q_2 0 Л s_0 0 q_2*0 s_0
q_2 *→ q_1 * Л s_0 0 * q_1 0 s_0
q_1 0→ q_1 1 Л s_0 0 * 1 q_2 s_0
q_2 s_0→ q_0 s_0 С s_0 0 * 1 q_0 s_0 - заключна конфігурація машини

З часом з'явилися різноманітні модифікації машин Тьюрінга, залежно від кількості використовуваних стрічок пам'яті, станів управляючого пристрою та ін., наприклад, існує машина Тьюрінга з двома виходами.

1.4. Інформація, інформаційні сигнали і методи їх перетворення

Інформація – поняття, яке має широке різнопланове відображення і глибокий зміст. В якості прикладів термінології такого відображення можна навести: інформаційні системи; інформаційні технології; інформаційні процеси та інше. Функціонування та управління технічних, економічних, біологічних, соціальних та інших систем базуються на використанні різного виду інформаційних процесів. У більшості випадків при дослідженнях технічних систем, до яких належать цифрові автомати, виділяють наступні інформаційні процеси:

- збирання та прийом;
- передача;
- обробка;
- збереження та відображення інформації.

Можна констатувати, що на сьогодні чіткого і формалізованого загального визначення інформації не існує. Але в конкретному випадку можна сформулювати поняття інформації, адаптоване до науково-технічної проблематики цифрових автоматів в наступній редакції.

Інформація – це відомості про відповідні властивості, характеристики, параметри досліджуваних явищ та об'єктів у просторі і часі, їх взаємозв'язки між собою.

Інформація, яка втілена і формується у досліджуємому матеріальному середовищі іменується *повідомленням* або *даними спостережень (вимірювань)*, а фізичний засіб передачі повідомлення у просторі і часі – *сигналами* або *інформаційними сигналами.*

Комп'ютерна логіка є науково-технічним напрямом досліджень загальної теорії сигналів і систем, яка має свою специфіку і широке коло предметних областей її використання.

У даному підрозділі наведено лише короткі відомості про інформаційні сигнали і методи їх перетворення для подальшого використання в цифрових автоматах.

Відомо [5, 7, 8, 21 та ін.], що сучасні інформаційні системи є апаратно-програмними комплексами. В цифрових автоматах апаратною підсистемою є сам цифровий автомат, а програмна підсистема складається з наступних видів забезпечення, які називають «м'яким обладнанням»:

- інформаційного;
- математичного;
- програмного.

Інформаційне забезпечення є сукупністю баз даних значень вимірювань інформаційних сигналів з відповідними системами управління (СУБД), інтерфейсних каналів, засобів захисту інформації від несанкціонованого доступу, засобів управління набору даних та інших засобів.

Математичне забезпечення є сукупністю математичних моделей досліджуємих інформаційних сигналів, методів і алгоритмів визначення їх характеристик та параметрів, на базі використання яких розв'язуються поставлені задачі досліджень.

Програмне забезпечення реалізує математичне забезпечення і складається з *системного* і *функціонального (прикладного)* програмного забезпечення.

Більш детального розглянемо інформаційні сигнали, які є основними об'єктами досліджень інформаційних систем.

Фізична природа формування сигналу у більшості випадків є стохастичною, випадковою і згідно класифікації А.М. Колмогорова можливі три варіанти:

- сигнал може бути описана детермінованими моделями, які однозначно визначаються початковими умовами, але самі початкові умови є випадковими;
- сигнал як функція часу і (або) просторових змінних є випадковою, тобто описується випадковим процесом або випадковим полем;
- часова динаміка сигналу описується різними комбінаціями, в більшості випадків адитивною або мультиплікативною сумішшю детермінованих сигналів і випадкових процесів.

Наведені варіанти випадковості сигналу по суті визначають як сам сигнал, так і їх можливі більш складні форми – комбінації сигналів.

При класифікації фізичної природи і каналів розповсюдження виділяють наступні інформаційні сигнали:

- електричні;
- електромагнітні;
- акустичні;
- вібраційні;
- оптичні;
- коливання земної кори;
- потоки ядерних частин;
- біофізичні;
- інші сигнали.

Важливу роль при дослідженнях інформаційних сигналів відіграють їх математичні моделі.

Під математичною моделлю інформаційного сигналу розуміють сукупність знань, припущень, гіпотез, умов, побудованих у вигляді цілісної, логічно витриманої і несуперечливої структури, яка *гомоморфно* відображає основні властивості та характеристики досліджуємої інформації, взаємозв'язок, взаємодію та відношення між компонентами інформації, записана з використанням математичних об'єктів, символів і призначена для розв'язання відповідного класу задач.

Відомо, що для однієї інформації в залежності від постановки задач досліджень можуть використовуватись різні математичні моделі. В той же час одна математична модель може описувати інформацію про різні об'єкти.

Математична модель інформаційного сигналу є динамічною структурою і може змінюватись по мірі накопичення знань про досліджувану інформацію.

Наведемо класифікацію інформаційних сигналів, які є основними об'єктами досліджень цифрових автоматів (див. рис. 1.3).

Розглянемо більш детально клас цифрових інформаційних сигналів, які є об'єктом обробки сигналів в цифрових автоматах.

В більшості випадків первинні інформаційні сигнали є аналоговими, тобто неперервними як в області визначення, так і в області значень. За допомогою використання аналогово-цифрових перетворювачів (АЦП) аналогові сигнали перетворюються в цифрові. Розглянемо більш детально операції такого перетворення сигналів.

Заданий аналоговий сигнал $\{U(t), t \in T\}$, областю визначення якого є неперервний інтервал часу Т. Областю значень сигналу $U(t)$ є також неперервний інтервал часової вісі $R = (-\infty, \infty)$. Таким чином множини значень областей визначення і значень сигналу $U(t)$ є нескінченними і мають потужність *континуума*.

1. Перша операція перетворення аналогових сигналів в АЦП іменується *дискретизацією по часу* і зводиться до формування послідовності відліків сигналу $U(t) \Rightarrow \{U(t_j), j = \overline{1,n}\}$.

Відомо, що при обґрунтуванні вибору значення кроку дискретизації по часу $\Delta t = h$ сигналу $U(t)$ використовується так звана теорема відліків (яка відома також під назвою теореми Уіттекера – Найквіста – Котельнікова – Шеннона). Згідно цієї теореми для досліджуємого сигналу $U(t)$ виконуються наступні умови, а саме:

- умови Діріхле (обмеження значень, скінченне число екстремумів, абсолютна інтегрованість);

- фінітний спектр (перетворення Фур'є сигналу) обмежений частотою $F_{â}$.

Рис. 1.3. Класифікація інформаційних сигналів

Тоді має місце розклад сигналу $U(t)$ в нескінченний інтерполяційний ряд виду

$$U(t) = \sum_{k=-\infty}^{\infty} U(kh) \frac{\sin 2\pi F_{\hat{a}}(t-kh)}{2\pi F_{\hat{a}}(t-kh)}.$$

Такий ряд також можна інтерпретувати як розклад сигналу $U(t)$ в нескінченний ортогональний ряд по ортогональному базису виду

$$\left\{ \frac{\sin kt}{kt}, k \in Z \right\}.$$

Особливість такого розкладу сигналу $U(t)$ в ортогональний ряд полягає в тому, що в якості коефіцієнтів такого ряду використовуються відліки сигналу $U(t)$ на рівномірній нескінченній часовій гратці

$$\ldots,-kh,\ldots,-3h,-2h,-h,0,h,2h,3h,\ldots,kh,\ldots$$

Крок по часу $\Delta t = h$ визначається виразом

$$h = \frac{1}{2F_{\hat{a}}},$$

де $F_{\hat{a}}$ - верхня частота перетворення Фур'є вхідного сигналу. З умови теореми відліків перетворення Фур'є сигналу $U(t)$, як було визначено раніше, є фінітною функцією частоти, тобто

$$S(f) = \int_{-\infty}^{\infty} e^{-i2\pi ft} U(t)dt = \begin{cases} S(f), f \in [-F_в, F_в] \\ 0, f \notin [-F_в, F_в] \end{cases}.$$

Умови, при яких має місце теорема відліків, на практиці виконується не в повній мірі, тому при практичній реалізації теореми відліків використовуються відомі в теорії сигналів практичні рекомендації. В залежності від моделі інформаційного сигналу на практиці крок дискретизації вибирається по формулі

$$h = \frac{1}{(5...20)F_{\hat{a}}*},$$

де значення $F_{\hat{a}}*$ також обґрунтовується на практиці.

Число відліків досліджуємого сигналу $U(t)$ вибирається скінченним

$$U(t) \Rightarrow \{U(t_j),\ j = \overline{1,n}\}.$$

Таким чином, в результаті операції дискретизації по часу аналогового сигналу $U(t)$ отримуємо:

- скінченну множину відліків сигналу $\{U(t_j),\ j = \overline{1,n}\}$;
- множина значень сигналу $\{U(t_j)\}$ залишається нескінченною потужністю континуума.

2. Друга операція перетворення дискретизованих по часу значень сигналу $\{U(t_j)\}$ в АЦП іменується *квантуванням значень сигналу по рівню* і зводиться до формування послідовності квантованих по рівню відліків сигналу

$$\{U(t_j)\} \Rightarrow \{\hat{U}(t_j),\ j = \overline{1,n}\}.$$

Відомо, що при обґрунтуванні вибору значення кроку квантування рівня сигналу ΔU задається похибка квантування і відповідно вибирається розрядність АЦП. Так, наприклад, для 12-розрядного АЦП похибка квантування при $\max U(t_j) = 5B$ визначається за формулою

$$\frac{\Delta U}{2} = \frac{5}{4096} \approx 1{,}2 \text{ìÂ} \ .$$

Таким чином, в результаті операції дискретизації по часу і квантування по рівню сигналу $U(t)$ в АЦП отримуємо:

- скінченну множину відліків сигналу $\{\hat{U}(t_j),\ j = \overline{1,n}\}$;
- множина квантованих по рівню значень сигналу $\{\hat{U}(t_j)\}$ також є скінченною.

3. Третя операція перетворення дискретизованих по часу і квантованих по рівню значень сигналу $\{\hat{U}(t_j)\}$ іменується *кодуванням* отриманих значень сигналу $\{\hat{U}(t_j)\}$ у відповідну систему числення, як правило, двійкову.

Таким чином отримуємо *цифровий сигнал* $\{z_j,\ j = \overline{1,n}\}$.

При перетворенні аналогового сигналу $\{U(t), t \in T\}$ в цифровий сигнал $\{z_j, j = \overline{1,n}\}$ шляхом використання АЦП вносяться відповідні похибки, які необхідно враховувати при цифровій обробці сигналів.

1.5. Предметна область використання цифрових автоматів – цифрова обробка сигналів

Значне коло задач цифрової обробки сигналів (ЦОС), які розв'язуються з використанням цифрових автоматів, обумовлює необхідність в короткій редакції окреслити науково-технічну проблематику ЦОС.

Це наступні приклади задач ЦОС:

- цифрової фільтрації сигналів, при цьому структури і характеристики різних цифрових фільтрів реалізуються програмними засобами;
- виявлення корисних сигналів при дії завад в задачах геофізики, радіофізики, радіолокації та гідроакустиці;
- реалізації методів кореляційної і спектральної обробки сигналів;
- статистичної обробки часових рядів в різних галузях науки і техніки, економіки, соціології та інших;
- розпізнавання образів при обробці мовних, сейсмічних сигналів;
- статистичної обробки даних вимірювань сигналів функціонування різних технічних об'єктів контролю та діагностики;
- статистичної обробки даних вимірювань і формування сигналів управління складними технічними об'єктами;
- створення відповідних баз даних результатів функціонування різних об'єктів:
- створення інформаційного, математичного та програмного забезпечення функціонування апаратно-програмних комплексів у різних галузях науки і техніки.

Комп'ютерні системи ЦОС – це системи реального часу, які призначені для розв'язання задач приймання, обробки, зменшення надлишковості та передавання інформації в реальному часі. Сучасний етап розвитку теорії ЦОС тісно зв'язаний з інтенсивним провадженням методів цифрових сигналів, які орієнтовані на застосування однокристальних мікропроцесорів ЦОС, архітектурно перепрограмованих надвеликих інтегральних схем (НВІС) на базі програмованих логічних інтегральних

схем (ПЛІС) і багатопроцесорних систем, побудованих на їх основі. Більш детально ці питання розглянуті у наступних розділах.

Після введення інформації в цифровий автомат здійснюється її обробка цифровими методами із застосуванням відповідного програмного забезпечення. Розглянемо основні способи обробки інформації, яка поступає від зовнішніх приладів у цифровий автомат.

*Компенсація дрейфу***.** Дрейфом називається хаотичний, непередбачуваний вихідний сигнал аналогових пристроїв (датчиків, операційних підсилювачів, тощо) за умови, що вихідний сигнал цього пристрою дорівнює нулю. Дрейф накладається на корисний сигнал і спотворює його. Щоб компенсувати дрейф, його періодично запам'ятовують, викликаючи на цей час вхідний сигнал, і віднімають від кожного значення сигналу.

*Перевірка достовірності сигналу***.** Щоб переконатися у достовірності сигналу, слід перевірити, чи значення сигналу не виходять за межі робочого діапазону зовнішнього датчика. Вихід значень за межі робочого діапазону свідчить про аварійну ситуацію. Крім того, потрібно перевірити, чи швидкість зміни сигналу не виходить за межі діапазону швидкостей закладених при проектуванні цифрового автомату.

*Статистична обробка сигналу***.** Сигнали, що поступили в цифровий автомат, обробляються: обчислюється середнє значення, середнє квадратичне відхилення. Значення сигналів, що різко відхиляється від середніх значень, слід відкинути.

*Цифрова фільтрація***.** Крім аналогової фільтрації, над сигналами у цифровій формі можна здійснювати цифрову фільтрацію. Цифрові фільтри, на відміну від аналогових, реалізуються програмно. Це дає змогу просто змінювати параметри фільтра, реалізувати фільтри високих порядків. Цифрові фільтри мають значно більшу стабільність характеристик ніж аналогові.

Лінеаризація. Датчики фізичних величин, що використовуються в складних цифрових системах мають здебільшого нелінійну залежність між вхідним і вихідним сигналами. Цифрові сигнали, що обробляються системою, мають бути лінійно залежними від відповідних вхідних сигналів. Щоб зробити цю залежність лінійною, необхідно здійснити нелінійне перетворення, обернене до нелінійного перетворення датчика. Така процедура називається лінеаризацією датчика.

Успіхи в теорії ЦОС, інтенсивний розвиток НВІС - технології забезпечили створення високопродуктивних малогабаритних комп’ютерних систем ЦОС, які істотно розширили галузі застосування технологій ЦОС. Перелік деяких галузей застосування засобів ЦОС, задачі та відповідні алгоритми, що використовуються для їх розв’язання, наведений в табл. 1.2.

Таблиця 1.2

Галузі застосування засобів ЦОС

№ п/п	Галузі застосування ЦОС та розв'язуванні задачі	Алгоритми і операції обробки
1	**2**	**3**
1	**Системи зв'язку та захисту інформації:** • підвищення надійності, пропускної здатності, скритності інформації, і т.д.; • виділення символів, згортка символьних послідовностей; • скорочення надлишковості; • підвищення завадостійкості; • управління рухом автомобілів, кораблів, літаків, космічних апаратів, потягів та ін.	• операції над матрицями; • логічні операції; • згортка; • швидке перетворення Фур'є комплексної послідовності (ШПФ); • повороти координат; • перетворення координат Декартові - полярна; • арифметичні операції (Ариф. опер.); • обчислення тригонометричних функцій.
2	**Радіолокація:** • контроль повітряного простору; • синтез радіозображень; • класифікація об'єктів за їх радіозображенням.	• згортка; • ШПФ; • операції над матрицями (ОМ); • перетворення координат Декартові - полярна; • обчислення тригонометричних функцій (ОТФ); • пошук максимумів; • сортування; • Ариф. опер.; • статистична обробка.

Продовження табл. 1.2

1	2	3
3	**Гідроакустика:** • картографірування і профілювання дна; • пошук корисних копалин; • контроль водного простору; • підводна сейсмологія; • гідронавігація по радіомаякам.	• згортка; • ШПФ; • ОМ; • перетворення координат Декартові - полярна; • ОТФ; • пошук максимумів; • сортування; • Ариф. опер.; • статистична обробка; • нормування.
4	**Геофізика:** • пошук нафтоносних (водоносних) шарів.	• згортка; • ШПФ; • ОТФ; • ділення; • отримання квадратного кореня; • піднесення до квадрату; • логарифмування; • потенціонування.
5	**Біомедицина:** • візуалізація органів; • діагностика.	• згортка; • ШПФ; • ОМ; • поліноміальні розклади координат; • інтерполяція даних.

Алгоритмічне забезпечення комп'ютерних систем обробки сигналів охоплює методи та алгоритми обробки сигналів і зображень. Зокрема, це методи і алгоритми кореляції, цифрової фільтрації, спектрального аналізу сигналів на базі дискретних ортогональних тригонометричних перетворень та нейромережевих технологій. На сьогодні розроблено досить потужний

математичний апарат технологій цифрової обробки сигналів і зображень, який постійно вдосконалюється, розширюється і орієнтується на апаратні НВІС - реалізації.

1.6. Обмін інформацією між різноманітними інформаційними пристроями

Обмін інформацією між різноманітними пристроями системи, які реалізують інформаційні процеси, в основному прийом і передачу інформації, забезпечується так званими інтерфейсами.

Сукупність усіх, в основному уніфікованих, технічних (апаратних) і програмних засобів, які забезпечують інформаційну взаємодію між інформаційними пристроями, називається *інтерфейсом.* Інакше кажучи, *інтерфейс* – це сукупність апаратних і програмних засобів, призначених для організації інформаційного зв'язку між інформаційними пристроями.

Інформація передається за допомогою інтерфейсу за визначеним „*протоколом*", тобто з дотриманням певних правил, характерних для цього інтерфейсу.

Розрізняють ведучий інформаційний пристрій (ініціатор) і ведений (виконавець). Ведучий ініціює обмін інформацією (відомостями), ведений передає або приймає інформацію під керівництвом ведучого. До основних інтерфейсних процедур, зокрема, належать: *арбітраж* – запит і захоплення ведучим каналу зв'язку (магістралі) інтерфейсу; адресація потрібного веденого; обмін відомостями, виконуваний за протоколом конкретного інтерфейсу.

Стандартний інтерфейс – це сукупність уніфікованих апаратних, програмних і конструктивних засобів, необхідних для реалізації взаємодії різноманітних інформаційних пристроїв за умов, визначених стандартом і спрямованих на забезпечення інформаційної, електричної й конструктивної сумісності цих пристроїв.

1.7. Апаратні засоби зберігання й обробки інформації

Основним апаратним засобом будь-якої обробки інформації є інформаційна система, призначений для обробки кодованої інформації, прийому, передачі й зберігання її під керівництвом відповідних програм.

Передача й прийом інформації в інформаційній системі здійснюється через відповідні порти**,** які з'єднують інформаційну систему за допомогою певних інтерфейсів з його периферією (дисплей, клавіатура,

принтер, сканер і т. ін.) і з необхідною апаратурою користувача. Інформація зберігається в пам'яті інформаційної системи.

Пам'ять інформаційної системи – це функціональна її частина, призначена для запам'ятовування й (або) видачі різноманітної інформації, проміжних і остаточних результатів обробки інформації.

У пам'яті, зокрема, містяться також програми вирішення певних задач обробки інформації. Усю інформацію в пам'яті інформаційної системи подано символами внутрішнього алфавіту.

Для побудови запам'ятовуючих пристроїв як фізичні елементи використовують електронні схеми, феритові магнітні матеріали, магнітні диски, оптичні диски й т. ін. Оперативний запам'ятовуючий пристрій інформаційної системи зазвичай формується з електронних схем. В оперативному запам'ятовуючому пристрої кожне інформаційне слово розміщується в так званому *гнізді*, якому присвоюється певна адреса. При відключенні інформаційної системи інформація, записана в оперативному запам'ятовуючому пристрої, зникає. При включенні інформаційної системи відбувається автоматичне перезавантаження операційної системи з відповідного магнітного диска в оперативному запам'ятовуючому пристрої. У пам'яті на дисках інформація групується у *файли*, які мають певні імена. Інформація, записана в пам'яті такого типу, зберігається після відключення інформаційної системи.

Обробку інформації виконує процесор, або мікропроцесор інформаційної системи, який управляє також усією апаратурою інформаційної системи в цілому, у тому числі зовнішньою апаратурою, підключеною до нього. До складу мікропроцесора зазвичай входять такі основні блоки (вузли) [3, 13, 20, 21, 30, 34]:

– арифметично-логічний пристрій;

– кілька регістрів, кожен з яких призначений для зберігання одного інформаційного слова;

– пам'ять, де розміщуються мікропрограми, які управляють функціями арифметично-логічного пристрою і регістрами мікропроцесора.

Арифметично-логічний пристрій – функціональна частина мікропроцесора, яка виконує логічні й арифметичні дії, необхідні для обробки інформації, що зберігається в пам'яті інформаційної системи.

Арифметично-логічний пристрій може виконувати такі елементарні дії, як порівняння, складання, віднімання, множення, ділення, зсуви вправо або вліво, додатне або від'ємне приростення, інверсію й цілий набір елементарних логічних операцій. Функції арифметично-логічного пристрою визначають архітектуру мікропроцесора у цілому.

Регістри мікропроцесора беруть участь у реалізації всіх функцій мікропроцесора, зокрема функцій його арифметично-логічного пристрою. Майже всі мікропроцесори мають шість основних регістрів: *стану,*

буферні, команд, адреси пам'яті або порту інформаційної системи, *лічильник команд і акумулятор.* Акумулятор призначений для зберігання слова відомостей, посланого з арифметично-логічного пристрою або пам'яті інформаційної системи. Він є головним регістром мікропроцесора у різноманітних операціях з даними. Більшість арифметичних і логічних операцій здійснюється шляхом використання арифметично-логічний пристрій й акумулятора.

Структура мікропрограм визначає склад базових команд асемблера певного мікропроцесора, кожній з яких транслятор ставить у відповідність певний код (зазвичай у двійковій системі числення). У свою чергу, кожній команді, оператору мови програмування високого рівня (Паскаль, С (Сі), С++, Фортран і т. ін.) відповідний компілятор або інтерпретатор ставить у відповідність певну групу команд асемблера.

Коли мікропроцесор вилучає за черговою адресою гнізда оперативного запам'ятовуючого пристрою інформаційної системи код команди асемблера, то за цим кодом активізується відповідна мікропрограма, яка, у свою чергу, генерує послідовність кодів команд, що керують мікропроцесором у цілому, зокрема, арифметично-логічним пристроєм і регістрами мікропроцесора.

Такий спрощений вигляд має ієрархічна структура управління роботою інформаційної системи. Отже, обробка інформації програмою будь-якої складності й будь-якого характеру зводиться в підсумку до виконання відповідної послідовності елементарних команд, виконуваних арифметично-логічним пристроєм мікропроцесора. Програмне введення й виведення інформації в усіх випадках здійснюється „крізь" мікропроцесор, через його регістри.

1.8. Задачі захисту інформації з обмеженим доступом

Суттю положень Концепції технічного захисту інформації з обмеженим доступом в Україні [11, 17] є забезпечення таких властивостей інформації як *конфіденційність, цілісність та доступність*. Пріоритетність забезпечення цих властивостей залежить від типу, приналежності та важливості інформації, а ступінь невідповідності зазначеним вимогам визначає шкоду, нанесену власнику інформації.

Вимога *доступності* інформації передбачає гарантії безперешкодного, своєчасного та на належному рівні доступу до інформації користувачам та учасникам інформаційного обміну.

Вимога *цілісності* інформації передбачає гарантії обігу інформації тільки за тими адресами та тільки в такому вигляді і з таким змістом, які вибирають учасники інформаційного обміну.

Вимога *конфіденційності* інформації передбачає гарантії збереження усіх передбачених законодавством України таємниць, включаючи особисті дані та дані про особу, комерційну та інші таємниці. Слід зважити, що в цьому випадку мова йде в тому числі і про інформацію, яка у фрагментарному вигляді не містить відомостей з обмеженим доступом, але в сукупному вигляді, накопичена у справі та за часом, може бути віднесена до інформації з обмеженим доступом.

Об'єктивна необхідність розглядати проблему технічного захисту інформації з обмеженим доступом під таким кутом зору має зовнішньополітичний, внутрішньополітичний та науково-технічний аспекти [4, 11, 17, 19 та ін.].

Науково-технічний аспект полягає в тому, що процес інформатизації, який має місце в Україні, призвів до реконструкції та обновлення ліній телекомунікацій та супроводжується широким впровадженням сучасних засобів зв'язку – здебільшого засобів цифрового зв'язку, телекомунікацій та обчислювальної техніки. В зв'язку з відсутністю на українському ринку зазначених засобів вітчизняного виробництва, що задовольняли б у повній мірі набору базових функцій у відповідності до вимог міжнародних стандартів, в Україні впроваджуються засоби виробництва іноземних фірм. Зазначені явища призвели до інтенсивного поширення в країні найсучаснішого цифрового обладнання, що, забезпечуючи необхідний рівень реалізації інформаційних технологій та поширення інформації на необмежених територіях, водночас потребує сучасних підходів до технічного захисту інформації з обмеженим доступом, в яких важливу роль починають грати проблеми забезпечення її цілісності та доступності.

За зазначених умов сьогодні в Україні виникла ціла низка проблем, для розв'язання яких необхідне формування відповідної державної політики в галузі технічного захисту інформації з обмеженим доступом.

Одним із напрямків цієї політики є необхідність створення відповідних нормативних документів для забезпечення сприятливих умов створення та розповсюдження засобів технічного захисту інформації з обмеженим доступом із достатнім рівнем ефективності для вирішення не тільки проблем технічного захисту інформації з обмеженим доступом внутрішньодержавного характеру, але також і проблем, пов'язаних з технічним захистом інформації з обмеженим доступом при обміні інформацією на міждержавному рівні.

Важливим також є вирішення проблеми різкого підвищення уразливості інформації в мережах інформаційних систем, мережах зв'язку та інформаційних системах. З однієї сторони це пов'язано з можливостями сучасного обладнання здійснювати дистанційний доступ до інформаційних та технічних ресурсів без обмежень технічного, часового та

територіального характеру, з іншої – з труднощами вивчення обладнання зв'язку та обчислювальної техніки іноземного походження, що здебільшого впроваджується в Україні. Останнє не дозволяє своєчасно вживати заходи щодо технічного захисту інформації з обмеженим доступом.

В галузі впровадження цифрових технологій передачі даних потребує вирішення проблема забезпечення контролю з боку держави за надходженням інформації із-за кордону. За відсутності такого контролю національний інформаційний простір наповнюватиметься непотрібною чи навіть шкідливою інформацією, що може порушити функціонування інформаційних систем та призвести до неефективного використання програмних та апаратурних ресурсів.

Враховуючи міжнародний досвід, регулювання вхідних та вихідних інформаційних потоків через кордон може забезпечуватись шляхом створення штучних шлюзів на їх шляху.

В Україні створено та створюється досить великий обсяг засобів захисту мовної інформації з обмеженим доступом та програмних і технічних засобів захисту інформації в обчислювальних мережах, мережах зв'язку та інформаційних системах. За станом на сьогоднішній день закінчено створення засад нормативного забезпечення системи технічного захисту інформації з обмеженим доступом. По-перше, це сукупність нормативних документів, що становитимуть основу вітчизняної нормативної бази щодо захисту інформації з обмеженим доступом у інформаційних системах, цифрових автоматичних телефонних станціях, захисту мовної інформації, захисту інформації від витоку каналами зв'язку, а по-друге сертифікація засобів забезпечення технічного захисту інформації з обмеженим доступом.

Цілком природно, що розглянуті питання не охоплюють всіх проблем розвитку технологічного забезпечення системи інформаційної безпеки. В той же час ця складова розвивається найбільш динамічно і існують реальні передумови до зниження темпів зростання потенційних загроз інформаційної безпеки України в технологічній області.

Досягнутий рівень сучасних технологій створення мікропроцесорних систем захисту інформації з обмеженим доступом по багатьох системних характеристиках інтегральних компонентів (швидкодії, габаритам, надійності) відповідає вимогам широкого класу застосувань спеціалізованих пристроїв та мікропроцесорів для оперативної обробки сигнальної інформації, яка формується, передається, обробляється, накопичується і зберігається в інформаційних системах, а також питаннях інформаційної безпеки інформаційних систем.

1.9. Етапи розвитку електронно-обчислювальної техніки та технічних служб захисту інформації з обмеженим доступом

1.9.1. Розвиток електронно-обчислювальної техніки

Історія розвитку цифрових систем захисту інформації тісно пов'язана із становленням та етапами розвитку обчислювальної техніки. Традиційно розробки обчислювальної техніки прийнято поділяти на покоління, див. табл. 1.3, при цьому основною ознакою зміни поколінь обчислювальних машин та відповідних систем захисту інформації з обмеженим доступом є зміна їх елементної бази. Слід пам'ятати, що люба класифікація не є абсолютною, оскільки завжди можливо знайти об'єкт класифікації, який по одним параметрам відноситься до одного класу, а по іншим до другого, рис. 1.4.

Таблиця 1.3

Покоління електронно-обчислювальних машин

Покоління	**Елементна база**	**Роки існування**	**Галузі застосування обчислювальної техніки**
Перше	Електронні лампи	50-60 рр. 20-го сторіччя	Науково-технічні розрахунки
Друге	Транзистори, феритові сердечники	60-70 рр. 20-го сторіччя	Науково-технічні розрахунки, економічні розрахунки
Трете	Інтегральні схеми	70-80 рр. 20-го сторіччя	Науково-технічні розрахунки, економічні розрахунки, системи керування
Четверте	Великі інтегральні системи, сверхвеликі інтегральні системи, ін.	80-90 рр. 20-го сторіччя	Всі сфери діяльності
П'яте	Ультра великі інтегральні системи, нанотехнології, фотоніка, нейронні мережі та ін.	Початок 21-го сторіччя та по теперішній час	Всі сфери діяльності

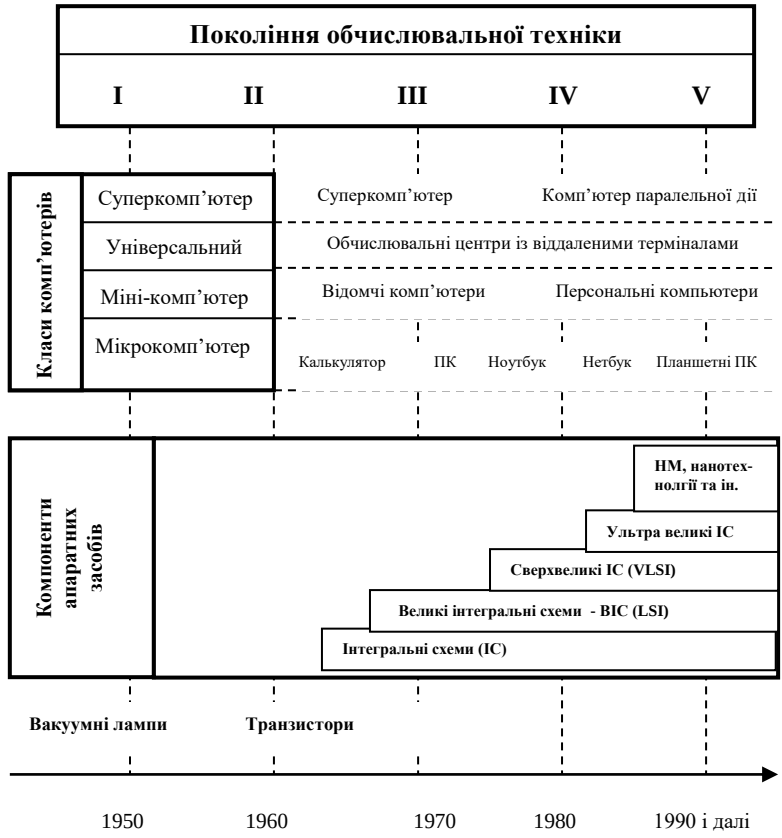


Рис. 1.4. Класи сьогоднішніх інформаційних систем і поколінь комп'ютерів

Так наприклад, за останні роки намітилася тенденція активного впровадження в архітектуру інформаційних систем новітніх технологій, зокрема, таких як – нейронні мережі [16], нанотехнології, фотоніка [17, 37, 46 та ін.].

У табл. 1.4 наведені основні етапи розвитку засобів обчислювальної техніки та інформаційних технологій в Україні. Слід зазначити, що наведені дані не є повними. Більш детальну інформацію стосовно історії розвитку інформаційних систем можливо знайти у наступних джерелах [13, 21, 30, 37].

Таблиця 1.4

Основні етапи розвитку засобів обчислювальної техніки та інформаційних технологій в Україні

Рік	Подія	Автори (Організація)
1914	Висловлено ідею механізації формалізуємих логічних дій. Побудована "Машина механічного мислення".	О.М. Щукарєв (Харківський технологічний інститут).
1951	Прийняла в експлуатацію Державною комісією перша в СРСР та континентальній Європі цифрова електронна обчислювальна машина - Мала електронна лічильна машина "МЭСМ").	С.О. Лебедєв (АН УРСР).
1954 - 1957	Розроблено принципи побудови, структура й архітектура і створена перша в Україні асинхронна ЕОМ "Київ" з використанням "адресної мови".	Б.В. Гнеденко, В.М. Глушков, В.С. Королюк, К.Л. Ющенко, Л.Н. Дашевський, К.О. Шкабара, С.Б. Погребинський (АН УРСР).
1961	Розроблено теорію цифрових автоматів, яка стала теоретичною основою при проектування ЕОМ. За монографію "Синтез цифрових автоматів" і ряд інших робіт у цій галузі В.М. Глушков одержав Ленінську премію (1964).	В.М. Глушков (АН УРСР).
1963	Отримано авторське свідоцтво на ступінчасте мікропрограмне керування Реалізовано в машинах сімейства "МИР".	В.М. Глушков (АН УРСР).
1964 - 1967	Розроблено принципи побудови, структура та архітектура і створено перший в Україні інформаційно-керуючий комплекс "Днепр-2" для автоматизованих систем керування.	В.М. Глушков, А.О. Стогній, А.Г. Кухарчук та ін. (АН УРСР, НВО "Електронмаш").

Продовження табл. 1.4

1965	Розроблено принципи побудови, створений та випробуваний на промисловій установці перший в СРСР цифровий регулятор "Автооператор".	Е.Т. Бєліков (Сєвєродонецьке НВО «Імпульс»).
1972	Підготовлено та опубліковано першу в світі "Енциклопедію кібернетики" українською та російською мовами.	В.М. Глушков, А.І. Кухтенко, Б.М. Малиновський та ін. (АН УРСР).
1974	Вперше в Україні, колишньому СРСР та Європі розпочате масове виробництво великих інтегральних схем.	С.О. Моральов, К.М. Кролевець, В.П. Бєлявський (НВО «Кристал»).
1976	Перший в Україні сигнальний процесор для обробки цифрових сигналів.	М.В. Семотюк (АН УРСР).
1977	Міні ЕОМ "Процесор".	О.В. Палагін, АН УРСР. В.П. Денисенко, С.Д. Погорелий. ВО (ім. С.П. Корольова).
1979	Мікрокомп'ютер "УВС-01".	А.В. Кобилинський, О.В. Палагін, С.Д. Погорелий ВТО «Кристал» (АН УРСР, ВО ім. С.П.Корольова).
1965-1980	Розроблено і випускалися великими серіями цифрові засоби промислової системотехніки, які забезпечили створення широкого спектру керуючих систем у промисловості та енергетиці для всього колишнього СРСР. Створено і випускалися надпотужні багатопроцесорні комплекси для систем геофізичної розвідки корисних копалин і ряду унікальних систем військового призначення.	А.О. Новохатній, В.В. Рєзанов та ін. (Сєвєродонецьке НВО «Імпульс»).

Продовження табл. 1.4

1975 - 1980	Створено та випускався великою серією перший в Україні комплекс мікропроцесорних засобів "Нейрон" та засобів налагодження до них "СО-01" - "СО-04".	Б.М.Малиновський, О.В. Палагін, В.І. Сигалов, С.Д. Погорелий, А.І. Слободянюк та ін. (АН УРСР).
1981 - 1986	Сумісний ряд персональних мікропроцесорних комп'ютерів: "ЕС 1840", "ЕС 1841", "ЕС 1842". Для використання в автономному режимі в локальних і глобальних мережах при розв'язанні широкого кола науково-технічних, економічних, спеціальних задач, задач керування та діловодства.	Ю.С. Яковлев, Ф.А. Цвентух, Н.В. Несте-ренко, Б.В.Нові-ков (АН УРСР, НДІКОМ і МВООТ м. Мінськ).
1981-1983	Розробка, освоєння серійного випуску обчислювальних комплексів "СМ3" та "СМ4" системи малих ЕОМ.	А.Ф. Незабитовський, В.А. Афанасьєв, С.С. Забара (НВО «Електронмаш»).
1988-1989	Перший у СРСР нейрокомп'ютер на основі ідеології ансамблевих стохастичних нейромереж.	Е.М. Куссуль (АН УРСР).
2005	Кластерна супер-ЕОМ. Сумарна продуктивність понад півтрильйона опер./сек.	І.В. Сергієнко, В.М. Коваль (Інститут кібернетики імені В.М. Глушкова Національної академії наук України).

В даний час мережі інформаційних систем є великими розподіленими системами програм і пристроїв, що взаємодіють між собою з метою обміну, зберігання та обробки інформації. Ускладнення структури мереж, підвищення навантаження на них, регулярна поява нових методів порушення роботи, способи обмеження доступу до даних законних власників і цілісності інформації, перетворення сфери розробки небажаного програмного забезпечення з любительського заняття в прибутковий бізнес визначають необхідність серйозного відношення до питань інформаційної безпеки.

З ростом навантаження на мережі інформаційних систем і відповідні системи захисту інформації з обмеженим доступом у останні десятиріччя з'явились і новітні напрямки удосконалення апаратної частини інформаційних систем із використовуванням систем штучного інтелекту, нанотехнологій та ін. (т.з. п'яте покоління інформаційних систем). Дослідження відомих засобів та методів захисту інформації з обмеженим доступом, показали необхідність подолання ряду складних і актуальних науково-технічних проблем, зв'язаних з розробкою більш «інтелектуальних» цифрових компонентів для систем захисту інформації з обмеженим доступом, складністю створення або вибору необхідних детекторів для виявлення несанкціонованого втручання у роботу мереж, високим рівнем позитивних помилок і низьким рівнем виявлення втручання.

1.9.2. Розвиток прикладної теорії цифрових автоматів

Серед багатьох робот які присвячені основам прикладної теорії цифрових автоматів ми зупинимося на наш погляд на найбільш важливих.

У 1964 році вийшла монографія В.М. Глушкова «Синтез цифрових автоматів» [6], яка і по сей день є настільною книгою фахівців у галузі проектування цифрових систем.

Серед видань закордонних авторів за період 50-90 рр. 20-го сторіччя, можливо відмітити наступні роботи:

- А. Тюрінг. «Чи може машина мислити?» - [32];
- Брауер В. «Вступ до теорії кінцевих автоматів» - [1];
- Флойд Т. «Цифрові основи» - [38];
- Точі Р., Відмер Н. «Цифрові системи. Теорія та практика» -[30];
- та інші [34,41].

У 70-90 рр. 20-го сторіччя було видано багато навчальний посібників та навчальних посібників з курсу «Комп'ютерна логіка» [3, 14, 20-25], з яких на наш погляд найбільш вагомою є роботи [22, 25,31].

За останні роки в Україні та Росії з'явилось чимало нових видань з цієї проблематики, але на наш погляд найбільш цікавими є роботи:

- навчальний посібник «Комп'ютерна логіка», автори - Жабін В.І., Жуков І.А., Клименко І.А., Ткаченко В.В. [8];
- навчальний посібник «Комп'ютерна логіка», автори Самофалов К.Г., Романкевіч О.М. та ін [27];
- навчальний посібник «Теорія автоматів», автори Горбатов В.А., Горбатова М.В., Горбатов О.В. [7];
- та ін.

Значний внесок в розвиток прикладної теорії цифрових автоматів та її застосування в галузі обчислювальних машин зробили таки фахівці як

Акушський І.Й., Брюхович О.С., Бузовський О.В., Глушков В.М., Жабін В.І., Карпов Ю.Г., Крутовських С.А., Лазарев В.Г., Луцький Г.М., Майоров С.А., Мельник А.О., Николайчук Я.М., Палагін О.В., Романов С.І., Савельев А.Я., Самофалов К.Г. , Тарасенко В.П., Трахтенброт Б.О., Петришин Л.Б., Поспелов Д.А., Черняков В.П., Широчин В.П., та інші.

Наведені вище навчальний посібники та навчальні посібники, безумовно є такими, що заслуговують уваги з боку студентів які вивчають дисципліну «Комп'ютерна логіка», проте на відміну від традиційного викладання цього курсу, ми приділили значну увагу цифровим компонентам, в першу чергу, систем захисту інформації з обмеженим доступом та деяким аспектам проектування, програмного моделювання і використовування цих компонентів для розв'язання важливої проблеми підвищення рівня інформаційної безпеки.

1.9.3. Розвиток технічних засобів та служб захисту інформації з обмеженим доступом в Україні

Важливість наукових та прикладних задач, зокрема у галузі інформаційної безпеки, які вирішуються за допомогою цифрової обробки даних, зумовлюють незмінну увагу до досліджень та розробок альтернативних цифрових пристроїв для систем захисту інформації з обмеженим доступом.

Питання ТЗІ розбиваються на два великих класи задач [4, 11, 19]:

1) захист інформації від несанкціонованого доступу;

2) захист інформації від витоку технічними каналами.

Під несанкціонованим доступом звичайно розуміється доступ до інформації, що порушує встановлену в інформаційній системі політику розмежування доступу. Під технічними каналами розглядаються канали побічних електромагнітних випромінювань і наводок, акустичні канали, оптичні канали та інші [4, 19].

На сьогодні основна увага розробників комплексів систем захисту інформації приділяється програмним засобам. Але, слід відзначити, що ці засоби мають певні недоліки, зокрема, головний з них – недостатня стійкість до зламу, особливо в умовах зростаючої інтенсивності обробки інформації та конфліктності потоків даних. Цих недоліків позбавлені комплекси захисту інформації які базуються на апаратних засобах, зокрема цифрових системах, які забезпечують нарощення функціональності цих систем в умовах зростання навантаження на комп'ютерні системи та мережі.

Взагалі апаратні засоби (АЗ) можна поділити на наступні групи:

– апаратні засоби виконання криптографічних алгоритмів;

– апаратні засоби виконання криптографічних функцій;

– апаратні засоби виконання криптографічних протоколів.

В Україні вже багато років запроваджені відповідні стандарти на алгоритми захисту інформації. До переліку алгоритмів входять:

– алгоритми симетричного шифрування (ГОСТ 28147-89, DES, IDEA, AES,RC, CAST та ін.) [4,19];

– алгоритми обчислення хеш-функцій (ГОСТ Р34.11-94, SHA-1, MD4, MD5) [4,19];

– криптографічний захист інформації. Цифровий підпис, що ґрунтується на еліптичних кривих (ДСТУ 4145-2002) [47];

– процедури цифрового підпису основані на асиметричних алгоритмах шифрування (алгоритми наступних модульних операцій – додавання, віднімання, обчислення обернених елементів над числами великої розрядності та ін.) [4, 19, 39, 42].

В сучасних комплексах систем захисту інформації з обмеженим доступом перелічені алгоритми реалізуються на базі універсальних програмованих процесорів. Суттєвий приріст продуктивності обробки даних досягається при використанні апаратно-орієнтованих процесорів для виконання криптографічних алгоритмів. У цих процесорах операційний пристрій орієнтований на виконання повного або частини потокового криптографічного алгоритму, що досягається завдяки використовуванню конвеєрної організації обчислень.

Завдяки досягненням в галузях проектування та виробництва цифрових систем на сьогодні сформувався новий підхід до проектування комп'ютерних засобів, який передбачає розробку та продаж на ринку ядер пристроїв інформаційних систем та відповідної технічної документації. До складу цієї конструкторської документації входить програмна модель пристроїв інформаційних систем, написана мовою опису апаратних засобів, програмні моделі пристрою орієнтовані на його реалізацію в конкретному кристалі, засоби перевірки функцій та параметрів пристрою, детальний опис його інтерфейсу та режимів функціонування.

Ядра цифрових пристроїв комплексів систем захисту інформації з обмеженим доступом включають наступні моделі, написані на одній із мов опису апаратних засобів, як правило VHDL або Verilog [28]:

– моделі тракту обробки даних та системи керування із найпростішими інтерфейсами даних та службової інформації – такий підхід дозволяє розробникам системи захисту інформації з обмеженим доступом використати готову або спроектувати свою інтерфейсну логіку і максимально наблизити технічні параметри отриманої системи до певних потреб;

– закінчені моделі цифрових систем захисту інформації з обмеженим доступом із фіксованою інтерфейсною логікою, що дозволяє

будувати комплекс систем захисту інформації з обмеженим доступом, без додаткових затрат.

У табл. 1.5 наведені основні етапи розвитку технічних служб системи захисту інформації з обмеженим доступом в Україні. Більш детальна інформація з цього питання наведена у наступних джерелах [4, 19].

Таблиця 1.5

Основні етапи розвитку технічних служб системи захисту інформації з обмеженим доступом в Україні

Роки	Події
1921	В системі ВЧК було створено Спеціальний відділ – перша повноцінна криптографічна служба.
1930	Перша лінія ВЧ-зв'язку між Москвою та Харковом.
1937	В Україні було вперше встановлено апаратуру засекречування в Києві та Харкові.
1939-1945	Створення на території України вузлів урядового зв'язку Державної служби спеціального зв'язку та захисту інформації (ДССЗЗІ).
1950 - 1970	Роботи по створенню більш досконалих засобів шифрування, апаратури засекречування телефонних каналів зв'язку гарантованої стійкості, яка могла б працювати на проводних, радіорелейних, тропосферних та супутникових каналах зв'язку як на стаціонарах, так і на рухомих об'єктах. Із середини 70-х років така апаратура стала вироблятися промисловістю і постачатися до територіальних підрозділів та військ урядового зв'язку на території України.
1980 - 1990	У 80-х роках до штатів майже всіх обласних підрозділів ДССЗЗІ було введено рухомі вузли зв'язку та окремі станції комплексу «Дон», оснащені спецапаратурою, що значно збільшило можливості щодо забезпечення урядового зв'язку з місць надзвичайних подій та з непідготовлених у відношенні зв'язку районів.
1992	Прийнято Закон України "Про Службу безпеки України". Цей закон законодавчо визначив порядок забезпечення засекреченим і шифрованим зв'язком державних органів України та відповідних посадових осіб.
1992-1995	Закладені основи сучасної науково-технічної бази, які дозволяли розвивати технічні системи спеціального зв'язку й формувати систему захисту інформації з обмеженим доступом України.

Продовження табл. 1.5

1998	На базі Головного управління урядового зв'язку СБ України із залученням фахівців Головного управління технічного захисту інформації Держкомсекретів України було створено Департамент спеціальних телекомунікаційних систем та захисту інформації СБ України (ДСТСЗІ СБ України), який став головною структурою в державі з питань криптографічного та технічного захисту інформації.
2005	Указом Президента України від 07.11.2005 № 1556/2005 "Про додержання прав людини під час проведення оперативно-технічних заходів" визначено необхідність створення в державі Служби спеціального зв'язку та захисту інформації України, як центрального органу виконавчої влади із спеціальним статусом, визначивши її основними завданнями реалізацію державної політики у сфері захисту державних інформаційних ресурсів у мережах передачі даних, забезпечення функціонування Державної системи урядового зв'язку, Національної системи конфіденційного зв'язку, криптографічного та технічного захисту інформації. Необхідність існування такого державного органу також визначена в Законах України "Про захист інформації в інформаційно-телекомунікаційних системах", "Про електронний цифровий підпис", "Про електронні документи та електронний документообіг" тощо.

Разом з появою нових задач удосконалюються і вимоги до технічних засобів, зокрема цифрових, захисту інформації з обмеженим доступом в інформаційних системах, які на сучасному етапі передбачають:

- захист інформації з обмеженим доступом при передачі її по каналах зв'язку, зберіганні і обробці;
- забезпечення цілісності і автентичності інформації;
- підтвердження автентичності відправника або одержувача інформації;
- контроль доступу до ресурсів мережі, устаткування і даних абонентів;
- можливість доказу неправомочності дій користувачів і обслуговуючого персоналу в мережі;
- захист від відмов з боку відправника або одержувача інформації;
- забезпечення взаємодії між різними локальними інформаційними системами
- ін.

Цілком природно, що досягнення цих вимог неможливе без вирішення проблем забезпечення безпеки функціонування національних інформаційно-телекомунікаційних систем, створення комплексної системи захисту інформації з обмеженим доступом із застосуванням сучасних цифрових систем в країні, розробки безпечних інформаційних технологій, методів і засобів захисту інформації з обмеженим доступом.

1.10. Загальні поняття про цифровий автомат й алгоритм

Необхідність формального опису інформаційної системи та її окремих частин у процесі проектування вимагає застосування спеціального математичного апарату, який необхідний для будь-яких розробок різноманітних методів обробки інформації, синтезу й аналізу будь-яких інформаційних процесів. Для цього введено поняття абстрактного цифрового автомата.

Пристрої, призначені для обробки й перетворення цифрової інформації, називаються *цифровими автоматами*. Найбільш поширеним видом цифрових автоматів є електронні цифрові обчислювальні машини з програмним управлінням, тобто інформаційні системи [1,3,8,14,18,25,27].

Введемо інше визначення поняття цифрового автомата [14,18,25,27].

Цифровий автомат – це пристрій, який характеризується набором деяких внутрішніх станів – $A=\{a_1(t), a_2(t), \ldots, a_n(t)\}$, у які він потрапляє під впливом вхідних сигналів і команд програми розв'язання задачі.

Нехай є автомат з одним входом і одним виходом (див. рис. 1.5).

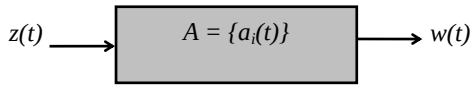


Рис. 1.5. Схема цифрового автомата

Тоді математичною моделлю цифрового автомата є деякий абстрактний автомат, заданий таким чином: у початковий момент часу $t = t_0$ автомат перебуває у стані $a(t_0) = a_1$ і залишається в ньому до моменту $t = t_1$, коли з'являється вхідний сигнал $z(t_1)$. Під впливом сигналу $z(t_1)$ автомат переходить із стану a_1 в стан $a(t_1) = a_2$. При цьому виникає вихідний сигнал $w(t_1) = w_1$, що визначається як функція $w(t_1) = \lambda[a(t_1), z(t_1)]$. Таким чином, можна прийняти, що під час подачі довільного сигналу $z(t) = z_f$ автомат

переходить із стану $a(t)$ у стан $a(t+1)$, яким є функція $a(t+1) = \delta[a(t), z(t)]$, і в результаті виробляє вихідний сигнал $w(t)$.

Вихідні сигнали можуть вироблятися під час кожного переходу автомата із стану $a(t)$ в стан $a(t+1)$ або лише за певних поєднань вхідного сигналу й станів автомата.

Поняття *стану автомата* використовують для опису систем, виходи яких залежать не лише від вхідних сигналів у певний момент часу, але й від певної передісторії, тобто сигналів, які надійшли на входи системи раніше. Стан автомата відповідає певній пам'яті про минуле, дозволяючи усунути час як явну перемінну й виразити вихідні сигнали як функцію станів і вхідних сигналів.

Таким чином, абстрактний автомат можна описати за допомогою таких параметрів [1, 3, 8, 14, 18, 25, 27 та ін.]:

a_1 – початковий стан автомата;

$A = \{a_1, a_2, ..., a_m\}$ – множина внутрішніх станів;

$Z = \{z_1, z_2, ..., z_k\}$ – множина вхідних сигналів;

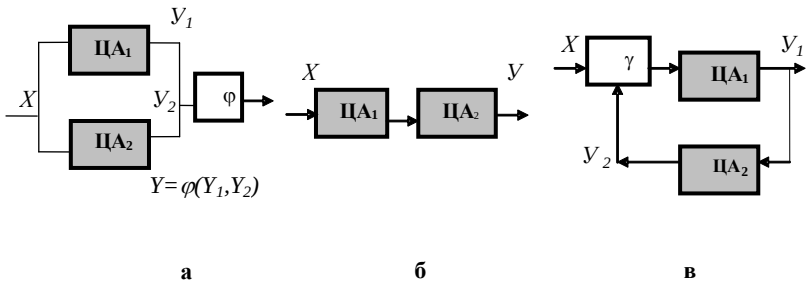
$W = \{w_1, w_2, ..., w_h\}$ – множина вихідних сигналів;

$\Delta = \{\delta_1, \delta_2, ..., \delta_n\}$ – сукупність функцій переходу з одного стану в інший;

$\lambda = \{\lambda_1, \lambda_2, ..., \lambda_p\}$ – сукупність функцій виходів.

Отже, абстрактний автомат реалізує деяке відображення множини слів вхідного алфавіту *(Z)* у множину слів вихідного алфавіту *(W)*.

Цифрові автомати можуть бути об'єднані між собою для формування більш складного пристрою обробки цифрової інформації. Існує кілька способів з'єднання автоматів, наприклад, паралельне з'єднання автоматів ЦА$_1$ і ЦА$_2$ (рис. 1.6 а), послідовне з'єднання (рис. 1.6. б) і з'єднання із зворотним зв'язком (рис. 1.6. в).



а **б** **в**

Рис. 1.6. Схеми з'єднання цифрових автоматів

У першому випадку пристрій φ об'єднує виходи двох автоматів і тим самим утворюється новий автомат, у якого $Y = \varphi(Y_1, Y_2)$.

У другому випадку вихідний алфавіт першого автомата є вхідним для другого.

У третьому випадку наявний деякий перетворювач γ, який є автоматом без пам'яті, що реалізує відображення $Y_1=\gamma(X, Y_2)$. Очевидно, що в усіх цих випадках у підсумку формується деякий цифровий автомат з новими характеристиками.

Сукупність правил переходу автомата з одного стану в інший залежно від вхідної інформації й внутрішніх станів автомата називається *алгоритмом перетворення (переробки) інформації.*

Алгоритм – скінченна сукупність точно сформульованих правил вирішення задачі.

Можна навести ще одне розширене визначення алгоритму. *Алгоритм* – це суворо формальний опис скінченної послідовності деяких „елементарних" дій або процедур, які треба виконати над вихідними даними й над проміжними результатами, що виникли під час виконання цих операцій, для того щоб прийти до інформації, яка є результатом обробки вихідних даних [1,6,8,14,15,24,27,34,35]. Опис принципів розробки алгоритмів та їх блок-схем наведено в 8-му розділі.

Перераховані в цьому розділі поняття стосуються абстрактної теорії цифрових автоматів, у якій розглядають автомати, що мають один вхід і один вихід. Тому застосувати все це до інформаційних систем можна лише в найзагальнішому вигляді, обмежуючи коло аналізованих пристроїв тими, які входять до складу процесора, або ж у випадку, коли розглядають узагалі деякий вузол інформаційної системи, призначений для елементарної обробки цифрової інформації. Наприклад, залежно від команд, які подає пристрій управління процесора, арифметично-логічний пристрій здійснює відповідні дії (зміна внутрішніх станів) з виданням необхідних результатів. Однак зміна внутрішніх станів усієї інформаційної системи у цілому має настільки складний характер, що цей процес неможливо відобразити в аналітичному вигляді. Тому поняття цифрового автомата доцільно використовувати щодо алгоритмів, які реалізовують деяку програму або послідовність операцій. При цьому кожна операція подається як елементарна дія, що здійснюється в процесі перетворення інформації.

Запитання для самоперевірки

1. Дайте визначення поняття „інформація".
2. Що таке структурна міра інформації?

3. Яким чином проблема захисту інформації пов’язана з прикладною теорією цифрових автоматів?
4. Назвіть основни етапи розвитку обчислювальних машин та технічних засобів захисту інформації з обмеженим доступом.
5. Опишіть абстрактний автомат.
6. Який вигляд має програма в машині Тьюрінга?
7. Чим відрізняються машини Поста й Тьюрінга.
8. Назвіть типи сигналів, використовувані в інформаційній системі.
9. Що таке дискретизація сигналу?
10. Що таке квантування сигналу?
11. Що таке код сигналу?
12. Дайте визначення цифрового автомата.
13. Що таке абстрактний автомат?
14. Дайте визначення поняття „стан автомата”.
15. Що таке алгоритм перетворення інформації?
16. Якими параметрами можна описати абстрактний цифровий автомат?

Література для самостійної підготовки за темою:

1, 3, 8, 11, 14, 17, 18, 25, 25, 26, 27, 35.

Розділ 2. ПРЕДСТАВЛЕННЯ ЧИСЛОВОЇ ІНФОРМАЦІЇ В ЦИФРОВОМУ АВТОМАТІ

2.1. Системи числення й поняття коду

Перетворення інформації у цифрову дає можливість цифровим автоматам оперувати з числами, представленими в певній системі числення.

Система числення – це сукупність прийомів і правил для запису чисел цифровими знаками. Запис числа в певній системі числення часто називають *кодом числа*.

Елементи (символи) алфавіту, які використовують для запису чисел у певній системі числення, прийнято називати *цифрами*. З кожною цифрою певного числа однозначно зіставляється її кількісний (числовий) еквівалент.

Розрізняють позиційні й непозиційні системи числення.

Непозиційна система числення – це система, для якої значення символу, тобто цифри, не залежить від його положення в числі. До таких систем належить, зокрема, римська система. Це найбільш відома система, після арабської. З нею ми доволі часто стикаємося в повсякденному житті. Це номери розділів у книгах, указівки на століття, числа на циферблатах годинників та ін. Виникла ця нумерація в давньому Римі. Тут, наприклад, символ *V* завжди означає п'ять, незалежно від місця його появи в записі числа. Цю нумерацію використовували для адитивної алфавітної системи числення. Цифри числа записували починаючи з більших значень і закінчуючи меншими, зліва направо. Якщо цифру з меншим значенням було записано перед цифрою з більшим значенням, то відбувалося її віднімання, наприклад *IV*. Або *CCXXXVII = 237, XXXIX = 39*.

Є й інші сучасні непозиційні системи. Недоліком непозиційних систем числення є необмежена кількість різноманітних цифр, необхідних для представлення будь-якого числа.

Позиційна система числення – це система, у якій значення кожної цифри залежить від її числового еквівалента й від її місця (позиції) в числі, тобто той самий символ (цифра) може набувати різних значень.

Найвідомішою позиційною системою числення є десяткова система числення, наприклад, у десятковому числі 254 перша цифра справа означає 4 одиниці, сусідня з нею – 5 десятків, а ліва – 2 сотні.

У зв'язку з тим, що в цифрових автоматах в основному використовують позиційні системи числення, то ми в подальшому розглядатимемо лише такі системи.

Двійкова система числення серед відомих займає особливе місце. Вона має властивості, які роблять її дуже вигідною для використання в цифрових автоматах (інформаційних системах). Офіційне народження двійкової арифметики пов'язане з іменем Г.В. Лейбніца, який опублікував у 1703 р. статтю, де розглянув правила виконання арифметичних дій над двійковими числами. Для двійкової системи необхідно лише два символи – 0, 1. У двійковій системі арифметичні операції надзвичайно прості.

Будь-якій позиційній системі числення властива специфічна основа.

Основа або базис q природної позиційної системи числення – це кількість знаків або символів, використовуваних для зображення числа в цій системі [21, 24, 35].

Тому можливою є нескінченна кількість позиційних систем, оскільки за основу можна прийняти будь-яке число, утворивши нову систему числення.

Коли ми записуємо якесь число в позиційній системі числення, то розміщуємо відповідні цифри числа на окремих потрібних позиціях, які прийнято називати розрядами числа в цій позиційній системі числення. Кількість розрядів у записі числа називається *розрядністю* числа й збігається з його *довжиною*.

У позиційній системі числення справедливою є рівність:

$$A_q = a_{n-1}q^{n-1} + a_{n-2}q^{n-2} + \ldots + a_k q^k + a_0 q^0 + a_{-1}q^{-1} + \ldots + a_{-m}q^{-m}, \quad (2.1)$$

де A_q – довільне число, записане в системі числення з основою q; a_i – коефіцієнти ряду, тобто цифри системи числення; n, m – кількість цілих і дробових розрядів відповідно.

Максимальне ціле число, яке може бути представлене в m розрядах, $A_{max}=q^n - 1$. Мінімальне значуще, що не дорівнює 0 число, яке можна записати в m розрядах дробової частини, $A_{min}=q^{-m}$.

Наприклад, число A=123,45 означає скорочений запис виразу

$$A=1\cdot10^2 + 2\cdot10^1 + 3\cdot10^0 + 4\cdot10^{-1} + 5\cdot10^{-2}.$$

Можна переконатися в тому, що в будь-якій системі числення, яка використовує арабські цифри, основа системи представляється як число 10.

Наприклад, відповідно до (2.1)

$1964{,}52_{10} = 1\cdot10^3 + 9\cdot10^2 + 6\cdot10^1 + 4\cdot10^0 + 5\cdot10^{-1} + 2\cdot10^{-2}$,

$124=537_8 = 1\cdot8^2 + 2\cdot8^1 + 4\cdot8^0 + 5\cdot8^{-1} + 3\cdot8^{-2} + 7\cdot8^{-3}$,

$$1001{,}1101_2 = 1\cdot2^3 + 0\cdot2^2 + 0\cdot2^1 + 1\cdot2^0 + 1\cdot2^{-1} + 1\cdot2^{-2} + 0\cdot2^{-3} + 1\cdot2^{-4}.$$

Індекс, що приписується до числа, указує систему числення, у якій представлено це число.

Основа системи числення показує, скільки різних значень у межах *i*-го розряду може приймати кожна цифра a_i числа *A*. Номери розрядів у позиційній системі числення рахуються в цілій частині вліво від коми, а в дробовій – вправо від коми, причому, нумерація розрядів починається з 0. Величина основи позиційної системи числення визначає її назву: для десяткової системи це буде 10, для вісімкової – 8, для двійкової – 2 і т. ін Як уже було відзначено, зазвичай замість назви системи числення використовують термін „**код числа**". Наприклад, під поняттям двійковий код мають на увазі число, представлене у двійковій системі числення, під поняттям десятковий код – у десятковій системі числення й т. ін.

Для запису числа в десятковій системі використовують 10 різних цифр від 0 до 9 A_{10} = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9).

У шістнадцятковій – 16 A_{16} = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F), де $A = 10, B = 11, C = 12, D = 13, E = 14, F = 15$.

У вісімковій – 8 A_8 = (0, 1, 2, 3, 4, 5, 6, 7).

У двійковій – 2 A_2 = (0,1).

У табл. 2.1 наведено відповідність десяткових, двійкових, вісімкових і шістнадцяткових чисел.

Таблиця 2.1

Відповідність десяткових, двійкових, вісімкових і шістнадцяткових чисел

Десяткові	Двійкові	Вісімкові	Шістнадцяткові	Десяткові	Двійкові	Вісімкові	Шістнадцяткові	Десяткові	Двійкові	Вісімкові	Шістнадцяткові
0	0	0	0	11	1011	13	B	22	10110	26	16
1	1	1	1	12	1100	14	C	23	10111	27	17
2	10	2	2	13	1101	15	D	24	11000	30	18
3	11	3	3	14	1110	16	E	25	11001	31	19
4	100	4	4	15	1111	17	F	26	11010	32	1A
5	101	5	5	16	10000	20	10	27	11011	33	1B
6	110	6	6	17	10001	21	11	28	11100	34	1C
7	111	7	7	18	10010	22	12	29	11101	35	1D
8	1000	10	8	19	10011	23	13	30	11110	36	1E
9	1001	11	9	20	10100	24	14	31	11111	37	1F
10	1010	12	A	21	10101	25	15	32	100000	40	20

Згідно з (табл. 2.1) кожен розряд числа у двійковій системі числення зліва від коми представляється двійкою у відповідному додатному ступені, а справа від коми – двійкою у від'ємному ступені, наприклад:

2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
16	8	4	2	1,0	0,5	0,25	0,125	0,0625

Приклади представлення чисел у різних системах числення:

$$10_{10} = 1010_2 = 12_8 = A_{16}$$

$$16_{10} = 10000_2 = 20_8 = 10_{16}$$

$$255_{10} = 11111111_2 = 377_8 = FF_{16}.$$

Для обробки інформації в цифрових автоматах зазвичай використовують двійкову систему числення. Це пояснюється, зокрема, тим, що для розміщення чисел (операндів) у цифрових автоматах використовують регістри й гнізда пам'яті, які складаються з тригерів або елементів з тригерною характеристикою, які, як відомо, мають два стійких стани. Одному з цих станів ставиться у відповідність 1, а другому – 0. Кожен з тригерів відведено для розміщення найменшої інформаційної одиниці у двійковій системі числення – двійкового розряду, який називається *бітом.* Вісім біт – це *байт.*

Кількість тригерів, тобто двійкових розрядів, у регістрі або гнізді пам'яті визначає довжину слова, характерну для певної інформаційної системи, а сукупність цих двійкових розрядів називається *розрядною сіткою*. Номер розряду такої сітки, відведеної для зображення цілого числа у двійковій системі числення, збігається з відповідним показником ступеня двійки.

Таким чином, *довжина числа* – це кількість позицій (або розрядів) у записі числа. Для різних систем числення характерна різна довжина розрядної сітки, необхідна для запису того самого числа. Наприклад, десяткове число 999 (999_{10}) відповідно в шістнадцятковій, вісімковій і двійковій системах числення матиме такий вигляд: $3E7_{16}$, 1747_8, 1111100111_2, тобто чим меншою є основа системи числення, тим більшою є довжина числа.

У будь-яких цифрових автоматах довжина розрядної сітки вибраної системи числення фіксована, що принципово обмежує точність і діапазон представлення чисел.

Діапазон представлення чисел у заданій системі числення – це інтервал числової осі між максимальним і мінімальним числами, значення яких залежить від довжини розрядної сітки.

Практичне переведення з двійкової системи в десяткову можна легко виконати, написавши над кожним розрядом відповідну йому вагу й склавши потім добуток значень відповідних цифр та їх ваги.

Наприклад, двійкове число 01000001_2 дорівнює 65. Дійсно, відповідно до табл. 2.2, $64 \cdot 1 + 1 \cdot 1 = 65$.

Таблиця 2.2

Приклад переведення числа з двійкової системи в десяткову

Вага	128	64	32	16	8	4	2	1
Цифра	0	1	0	0	0	0	0	1

Цифрові автомати містять велику кількість гнізд пам'яті й регістрів (від лат. *Regestum* – внесене, записане) для збереження двійкової інформації. Більшість цих гнізд мають довжину *n*, тобто їх використовують для збереження *n* біт двійкової інформації. Гнізда пам'яті й регістри складаються з елементів пам'яті. Кожен з таких електричних елементів може перебувати в одному з двох стійких станів. Один з таких фізичних станів створює високий рівень вихідної напруги елемента пам'яті, а інший – низький. На рис. 2.1 показано вихідний сигнал такого елемента пам'яті (наприклад, одного розряду регістру) при зміні його станів під впливом певного вихідного сигналу. Хоча перехід від 0 до 1 і від 1 до 0 відбувається не миттєво, однак у певні моменти часу сигнал досягає значень, які елементи ЕОМ сприймають як 0 або 1.

Окрім звичайної двійкової системи числення було запропоновано двійкові системи числення, у яких для зображення чисел використовують символи 1, -1 або 0, -1. Існують ще так звані надлишкові двійкові системи числення, наприклад, із символами 0, 1, -1. Але на практиці в переважній більшості випадків використовують звичайну двійкову систему числення.

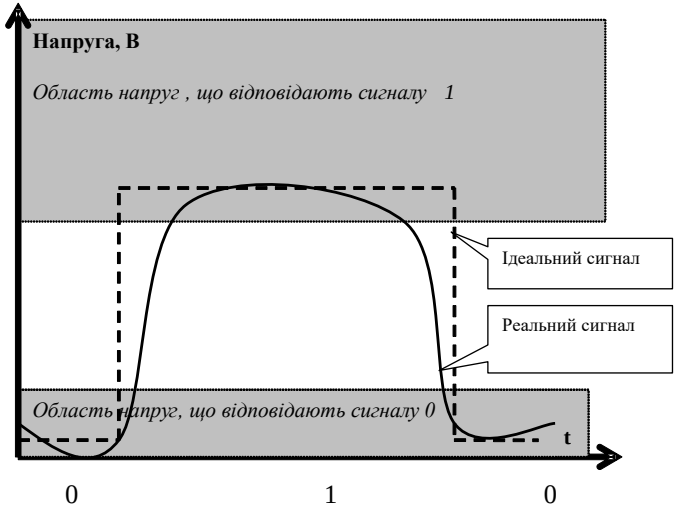


Рис. 2.1. Графічне зображення дискретного сигналу

При обміні даними між цифровим автоматом і зовнішніми пристроями виникає необхідність в обміні знаковими й буквеними символами. Цим символам у інформаційній системі також відповідає деякий код у двійковій системі числення. Для представлення цифр і букв у двійковій системі найпоширенішим зараз є код ASCII. Для представлення будь-якого символу в цьому коді відводиться 8 двійкових розрядів, тобто один байт. Приклади коду ASCII наведено в табл. 2.3.

Таблиця 2.3

Приклади коду ASCII

Символи	Десятковий код	Двійковий код	Вісімковий код	Шістнадцятковий код
0	48	0110000	060	30
1	49	0110001	61	31
2	50	0110010	62	32
A	65	1000001	101	41
B	66	1000010	102	42
F	70	1000110	106	46
:	58	0111010	72	3F
(	40	0101000	50	28

2.2. Вибір системи числення

Обираючи систему числення для цифрових автоматів, необхідно враховувати, що по-перше, основа системи числення визначає кількість стійких станів, які повинен мати функціональний елемент, вибраний для зображення розрядів числа; по-друге – довжина числа істотно залежить від основи системи числення; по-третє – система числення повинна забезпечити прості алгоритми виконання арифметичних і логічних операцій.

Широке застосування двійкової системи числення обумовлено такими причинами:

– розмаїттям і простотою технічної реалізації елементів з двома стійкими станами;

– надійним розрізненням двох станів, що зменшує ймовірність збою і внесення додаткових похибок у числові данні;

– простотою здійснення арифметичних операцій;

– економічністю обладнання.

Економічність обладнання при використанні двійкової системи числення проілюструємо на такому прикладі. Проектується пристрій, у якому для відображення кожної цифри числа потрібен один елемент. Тоді для представлення будь-якої цифри в системі числення з основою q потрібні q різних елементів. Для відображення в цій же системі числення будь-якого числа, що містить n_q розрядів, потрібно $L_q=q\cdot n_q$ елементів.

Наприклад, для представлення будь-якого числа від 000 до 999_{10} в десятковій системі потрібно $n_{10}=3$ розрядів кожен з яких містить по 10 цифр; при $L_{10}=30$ елементів. Для відображення цих же чисел у двійковій системі потрібно $n_2=10$ *розрядів*, кожен з яких містить по дві цифри, тобто $L_2=20$ елементів.

Таке порівняння десяткової й двійкової систем числення показує, що для реалізації десяткової системи необхідно в 1,5 рази більше елементів у порівнянні з двійковою.

Найбільш зручними є умови реалізації двійкових цифр, оскільки фізичних процесів, що мають два стійких стани, набагато більше, ніж процесів з кількістю чітко розрізнювальних станів більше двох. До того ж у процесах з двома стійкими станами відмінність між цими станами має якісний, а не кількісний характер, що забезпечує надійну реалізацію двійкових цифр.

Таким чином, простота арифметичних і логічних дій, мінімум використовуваного обладнання для представлення чисел і найбільш зручні умови реалізації лише двох стійких станів визначили застосування

двійкових систем числення практично в усіх наявних і проектованих цифрових обчислювальних машинах.

2.3. Формальні правила двійкової арифметики

У двійковій системі арифметичні операції особливо прості. У цій системі не існує „таблиці додавання”, яку треба було б запам’ятовувати, оскільки перенесення в старший розряд починається з 1+1=10. При додаванні великих чисел необхідно лише додавати за стовпцями або розрядами, як у десятковій системі, пам’ятаючи лише про те, що як тільки сума в стовпці досягне числа 2, двійка переноситься в наступний стовпець (уліво) у вигляді одиниці старшого розряду. Віднімання здійснюємо так само, як у десятковій системі, не замислюючись про те, що тепер у разі необхідності треба позичати із стовпця зліва 2, а не 10.

Однак за таку простоту виконання операцій при додаванні доводиться платити великою кількістю знаків при множенні навіть невеликих чисел.

У таблиці 2.4 наведено правила додавання, віднімання й множення двійкових чисел.

У загальному випадку процедури додавання й віднімання двох чисел у будь-якій позиційній системі числення розпочинаються з молодших розрядів (див. розділ 3).

Таблиця 2.4

Правила додавання, віднімання й множення двійкових чисел

Додавання	Віднімання	Множення
0 + 0 = 0	0 - 0 = 0	0 x 0 = 0
1 + 0 = 1	1 - 0 = 1	1 x 0 = 0
0 + 1 = 1	1 - 1 = 0	0 x 1 = 0
1 + 1 = 10	10 - 1 = 1	1 x 1 = 1

Код суми кожного *i*-го розряду c_i отримуємо в результаті складання a_i+b_i+1, де одиниця відповідає перенесенню з молодшого *(i-1)* розряду в *i*-й, якщо в молодшому розряді код суми вийшов більшим або рівним основі системи числення.

Код різниці кожного *i-го* розряду отримуємо в результаті віднімання a_i-b_i-1, де одиниця відповідає „позиції”, якщо вона була, в молодші розряди величини, яка дорівнює основі системи числення.

Отже, правила й методи додавання й віднімання в будь-якій позиційній системі числення в принципі залишаються такими ж, як у десятковій системі.

Розглянемо правила арифметики з числами, представленими в десятковому, двійковому, вісімковому й шістнадцятковому кодах на простому прикладі (див. табл. 2.5).

Таблиця 2.5

Правила додавання

Доданки	**Десятковий Код**	**Двійковий код**	**Вісімковий код**	**Шістнадцятковий код**
Перенос		1 111	11	1
1-е	166	10100110	246	A5
2-е	47	00101111	57	2F
Результат	213	11010101	325	D5

Віднімання також здійснюється порозрядно, починаючи з молодшого розряду. При відніманні в цьому розряді з нуля одиниці необхідно позичити одиницю з сусіднього старшого розряду, яка дорівнює двом одиницям цього розряду.

Додавання двійкових чисел у цифрових автоматах здійснюється за допомогою двійкових суматорів, а віднімання – двійкових віднімальників. Разом з тим, як буде показано далі, віднімання можна організувати також за допомогою процедури додавання, тобто за допомогою двійкових суматорів, якщо від'ємник представити в додатковому або оберненому коді й тим самим виключити необхідність у двійкових віднімальниках.

Множення двійкових чисел здійснюється шляхом утворення проміжних добутків і наступного їх сумування.

Арифметичні дії з двійковими числами детально буде розглянуто далі.

У виконанні будь-яких арифметичних дій важливе значення мають такі електронні пристрої, як двійковий півсуматор і двійковий суматор, які виконують побітове двійкове складання за раніше наведеними правилами. Для двійкового віднімання іноді використовують і двійковий віднімальник.

Наведемо умовне позначення двійкових півсуматора й суматора, див. рис. 2.2.

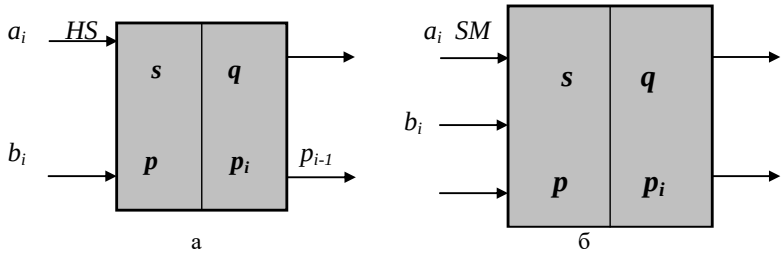


Рис. 2.2. Умовне позначення півсуматора (а)
і двійкового суматора (б)

Тут a_i і b_i – це *i-і* розряди чисел *A* і *B*, які додаються, а s_i – *і-ий* розряд суми цих чисел, p_i – перенесення з цього розряду в сусідній наступний старший, p_{i-1} – перенесення із сусіднього молодшого в цей розряд.

Якщо для представлення двійкових чисел *A, B, C* та їх знаків виділено *n* розрядну сітку, то очевидно, що для організації процедури додавання необхідно *n* двійкових суматорів, які з'єднуються між собою за певною схемою, що залежить від того, у якому коді представляються ці двійкові числа: прямому, оберненому чи додатковому.

В арифметичних пристроях цифрових автоматів, окрім двійкових суматорів, використовуються також регістри, лічильники, різноманітні тригери й електронні пристрої, що виконують різноманітні логічні процедури. Зазвичай використовувані регістри повинні дозволяти не лише паралельно записувати в них двійкові коди чисел, але й зсувати зображення цих чисел уліво й вправо на необхідну кількість двійкових розрядів.

Найпростішу блок-схему вузла, який виконує процедуру додавання *A+B=C*, можна представити таким чином:

P_r – регістри, у які записуються двійкові числа *A, B, C*; *CM* – суматор (група суматорів *n·CM*, де *n* – довжина розрядної сітки, відведеної для представлення чисел *A, B, C*).

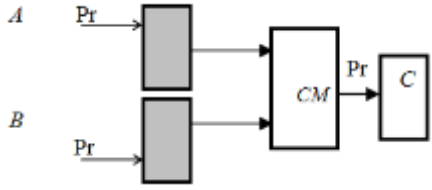


Рис. 2.3. Блок-схема вузла, який виконує процедуру додавання

Окрім *арифметичних* операцій, у цифрових автоматах реалізуються також *логічні* операції.

Крім цих операцій, у цифрових автоматах, інформаційних системах, виконується ще одна операція над двійковими числами – це *зсув* числа розрядною сіткою вліво або вправо. У випадку зсуву вліво фактично здійснюється множення двійкового числа на 2, а зсуву вправо – ділення на 2, де m – кількість розрядів, на яку зсувається двійкове число. Наприклад: $000011_2 = 3_{10}$ зсунемо вліво на два розряди, отримаємо $001100_2 = 12_{10}$, тобто $3x4(2^2) = 12_{10}$, а тепер $001000_2 = 8_{10}$ зсунемо на два розряди вправо, отримаємо $000010_2 = 2_{10}$, тобто $8:4(2^2) = 2_{10}$.

У інформаційних системах часто використовується *циклічний зсув*, під час виконання якого розрядна сітка, відведена для операнда, представляється замкнутою в кільце. Тоді при зсуві вліво вміст старшого розряду потрапляє в молодший розряд операнда, а при зсуві вправо – навпаки.

2.4. Переведення числової інформації з одної позиційної системи числення в іншу

Як уже було відзначено, будь-яка обробка інформації в інформаційній системі зазвичай здійснюється у двійковій системі числення. Разом з тим, під час обміну інформацією між інформаційною системою і користувачем для більшої наочності представлення відомостей використовуються десяткова, двійково-десяткова, вісімкова або шістнадцяткова системи. Кожен розряд числа у вісімковому й шістнадцятковому кодах еквівалентний трьом і чотирьом двійковим розрядам відповідно. Тому представлення чисел у цих системах числення виходить більш компактним і наочним.

Для переведення цілого числа з одної системи числення в іншу необхідно, діючи у вихідній системі, ділити число, що переводиться, на нову основу. Отриману частку необхідно знову ділити, і т. д. до отримання неподільної частки. Результат записують як останню частку й остачі в порядку, зворотному їх отриманню.

Наприклад, необхідно перевести число 75_8 з вісімкової системи числення в десяткову. Десяткову основу у вісімковій системі представлено як число 12_8. Дотримуючись вище наведених правил, у вісімковій системі ділимо 75_8 на 12_8:

$$\begin{array}{l|l} 75_8 & 12_8 \\ \underline{74_8} & \\ 1_8 & 6_8 \end{array}$$

Отримали неподільну частку - 6 і остачу - 1, записані у вісімковій системі числення. Вони й представлятимуть цифри шуканого десяткового числа 61_{10}.

Переведемо число 135_8 в десяткову систему числення.

$$\begin{array}{l|l} 135_8 & 12_8 \\ 12_8 & \\ \hline & 11_8 \\ 15_8 & \\ 12_8 & \\ \hline 3_8 & \end{array}$$

Отримали неподільну частку – 11_8 і остачу 3_8, які представляють значення цифр шуканого десяткового числа. Ураховуючи, що $3_8=3_{10}$, а $11_8=9_{10}$, отримаємо $135_8=93_{10}$.

Для полегшення переведення чисел з одної позиційної системи числення в іншу можна скористатися таблицею 2.6.

Таблиця 2.6

Перетворення десяткових чисел у шістнадцякові

Множник	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	1	2	3	4	5	6	7	8	9
10	A	14	1E	28	32	3C	46	50	5A
10^2	64	C8	12C	190	1F4	258	2BC	320	384
10^3	3E8	7D0	BB8	FA0	1388	1770	1B58	1F40	2328
10^4	2710	4E20	7530	9C40	C350	EA60	11170	13880	15F90
10^5	186A0	30D40	493E0	61A80	7A120	927C0	AAE60	C3500	DBBA0
Приклад: $1234=1000+200+30+4=(3E8)_{16}+(C8)_{16}+(1E)_{16}+(4)_{16}=(4D2)_{16}$									

Переведення двійкового числа в десятковий його еквівалент можна виконати за допомогою формули (2.1). Є й більш простий спосіб. Перетворення здійснюється шляхом сумування значень ступенів числа 2, які відповідають тим розрядам двійкового числа, що переводиться, у яких містяться одиниці. Наприклад, переведемо 10111010_2 в десяткове число.

Розряди числа, що переводиться	7	6	5	4	3	2	1	0
Число, що переводиться	1	0	1	1	1	0	1	0
Ступінь числа 2	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Результат	128	0	32	16	8	0	2	0

Отже, отримаємо 128+0+32+16+8+0+2+0=186. Таким чином, $10111010_2=186_{10}$.

Для дробових чисел (або дробових частин речовинних чисел) потрібна окрема процедура переведення. У випадку неправильного дробу процедура перетворення для цілої й дробової частин числа виконується окремо.

Для переведення правильного дробу (без цілої части) необхідно, діючи у вихідній системі числення, помножити число, що переводиться, на основу нової системи, у результату відділити цілу частину, а дробову частину, що залишилася, знову помножити на цю основу до отримання потрібного числа значущих цифр. Результат записується як *0,.*. і цілі части добутку в порядку їх отримання.

Наприклад, треба перетворити десятковий дріб $0,375_{10}$ у двійковий.

Відповідно з викладеним вище правилом, перетворення здійснюємо множенням дробу на основу системи числення, у якій дріб має бути представлений. Тобто 0,375· 2 = 0,75. Результат попередньої операції множення знову множимо на 2. Зазначимо, що якщо б результат попередньої операції множення був більше 1, то в цій операції множення брала б участь лише його дробова частина. У цьому випадку 0,75·2=1,5. Кроки описаної процедури повторюються до того часу, поки або результат множення не буде точно дорівнювати 1, або не буде досягнуто потрібної точності. У нашому прикладі після виконання чергового кроку результат дорівнює 0,5·2=1,0. Тому черговому значущому розряду регістра присвоюється 1, тобто остаточно отримано двійковий дріб $0,011_2$.

Ще приклад: треба перетворити дріб $0,375_6$ у трійковий. Спочатку $0,543_6$·3=$2,513_6$. Далі 0, 513_6 ·3 =$2,343_6$. і т. д. остаточно отримаємо $0,221_3$.

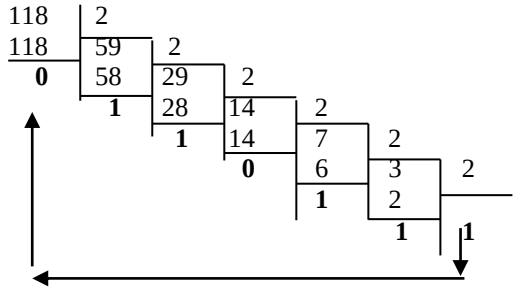
Слід зазначити, що не завжди шляхом повторення операцій множення можна досягти результату, який точно дорівнює 1. У такому випадку процес зупиняється після досягнення необхідної точності, а цілу частину результату останньої операції множення присвоюють молодшому значущому розряду.

Розглянемо ще один приклад: переведемо число $0{,}34375_{10}$ у двійкове. Спочатку 2·0,34375 = 0,6875. Далі 2·0,6875 = 1,375. Потім 2·0,375 = 0,75. Знову 2·0,75=1,5. Далі 2·0,5 = 1,0. 2·0=0. Отримаємо відповідь – $0{,}01011_{(2)}$.

Для переведення неправильного дробу (тобто дробу, який містить цілу частину) з одної системи числення в іншу необхідно здійснити окремо переведення його цілої й дробової частин, а результати записати послідовно, відділивши цілу частину від дробової за допомогою коми.

Наприклад, необхідно перевести дробове число $118,376_{10}$ з десяткової системи числення у двійкову. Результат переведення цілої частини – $118_{10}=1110110_2$.

Тобто



Здійснюємо переведення дробової частини з точністю до третього двійкового знака:

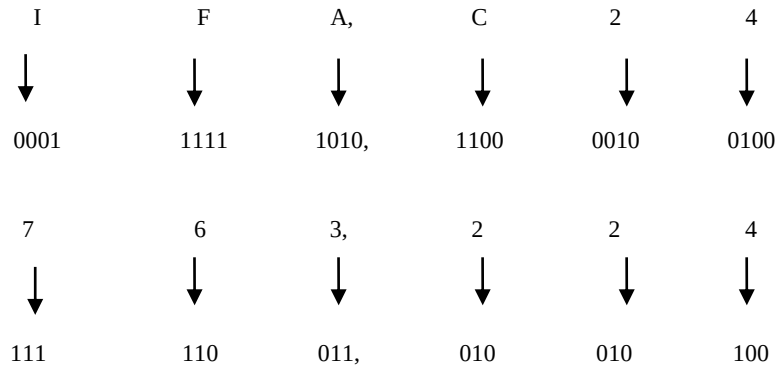
$0,\ 376_{10}$
x 2
$0,\ 752_{10}$
x 2
$1,\ 504_{10}$
x 2
$1,\ 008_{10}$

Отже, $0{,}376_{10} \approx 0{,}011_2$. Остаточний результат виглядатиме так $118,376_{10} \approx 1110110{,}011_2$.

Для переведення числа з шістнадцяткової й вісімкової систем числення у двійкову необхідно кожну цифру числа, що переводиться, представити відповідно чотири- або трирозрядним двійковим кодом, а отримані двійкові коди розкласти на місцях шістнадцяткових або

вісімкових цифр. Нулі в старших і молодших розрядах, які не змінюють значення числа, при цьому можна опустити.

Наприклад, шістнадцяткове число $IFA,C24_{16}$ і вісімкове число $763,224_8$ перетворюються таким чином у відповідні їм двійкові числа:



Опустивши незначущі нулі, отримаємо остаточно $IFA,C24_{16}=111111010,1100001_2$, $763,224_8=111110011,0100101_2$.

Для переведення чисел з двійкової системи числення в шістнадцяткову (вісімкову) слід двійкові цифри числа, яке переводиться, згрупувати по чотири (три) в обидва боки від коми. За необхідності неповні групи цифр доповнити 0. Кожну з отриманих груп двійкових цифр замінити відповідною їй цифрою в новій системі числення, а отримані шістнадцяткові (вісімкові) цифри розташувати на місцях замінюваних двійкових кодів.

Наприклад, перетворюємо число $111111010,1100001_2$ у шістнадцяткове.

0001	1111	1010,	1100	0010	0100
I	F	A	C	2	4

Таким чином, при переведенні числової інформації з одної позиційної системи числення в іншу всі дії повинні виконуватися за правилами арифметики вихідної системи числення.

На перший погляд, переведення чисел з одної системи числення в іншу – дуже складний процес, але вручну переведення робити необов’язково. Можна скористатися інженерним калькулятором (див. рис. 2.4).

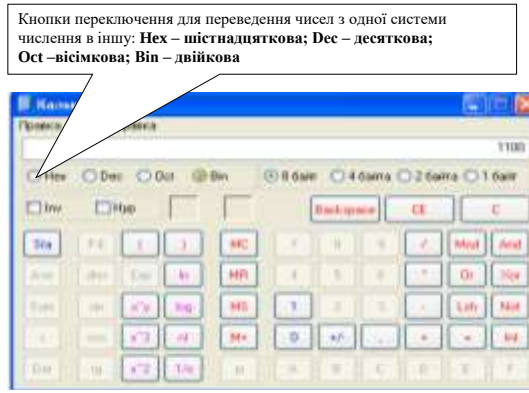


Рис. 2.4. Зовнішній вигляд калькулятора

Розглядаючи переведення з десяткової системи числення у двійкову й шістнадцяткову, можна знайти багато спільного. В обох випадках ми шукаємо максимальний ступінь, потім в обох випадках порівнюємо остачу з числом, піднесеним у ступінь розряду. Єдина різниця полягає в тому, що при переведенні у двійкову систему основою ступеня слугує двійка, а при переведенні в шістнадцяткову – число 16.

2.5. Форма представлення чисел з фіксованою комою

Формою представлення чисел у цифрових автоматах називається сукупність правил, що дозволяють установити взаємну відповідність між записом числа та його кількісним еквівалентом.

Машинне (автоматне) зображення числа – це представлення числа в розрядній сітці цифрового автомата. Умовну позначку машинного зображення числа, наприклад, *A* будемо представляти як *[A]*.

Через обмежену довжину машинних слів, множина чисел, які можна представити в машині, скінченна. Порівняння різноманітних форм представлення чисел у цифрових автоматах зазвичай здійснюється на основі оцінки діапазону й точності представлення числа.

На практиці найбільш поширеною є форма представлення чисел у вигляді послідовності цифр, розділеної комою на цілу й дробову частини. Числа, представлені в такій формі, називаються числами з природною

комою або числами в природній формі. У природній формі число записується в природному натуральному вигляді, наприклад 2360 – ціле число, 0,0065 – правильний дріб, 64,89760 – неправильний дріб.

У представленні чисел у такій формі обов'язково потрібна для кожного числа вказівка про положення його коми в розрядній сітці, виділеній для представлення числа в машині, що вимагає додаткових апаратних затрат достатньо великого обсягу. Тому в цифрових автоматах набули поширення дві інші форми представлення – *з фіксованою й плаваючою комою*.

Необхідність в указівці положення коми відпадає, якщо місце коми в розрядній сітці машини заздалегідь фіксовано. Така форма представлення чисел називається представленням з фіксованою комою (точкою), див. рис. 2.5.

У формі представлення з фіксованою комою всі числа зображуються у вигляді послідовності цифр з постійним для всіх чисел положенням коми, яка відділяє цілу частину від дробової. Наприклад, у десятковій системі числення є п'ять розрядів у цілій частині числа й п'ять розрядів – у дробовій. Числа, записані в таку розрядну сітку, матимуть вигляд: +00567,22100; +00000,00044; -23501,20450.

Припустимо, що в розрядній сітці необхідно розмістити двійкове число, яке містить цілу й дробову частини. Якщо для розміщення цілої частини відводиться *r* гнізд *n* – розрядної сітки, то (без урахування знака) для розміщення дробової частини залишиться *n-r* вільних гнізд. Якщо кількість розрядів у дробовій частині розміщуваного числа перевищує *n-r,* то деякі молодші розряди виявляться за межами розрядної сітки. Отже, будь-яке двійкове число, менше ніж 0. 0…1 (*n-r нулів)* сприймається як нульове й називається *машинним нулем.*

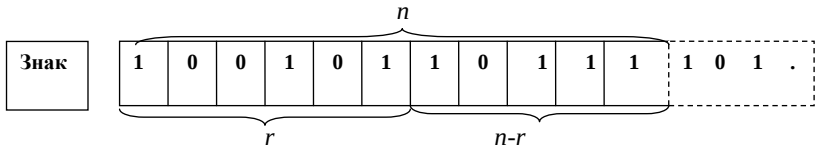


Рис. 2.5. Представлення числа з фіксованою комою

Оскільки числа бувають додатні й від'ємні, то формат (розрядна сітка) машинного зображення розбивається на *знакову частину й поле числа*. У полі числа розміщується саме зображення числа, яке ми умовно називатимемо *мантисою числа*. Для кодування знака числа використовується найстарший розряд розрядної сітки, відведеної для

зображення двійкового числа, а інші розряди відводяться під мантису числа. Положення коми в розрядній сітці суворо фіксується, зазвичай або правіше наймолодшого розряду мантиси, або лівіше найстаршого. У першому випадку число представляється як ціле, у другому – як правильний дріб. У цифрових автоматах у форматі з фіксованою точкою представляються цілі числа.

У знакову частину записується інформація про знак числа. Прийнято, що знак додатного числа „+" зображується символом 0, а знак від'ємного числа „-" – символом 1. Наприклад, у двійковому коді, використовуючи 6-розрядну сітку, число 7 у формі з фіксованою комою можна представити у вигляді – 0.00111_2. Цифра лівіше точки – це знак числа, а п'ять цифр правіше точки – мантиса числа в прямому коді. Тут мається на увазі, що *кома фіксована правіше молодшого розряду*, а точка в зображенні числа в цьому випадку просто розділяє знаковий біт від мантиси числа.

Можна використати й іншу форму представлення числа в машинній формі – $[0]00111_2$. Знаковий розряд виділяється квадратними дужками.

Кількість розрядів у розрядній сітці, відведена для зображення мантиси числа, визначає діапазон і точність представлення числа з фіксованою комою. Максимальне за абсолютною величиною двійкове число зображується одиницями в усіх розрядах, за винятком знакового, тобто для цілого числа [5,8,14,15,24,27]:

$$|A|_{max} = (2^{(n-1)} - 1),$$

де n – повна довжина розрядної сітки.

У випадку 16-розрядної сітки $|A|_{max} = (2^{(16-1)} - 1) =$ 32767_{10}, тобто діапазон представлення цілих чисел у цьому випадку буде від $+32767_{10}$ до -32767_{10} .

Для випадку, коли кома фіксується правіше молодшого розряду мантиси, тобто для цілих чисел, числа, у яких модуль більше, ніж $(2^{(n-1)} - 1)$ і менше одиниці, не представляються у формі з фіксованою комою. Числа, за абсолютною величиною менше одиниці молодшого розряду розрядної сітки, називаються в цьому випадку машинним нулем. **Від'ємний нуль заборонений.**

У деяких випадках, коли можна оперувати лише модулями чисел, усю розрядна сітка, включаючи найстарший розряд, відведено для представлення числа, що дозволяє розширити діапазон зображення чисел.

У інформаційній системі з метою спрощення виконання арифметичних операцій застосовують спеціальні двійкові коди для представлення від'ємних чисел: обернений і додатковий. За допомогою цих кодів спрощується визначення знака результату операції при

алгебраїчному додаванні. Операція віднімання (або алгебраїчного додавання) зводиться до арифметичного додавання операндів, полегшується вироблення ознак переповнення розрядної сітки. Як наслідок спрощуються пристрої інформаційної системи, які виконують арифметичні операції.

Одним із способів віднімання є заміна знака числа, що віднімається, на протилежний і додавання його до зменшуваного числа $A–B =A + (-B)$.

Таким чином, операцію арифметичного віднімання заміняють операцією алгебраїчного додавання, яку можна виконати за допомогою двійкових суматорів.

Для машинного представлення від'ємних чисел використовують *прямий, додатковий і обернений коди.*

Доповнивши запис числа у двійковому коді знаковим розрядом, який дорівнює нулю для додатних чисел й одиниці – для від'ємних, отримаємо запис знака й модуля числа в *прямому коді*, наприклад, +6=0.110, -6=1.110.

Обернений двійковий код від'ємного числа утворюється з прямого коду рівного йому за модулем додатного числа інвертуванням значень усіх його розрядів, наприклад, обернений код числа -6 дорівнює 1.001.

Додатковий код від'ємного числа утворюється з його оберненого коду додаванням одиниці до молодшого розряду, наприклад, додатковий код числа -6 дорівнює 1.010.

У табл. 2.7 наведено десяткові числа зі знаком та їх еквівалентні представлення в прямому, оберненому й додатковому кодах. Старший розряд слугує для представлення знака, а інші три розряди задають абсолютне значення числа.

У загальному випадку справедливими є такі вирази:

Прямий код числа A-$[A]_{пр}$. Нехай $A=a_1, a_2,\ldots a_m$;

Якщо $A>0$, то $[A]_{пр} =0, a_1, a_2,\ldots a_m$;

Якщо $A<0$, то $[A]_{пр} =1, a_1, a_2,\ldots a_m$;

Якщо $A=0$, то є неоднозначність: $[0]_{пр} =0,0$ або $=1,0\ldots$

Узагальнюючи результати, отримаємо:

$$[A]_{пр} = \begin{cases} A, & якщо \quad A \geq 0, \\ 1 - A, & якщо \quad A < 0. \end{cases}$$

Обернений код числа A-$[A]_{об}$. Позначка $\bar{a}$ означає величину, обернену *а (інверсію а)*, тобто якщо a=1, то $\bar{a}$ =0, і навпаки:

Якщо $A>0$, то $[A]_{об} =[A]_{пр} =0, a_1, a_2,\ldots a_m$;

Якщо $A<0$, то $[A]_{об} =1, \bar{a}_1, \bar{a}_2,\ldots \bar{a}_m$;

Якщо $A=0$, то є неоднозначність: $[0]_{об} =0,0..0$ або $=1,11\ldots1$.

Таблиця 2.7

Приклади представлення чисел зі знаком у 4-розрядних двійкових кодах

Десяткова цифра зі знаком	Код			Десяткова цифра зі знаком	Код		
	прямий	обернений	додатковий		прямий	обернений	додатковий
+7	0.111	0.111	0.111	-0	1.000	1.111	0.000
+6	0.110	0.110	0.110	-1	1.001	1.110	1.111
+5	0.101	0.101	0.101	-2	1.010	1.101	1.110
+4	0.100	0.100	0.100	-3	1.011	1.100	1.101
+3	0.011	0.011	0.011	-4	1.100	1.011	1.100
+2	0.010	0.010	0.010	-5	1.101	1.010	1.011
+1	0.001	0.001	0.001	-6	1.110	1.001	1.010
+0	0.000	0.000	0.000	-7	1.111	1.000	1.001

Узагальнюючи результати, отримаємо:

$$[A]_{об} = \begin{cases} A, & якщо \quad A \geq 0, \\ 10 - 1 \cdot 10^{-n} + A, & якщо \quad A \leq 0. \end{cases}$$

Додатковий код числа A-$[A]_{дод}$:
Якщо $A \geq 0$, то $[A]_{дод} = [A]_{пр} = 0, a_1, a_2, \ldots a_m$;
Якщо $A < 0$, то $[A]_{дод} = 1, \bar{a}_1, \bar{a}_2, \ldots \bar{a}_m + 0{,}000 \ldots 1$;
Узагальнюючи результати, отримаємо:

$$[A]_{дод} = \begin{cases} A, & якщо \quad A \geq 0, \\ 10 + A, & якщо \quad A < 0. \end{cases}$$

Розглянемо випадок, коли лише два розряди для представлення чисел у десятковій системі числення. Тоді максимальне число, яке можна зобразити, буде 99, а вага третього неіснуючого старшого розряду буде 10^2, тобто 100. У такому випадку для числа 20 додатковим буде число 80, яке доповнює 20 до 100 (100 – 20 = 80). Розглянемо схожий приклад для чисел, представлених 4-розрядним двійковим кодом. Знайдемо додаткове число для $0010_2 = 2_{10}$. Треба від [1]0000 відняти [0]0010, отримаємо число [0]1110, яке і є додатковим кодом 2. Розряд, зображений у квадратних дужках, насправді не існує. Але оскільки в нас 4-розрядна сітка, то виконати таке віднімання в принципі неможливо, а тим більше ми стараємось позбутися віднімання. Тому додатковий код числа отримують способом, описаним раніше, тобто спочатку отримують обернений код числа, а потім додають до нього 1. Проробивши все це з нашим числом (2), неважко переконатися, що вийде аналогічна відповідь.

Додатковий і обернений коди використовують лише для представлення від'ємних двійкових чисел у формі з фіксованою комою. Додатні числа в цих кодах не змінюють свого зображення й представляються, як у прямому коді.

Таким чином, цифрові розряди від'ємного числа в *прямому коді* залишаються незмінними, а в знаковій частині записується одиниця.

У таблиці 2.8 наведений приклад такого представлення числа.

Таблиця 2.8

Представлення числа в прямому, оберненому й додатковому кодах

Число	Прямий код	Обернений код	Додатковий код
-7	$[-7]_{пр}= 1.0001112$	$[-7]_{об}=1.1110002$	$[-7]_{доп} = 1.1110012$

Розглянемо випадок, коли процедура віднімання за допомогою представлення від'ємника в додатковому коді зводиться до процедури додавання. Віднімемо від 10 число 7. Якщо обидва операнди представлені в прямому коді, то процедура віднімання виконується так (див. табл. 2.9):

Таблиця 2.9

Процедура віднімання в прямому й оберненому кодах

Дія	Прямий код	Обернений код
10-7=3	0.001010 -1.000111 $0.000011= 3_{10}$	0.001010 + 1.111001 $1\ 0.000011= 3_{10}$.

У цифрових автоматах для представлення від'ємних чисел у форматі з фіксованою комою зазвичай використовується додатковий код.

2.6. Форма представлення чисел з плаваючою комою

Показова форма, або форма з плаваючою комою, застосовується для розширення діапазону й зменшення відносної похибки представлення чисел в обчислювальних пристроях.

У загальному випадку число у формі з плаваючою комою представляється у вигляді:

$$A = \pm m \cdot q^{\pm p},$$

де m – мантиса числа;

q – основа системи числення;

p – порядок числа, який для спрощення в прикладах будемо іноді зображувати як $\pm p$ *(не плутайте з перенесенням у старший розряд)*.

Тоді очевидно, що p – це показник ступеня порядку, який зазвичай називають просто порядком числа, оскільки в основному завжди q = 2. Отже, попередній вираз можна записати в такому вигляді:

$$A = \pm m_A \cdot \pm p_A,$$

маючи на увазі, що в інформаційних системах зазвичай q = 2.

Так, число 1964 у формі з плаваючою комою в десятковій системі числення можна записати так: $1964{,}0 \cdot 10^0$; $0{,}1964 \cdot 10^4$; $19{,}64 \cdot 10^2$; $196400 \cdot 10^{-2}$ і т. д.

Неоднозначність запису полягає в тому, що мантиса може приймати безкінечну кількість різних значень, менших за одиницю, при відповідних значеннях порядку (тобто кома в мантисі може плавати).

Число з плаваючою комою прийнято представляти в так званому *нормалізованому* вигляді для максимально точного представлення числа. Якщо виконується нерівність

$$q^{-1} \le | m | < 1,$$

а у випадку двійкової системи числення:

$$0.5 \le | m | < 1,$$

то вважається, що число представлено в нормалізованому вигляді. Наприклад, $0{,}1964 \cdot 10^4$ є нормалізованим виглядом числа 1964 у формі з

плаваючою комою в десятковій системі числення. Аналогічні міркування застосуємо й до двійкових чисел. Наприклад, нормалізована форма запису двійкового числа $1110110{,}011_2$ має такий вигляд: $0{,}1110110011 \cdot 10^{111}$.

Таким чином, у двійкового нормалізованого числа у формі з плаваючою комою мантиса – правильний дріб і в старшому розряді мантиси завжди стоїть 1. Операція приведення числа до нормалізованого вигляду називається *нормалізацією*. Нормалізація чисел у цифрових автоматах виконується або автоматично, або ж за спеціальною програмою.

Оскільки система числення для заданого цифрового автомата (комп'ютера) залишається постійною, то при представленні числа у форматі з плаваючою комою немає необхідності вказувати її основу, достатньо лише представити показник ступеня порядку числа.

Для представлення двійкового числа у формі з плаваючою комою в розрядній сітці слід виділити дві групи розрядів. Перша група (*r*-розрядів) призначена для розміщення мантиси, друга (*n*-*r*-розрядів) – для розміщення коду порядку (без урахування знакових розрядів мантиси й порядку), див. рис. 2.6.

Будемо вважати, що двійкові числа розміщуються в розрядній сітці в нормалізованому вигляді. Це означає, що числові значення мантиси завжди більші або дорівнюють 2^{-1}, але не перевищують 1. „Вага" молодшого розряду мантиси дорівнює 2^{-r}, а „вага" старшого розряду – 2^{-1}. Максимально представлене число буде при максимальних значеннях мантиси й порядку: $A = (1 - 2^{-r}) \cdot 2^{(2^{n-r}-1)}$, а мінімально представлене число при мінімальному значенні мантиси й максимальному за модулем від'ємному значенні порядку: $A = 2^{-1} \cdot 2^{(2^{n-r}-1)}$.

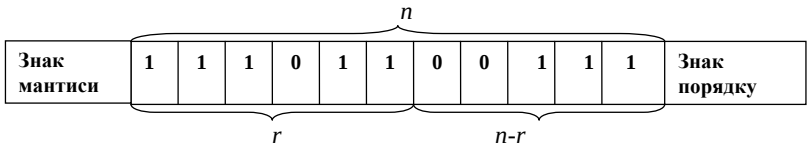


Рис. 2.6. Представлення числа з плаваючою комою

Зазвичай у форматі з плаваючою комою замість показника *p* використовують так звану характеристику („зміщений порядок") γ :

$$\gamma = \pm p + l \ ,$$

де *l* – надлишок (зміщення), значення якого підбирається таким чином, щоб при зміні значення показника від деякого мінімального значення $-|p_{\max}|$

до максимального $+|p_{max}|$, характеристика *r* змінювалася від 0 до r_{max}. Отже, характеристика не змінює свого знака.

Розглянемо кілька прикладів представлення чисел у формі з плаваючою комою. Попередньо нагадаємо, що показник ступеня двійки в розрядах розрядної сітки довжиною *n*, відведеною для представлення цілих чисел, змінюється від *0* до *n-1*, а у випадку правильних дробових чисел – від *-1* до *-n*.

Якщо для представлення показника порядку виділено чотири розряди, то $A = 2^3 = 8_{10} = 1000_2$. Для цього випадку в табл. 2.10 наведено значення показника порядку й мантиси для деяких чисел, представлених у формі з плаваючою комою.

Довжина розрядної сітки, виділеної під характеристику, визначає діапазон представлення чисел у форматі з плаваючою комою.

Основною перевагою представлення чисел у формі з плаваючою комою є великий діапазон машинних чисел і висока точність їх представлення.

Діапазон визначається довжиною розрядної сітки, виділеної під характеристику, а точність – довжиною розрядної сітки, виділеної під мантису.

Числа з фіксованою комою найчастіше мають формат слова (2 байти) або півслова (1 байт). Числа з плаваючою комою – формат подвійного (4 байти) і розширеного слова (8 байт) (математичні співпроцесори інформаційної системи можуть працювати з 10-байтовими словами). Послідовність кількох бітів або байтів часто називають *полем даних.* Поля перемінної довжини можуть мати будь-який розмір від 0 до 255 байтів, але він обов'язково повинен дорівнювати цілому числу байтів.

Таблиця 2.10

Показники порядку, характеристики й мантиси чисел, представлених у формі з плаваючою комою

Число A_{10}	**Порядок числа** p_{10}	**Мантиса** m_2
0	0	0,0
1	1	0,1
2	2	0,1
3	2	0,11
0,5	0	0,1
0,25	-1	0,1
0,75	0	0,11
0,375	-1	0,11

Як приклад розглянемо запис десяткового числа 193_{10} (11000001_2) у розрядній сітці цифрового автоматау, для числа з фіксованою комою у форматі слова зі знаком (рис. 2.7) і для числа з плаваючою комою у форматі подвійного слова (рис. 2.8).

Розряд	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Число	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**
	Знак	**Абсолютна величина числа**														

Рис. 2.7. Число з фіксованою комою у форматі слова зі знаком

Розряд	31	30	29	28	27	26	25	24	23	22	20	21	19	18	17	16	15	…	1	0
Число	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**
	Знак	**Порядок**							**Мантиса**											

Рис. 2.8. Число з плаваючою комою у форматі подвійного слова

Двійково-кодовані десяткові числа можуть бути представлені в цифрових автоматах полями перемінної довжини в так званому упакованому (рис. 2.9) і розпакованому форматах (рис. 2.10). В упакованому форматі для кожної десяткової цифри відводиться по 4 двійкових розряди (півбайта), при цьому знак числа кодується в крайньому правому півбайті числа (1100 – знак „+” і 1101 – знак „-”).

На рис. 2.9 і 2.10 Цф – цифра, знак – знак числа. Упакований формат використовується в цифрових автоматах під час виконання операцій додавання й віднімання двійково-десяткових чисел. У розпакованому форматі для кожної десяткової цифри виділено по цілому байту, при цьому старші півбайти (зона) кожного байта, крім молодшого, у цифрових автоматах заповнюються кодом 0011, а в молодших (лівих) півбайтах звичним чином кодуються десяткові цифри. Старший півбайт (зона) наймолодшого байта використовується для кодування знака числа (див. рис. 2.10).

Цф	Цф	…	Цф	Знак

Байт

Рис. 2.9. Структура поля упакованого формату

Зона	Цф	..	Цф	..	Зона	Цф	Знак	Цф

Рис. 2.10. Структура поля розпакованого формату

Наприклад, десяткове число -193_{10} =$-000110010011_{2\text{-}10}$ у цифровому автоматі буде представлено відповідно в упакованому й розпакованому форматах, як це показано на рис. 2.11 і 2.12.

0001	1001	0011	1101

Рис. 2.11. Структура поля упакованого формату

0011	0001	0011	1001	1101	0011

Рис. 2.12. Структура поля розпакованого формату

2.7. Переведення чисел з формату з фіксованою комою у формат з плаваючою комою й навпаки

Спочатку розглянемо алгоритм переведення числа з формату з фіксованою комою у формат з плаваючою комою.

Переведення числа з формату з фіксованою комою $[A]_{\phi}$ у формат з плаваючою комою $[A]_{п.з}$ можливий лише у випадку виконання такої умови:

$$n_m > n_{\phi},$$

де n_m – довжина мантиси для числа у форматі з плаваючою комою;

n_{ϕ} – довжина мантиси для числа у форматі з фіксованою комою.

Припустимо, що довжина розрядної сітки, виділеної для зображення мантиси числа й з плаваючою, і з фіксованою комою однакова і, наприклад, дорівнює 8, тобто $n_{\phi} = n_m = 8$. Для представлення характеристики числа з плаваючою комою, наприклад, відведено 6 розрядів. Тоді $l = 32_{10}$. На початку процедури переведення модуль числа $[A]_{\phi}$, представленого в прямому коді без знакового розряду, переписується в поле мантиси $[A]_{п.з.}$ Причому, таким чином, щоб незалежно від довжини мантиси зображення модуля числа $[A]_{\phi}$ розмістилося в наймолодших розрядах поля мантиси. У цьому випадку вихідна характеристика приймається такою: $r = l + n_{\delta} = 32_{10} + 8_{10} = 40_{10}$. Це пояснюється тим, що під час переведення числа з формату з фіксованою комою у формат з плаваючою комою показник ступеня порядку p може бути меншим або дорівнювати n_{ϕ}. У процесі переведення перевіряється старший розряд мантиси, якщо він установлений на нульову позначку, то здійснюється зсув

зображення мантиси вліво на один розряд і віднімається 1 із значення *r*. Така процедура повторюється до того часу, поки в старшому розряді мантиси не з'явиться 1 або ж поки значення *r* не дорівнюватиме *l*. У цих випадках процедура переведення завершується.

У випадку, коли початкове число є правильним дробом, то дріб записують у старші розряди мантиси й вихідною характеристикою приймається *r=l*. Далі переведення здійснюється аналогічно до розглянутих вище прикладів.

2.8. Похибки представлення чисел у цифрових автоматах

Як уже було відзначено, представлення числової інформації в цифровому автоматі зазвичай тягне за собою появу похибок (помилок), величина яких залежить і від форми представлення чисел, і від довжини розрядної сітки цифрового автомата. Отже, під час переведення чисел з одної системи числення в іншу неминуче виникають похибки.

Абсолютна похибка представлення – це різниця між істинним значенням вхідної величини *A* та її значенням, отриманим з машинного зображення $A_м$, тобто $\Delta[A]=A-A_м$.

Відносна похибка представлення дорівнює – $\delta[\mathrm{A}] = \left|\Delta[\mathrm{A}]/\mathrm{A}_{\mathrm{M}}\right|$.

У табл. 2.11 показано діапазони представлення чисел і зміни відносних похибок для форми запису з фіксованою й плаваючою комою.

Таблиця 2.11

Діапазони представлення чисел і зміни відносних похибок для форми запису з фіксованою й плаваючою комою

Форма представлення числа	Кома			
	Фіксована		Плаваюча	
Межа	Нижня	Верхня	Нижня	Верхня
Діапазон чисел	$2^{-(n-r)}$	$2^r - 2^{-(n-r)}$	$2^{-(2^{n-r})}$	$(1-2^r)\cdot 2^{(2^{n-r}-1)}$
Діапазон зміни похибок	$1/(2^n-1)$	1	$1/(2^{r-1})$	$2^{-(r-1)}$
де *n* – кількість розрядів сітки цифрового автомата *r* – кількість гнізд для розміщення дробової частини				

Запитання для самоперевірки

1. Що таке система числення?
2. Які системи числення використовують у цифрових автоматах?
3. Виконайте операції переведення кількох чисел (наприклад, 139 і 486) з десяткової системи числення у двійкову.
4. Дайте стислу характеристику форм представлення чисел з плаваючою й фіксованою комою (точкою).
5. Як здійснюється переведення неправильних дробів з одної системи числення в іншу?
6. Що таке нормалізована форма представлення числа?
7. Які критерії вибору системи числення для цифрових автоматів?
8. Які формальні правила двійкової арифметики Ви можете назвати?
9. Сформулюйте правила переведення числової інформації з одної позиційної системи числення в іншу.
10. Як визначають похибки представлення чисел у цифрових автоматах?
11. Що таке поле даних?
12. Що таке упакований і розпакований формат?
13. Що таке прямий, додатковий і обернений коди?
14. Дайте визначення форми представлення числа в цифровому автоматі?
15. Що становить собою машинне (автоматне) зображення числа?
16. Що таке абсолютна похибка представлення числа в цифровому автоматі?
17. Назвіть переваги представлення чисел у формі з плаваючою комою?
18. Які формати мають числа з фіксованою комою?
19. Для чого в цифрових автоматах використовують додатковий і обернений коди?
20. Що становлять собою елементи з тригерною характеристикою?
21. Виконайте операції переведення кількох чисел з двійкової системи числення в шістнадцяткову?
22. Що таке ASCII-коди? Наведіть їх структуру й укажіть призначення.
23. Розгляньте ASCII-коди представлення десяткових цифр.
24. Що таке відносна похибка представлення числа в цифровому автоматі?

Література для самостійної підготовки за темою:

8, 14, 15, 23, 24, 27, 35.

Розділ 3. АРИФМЕТИЧНІ ДІЇ В ЦИФРОВИХ АВТОМАТАХ

3.1. Додавання двійкових чисел

Операція додавання цілих багаторозрядних чисел у двійковій системі числення здійснюється відповідно до табл. 2.4. Нагадаємо, що внаслідок додавання 1+1 отримуємо $10_2=2$. Для представлення такого результату потрібні два розряди. При цьому з молодшого розряду в наступний, старший розряд, надходить одиниця переносу. Перенос додається із вмістом відповідного розряду чисел. На рис. 3.1 показано схему додавання цілих багаторозрядних чисел без знака для шестирозрядної сітки.

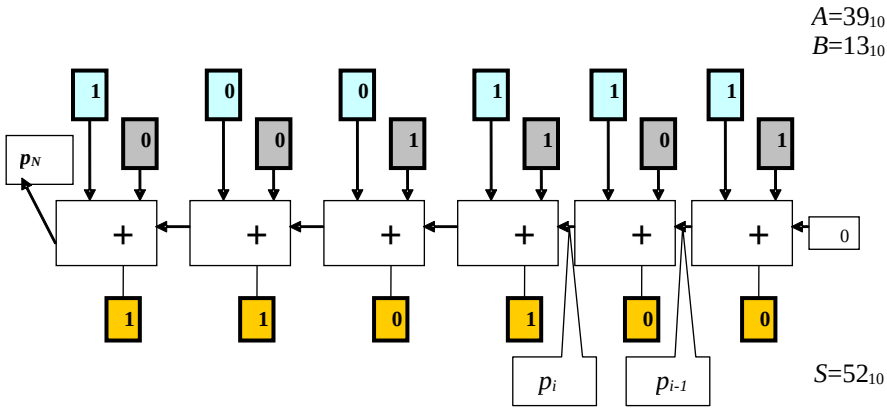


Рис. 3.1. Схема додавання цілих багаторозрядних чисел без знака

Додавання здійснюється послідовно, починаючи з молодшого розряду. В кожному розряді додаються значення трьох величин: розряду операнда a_i; розряду операнда b_i; переносу з попереднього розряду p_{i-1}. У результаті додавання в кожному розряді утворюється дві величини: значення суми s_i; значення переносу в наступний, старший розряд p_i.

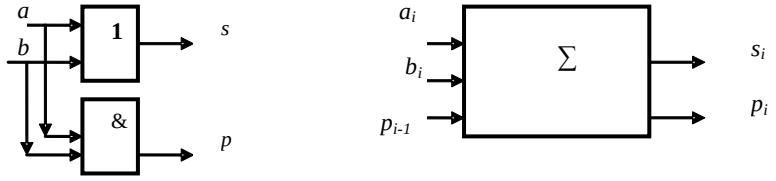
Значення s_i і p_i залежно від a_i, b_i, і p_{i-1} наведено в табл. 3.1. У підсумку отримали число, що містить сім розрядів, старший з яких відповідає розряду переносу, що знаходиться за межами розрядної сітки (прийняли шість розрядів). Таким чином, настало переповнення розрядної сітки.

Таблиця 3.1

Додавання однорозрядних двійкових чисел з урахуванням переносів

Розряд операнда, a_i,	Розряд операнда, b_i	Перенос з попереднього розряду, p_{i-1}	Сума, s_i	Значення переносу в наступний, старший розряд, p_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Додавання двійкових чисел в обчислювальних пристроях здійснюють комбінаційні суматори. Одним з найпростіших пристроїв додавання є однорозрядний півсуматор, схему якого представлено на рис. 3.2 а. Пристрій здійснює додавання двох однорозрядних чисел, сума *s* яких також має один розряд. На виході пристрою формується сигнал переносу *p* у наступний, старший розряд, якщо обидва числа, що додаються, мають одиничні значення. Два однорозрядних півсуматори утворюють однорозрядний суматор, схему якого показано на рис. 3.2 б.



а б

Рис. 3.2. Схеми комбінаційних суматорів

Арифметичні операції можна виконувати з двійковими числами, представленими в прямому, оберненому й додатковому кодах. Якщо операнди представлені в прямому коді й мають одинакові знаки, то над ними при алгебраїчному додаванні виконується процедура додавання. Якщо ж операнди мають різні знаки, то виконується процедура віднімання. Як уже було сказано, для спрощення апаратних засобів інформаційної системи процедуру віднімання замінюють додаванням завдяки тому, що від'ємний операнд представлено в оберненому або додатковому коді.

3.1.1. Алгебраїчне додавання чисел, представлених у формі з фіксованою комою

Додавання двійкових чисел, представлених у формі з фіксованою комою, здійснюється аналогічно сумуванню цілих чисел. Номери розрядів визначаються їх розташуванням у числі відносно коми, яка відділяє цілу частину від дробової. Так, прямий код числа $+118{,}375_{10}$ може бути представлений як $1.0001001{,}101_2 = -118{,}375_{10}$.

Порядок додавання не залежить від положення коми.

Якщо обидва операнди представлені в прямому коді й мають однакові знаки, то над такими операндами виконують процедуру додавання й результату присвоюють знак вихідних операндів.

Розглянемо приклади.

Додавання двох додатних чисел.

+39	0.100111
+13	+0.001101
+52	0.110100

Результат додавання не виходить за межі розрядної сітки. Отже, він правильний. Ще один приклад.

+52	0.110100
+13	+0.001101
-63	1.000001

У результаті додавання вийшло від'ємне число -63, тобто додавання неправильне, оскільки правильна відповідь 65 виходить за межі розрядної сітки.

При додаванні двох чисел, представлених в оберненому або додатковому коді, їх знакові розряди додаються, причому можливий перенос одиниці із старшого цифрового розряду в знаковий.

Прямий код	*Додатковий код*	*Обернений код*
$2_{10} + 4_{10} = 6_{10}$	$-2_{10} - 4_{10} = -6_{10}$	$-2_{10} - 4_{10} = -6_{10}$
0.0010	1.1110	1.1101
+0.0100	+ 1.1100	+ 1.1011
0.0110 = 6_{10}	1←1.1010 = -6_{10}	1 1.1000
		+1 корекція
		11001 = -6_{10}

Якщо ж операнди мають різні знаки, то повинна виконуватися процедура віднімання. Але, як уже було відзначено, процедуру віднімання заміняють процедурою додавання, від'ємний операнд (від'ємник) представляють в оберненому або додатковому коді, а додатний операнд – у прямому коді.

Розглянемо приклади.

Додавання двох чисел з різними знаками.

- 52	1.001100
+13	0.001101
- 39	1.011001

Результат додавання правильний і має знак більшого за модулем числа. При сумуванні чисел з різними знаками модуль результату завжди не перевищує модуля будь-якого з вихідних операндів. Тому переповнення розрядної сітки не відбувається. Розглянемо ще кілька прикладів.

$4_{10} - 2_{10} = 2_{10}$	$4_{10} - 2_{10} = 2_{10}$
0.0100	0.0100
+ 1. 1110	+ 1.1101
1←0. 0010 = 2_{10}	1←0.0001
	+1 корекція
	0.0010 = 2_{10}

Використання чисел зі знаком (прямого коду представлення чисел) ускладнює архітектуру цифрового автомата. У деяких цифрових автоматах від'ємні числа представлено у вигляді так званих доповнень. Це дозволяє замінити віднімання чисел додаванням.

Доповненням H n-розрядного числа A називається різниця

$$H=q^n-A,$$

де q – основа системи числення.

У табл. 3.2. показано приклади різноманітних випадків додавання чисел зі знаком.

Таблиця 3.2

Приклади додавання чисел зі знаком

Приклад	Доданки й результат	Примітка
1	A>0, B> 0, C<2^{15} 0 000 0110 0011 1010 A=+ 1594 +0 100 0100 1001 1011 B=+17563 0 100 1010 1101 0101 C=+19157	Результат коректний
2	A>0, B>0, C>=2^{15} 0 100 0100 1001 1011 A=+17563 +0 100 1010 1101 0101 B=+19157 1 000 1111 0111 0000 C=-28816	Переповнення розрядної сітки. Результат некоректний
3	A<0, B<0, |A|+|B| < 2^{15} 1 111 1001 1100 0110 A=- 1594 +1 011 1011 0110 0101 B=-17563 ***1*** 1 011 0101 0010 1011 C=-19157	Перенесення із старшого розряду не враховано. Результат коректний
4	A<0, B<0, |A|+|B| > 2^{15} 1 011 1011 0110 0101 A=-17563 +1 011 0101 0010 1011 B=-19157 ***1*** 0 111 0000 1001 0000 C=+28816	Переповнення розрядної сітки. Результат некоректний
5	A>0, B<0, |A| > |B| 0 100 0100 1001 1011 A=+17563 +1 111 1001 1100 0110 B= - 1594 ***1*** 0 011 1110 0110 0001 C=+15969	Перенесення із старшого розряду не враховано. Результат коректний
6	A>0, B<0, |A| < |B| 0 000 0110 0011 1010 A=+1594 +1 011 1011 0110 0101 B= -17563 1 100 0001 1001 1111 C= -15969	Результат коректний
Резюме: Відповідно до таблиці правило перевірки формулюється так: якщо знак операндів однаковий, а знак суми йому протилежний, то результат буде некоректним; у всіх інших випадках результат додавання правильний.		

Процедура алгебраїчного додавання з додатковими кодами простіша, ніж з оберненими кодами, оскільки в останньому випадку при виникненні одиниці переносу за межі розрядної сітки, виділеної для числа у форматі з фіксованою комою, доводиться коригувати результат за допомогою додаткової процедури додавання одиниці до результату. Нульовий результат отримуємо або в прямому коді, або в оберненому, що також вимагає корекції результату. Тому в наш час на практиці для

представлення від'ємних операндів використовують в основному додатковий код.

Отже, перед виконанням самої процедури алгебраїчного додавання в додатковому коді треба проаналізувати знаки доданків. Якщо вони різні, то виконують алгебраїчне додавання (фактично віднімання) без перевірки на переповнення результату, оскільки його в цьому випадку просто не може бути. Але контролюють, щоб у результаті не з'явився заборонений від'ємний нуль. Якщо ж у вихідних операндів знаки однакові, то такий само знак попередньо присвоюють результату й виконують саме додавання. Якщо знак остаточного результату не збігається з попередньо присвоєним знаком, то це є ознакою переповнення й, отже, неправильного результату.

Під час додавання чисел, які мають однаковий знак і представлені у формі з фіксованою комою, може виникнути переповнення розрядної сітки. Ознакою переповнення при додаванні чисел у прямому коді є поява одиниці переносу із старшого розряду цифрової частини числа, наприклад:

$$\begin{array}{r} 0.\,1010 \\ +\ \underline{0.\,0110} \\ 0.\,0000 \end{array}$$

Ознака переповнення розрядної сітки при додаванні чисел у додатковому й оберненому кодах – отримання знака результату, протилежного знакам операндів, наприклад:

$$\begin{array}{rr} 0.1100 & 1.0101 \\ \underline{+\ 0.1000} & \underline{+\ 1.0111} \\ 1.0100 & \mathbf{1}\ 0.1100 \end{array}$$

Переповнення може виникнути й при множенні будь-яких чисел.

Для виявлення переповнення розрядної сітки в складі цифрового автомата повинні бути передбачені апаратні засоби, які автоматично виробляють сигнал переповнення.

Один з методів виявлення переповнення розрядної сітки передбачає введення допоміжного розряду в знакову частину зображення числа, який називають *розрядом переповнення*. Таке представлення числа називають *модифікованим.*

Ці коди відрізняються від звичайних прямих, обернених і додаткових тим, що мають по два знакових розряди. Під час виконання арифметичних дій над двійковими числами ці два знаки дозволяють легко виявити переповнення розрядної сітки. Якщо вміст цих двох розрядів збігається, то переповнення відсутнє. В іншому випадку в інформаційній системі виробляється управляючий сигнал (сигнал переповнення) або на

зупинку інформаційної системи, або на усунення переповнення розрядної сітки.

Розглянемо приклади додавання двох чисел у модифікованому коді (див. наступний параграф, табл. 3.3).

Таблиця 3.3

Приклади додавання чисел зі знаком у модифікованому коді

Приклад	Доданки й результат	Примітка
1	$A=5_{10}$= 00. 0101_2 $B=3_{10}$ = 00. 0011_2 $A + B = 5_{10} + 3_{10} = 8_{10}$ 00. 0101 + 00. 0011 00. 1000 = 8_{10}	Результат коректний
2	$A= -5_{10}$ = - 0101_2 = 11. 0101 $B = -8_{10}$ = -1000_2 = 11. 1000 $A + B = -13_{10}$ 11. 1011 + 11. 1000 1 11. 0011 = -13_{10}	Результат коректний
3	$A =13_{10}$ = 0.1101 $B =13_{10}$ = 0.1101 $A + B = 26_{10}$ 00. 1101 + 00. 1101 1010	Переповнення розрядної сітки
4	$A =-13_{10}$ = -1101 $B =- 9_{10}$ = - 1001 $A + B = -22_{10}$ 11.0011 + 11.0111 10.1010 Після зсуву вправо отримаємо 11,01010 або - 10110	Переповнення розрядної сітки

3.1.2. Алгебраїчне додавання чисел, представлених у формі з плаваючою комою

Під час виконання будь-яких арифметичних дій над операндами, представленими у формі з плаваючою комою, операції, виконувані над мантисами й порядками (або характеристиками) цих чисел, різні. Тому перед початком будь-якої арифметичної процедури кожен з операндів розбивається на частини. Порядок (характеристика) відділяється від мантиси операнда, щоб можна було виконувати над ними окремі процедури. Після виконання арифметичної дії й процедури нормалізації результату його порядок і мантиса перетворюються у звичайний формат з плаваючою комою.

У випадку алгебраїчного додавання порядки операндів обов'язково мають бути однаковими. Тому на початку процедури додавання й віднімання чисел здійснюють (за необхідності) вирівнювання характеристик операндів. Для цього мантиса операнда з меншою характеристикою зсувається в розрядній сітці вправо з додаванням одиниці до його характеристики при кожному зсуві на один розряд. Ця процедура триває до тих пір, поки характеристики обох операндів не стануть рівними. Отримана таким чином характеристика, однакова для обох операндів, присвоюється як попередня результату операції. Далі здійснюють додавання або віднімання мантис за правилами, аналогічними для чисел з фіксованою комою, зокрема мантиса одного з операндів перетворюється в додатковий код, для того щоб процедуру віднімання звести до додавання. Якщо відповідь отримано в додатковому коді, то його перетворюють у прямий, оскільки мантиса числа з плаваючою комою – це модуль числа. Далі, за необхідності, виконують нормалізацію й округлення відповіді. У процесі нормалізації мантиса порозрядно зсувається вліво або вправо. При зсуві вліво на кожний розряд віднімається одиниця з характеристики, попередньо присвоєної відповіді. При кожному зсуві вправо – до неї додається одиниця. Зсув мантиси вправо необхідний у тих випадках, коли під час додавання мантис відбулося переповнення розрядної сітки. У ході цих операцій визначаються остаточні значення характеристики й мантиси відповіді.

Можливі два випадки денормалізації:

1) денормалізація вліво, яка відповідає переповненню розрядної сітки;

2) денормалізація вправо, яка виникає, коли в прямому коді мантиси після коми є один або кілька нульових розрядів.

Наявність денормалізації вліво визначається тими самими способами, що й переповнення розрядної сітки. Для денормалізації вправо характерне однакове значення розрядів мантиси по обидва боки від коми.

Спочатку розглянемо приклад додавання двох чисел, представлених у показовій формі A = $(+ 0{,}10101 \cdot 10^{101})_2$, B = $(- 0{,}11001 \cdot 10^{011})_2$. Зрівняємо порядок другого числа до порядку першого – B = $(-0{,}0011001 \cdot 10^{101})_2$.

Далі сумуємо мантиси:

$$\begin{array}{r} +\ 0.1010100 \\ -\ 0.0011001 \\ \hline +\ 0.0111011 \end{array}$$

Результат отримали в денормалізованій формі, оскільки в мантисі розряд, який іде за комою, має нульове значення. Прямий код додатної мантиси результату 0.0111011 має однакові, рівні 0, розряди по обидва боки від коми. Для нормалізації необхідно зсунути всі розряди мантиси вліво на один розряд і зменшити на одиницю значення порядку. Остаточно отримаємо $+0.1110110 \cdot 10^{100}$.

Ще один приклад. Маємо 8-розрядну мантису й 6-розрядний порядок, зміщення дорівнює 1000_2. Складемо два числа з мантисами m_1=0.10100000, m_2=0.10000000 і з характеристиками p_1=001011, p_2= 001010. Оскільки порядки різні, необхідно попередньо виконати їх вирівнювання. Тоді, $p_1 - p_2$ = 000001. Отже, треба зсунути m_2 на 1 розряд вправо, а до характеристики p_2 додати 1. Після перетворень отримаємо: m_2=0.01000000, p_2 = 001010 + 000001 = 001011. Складемо мантиси, отримаємо: $m=m_1+m_2$=0.10100000+0.01000000=0.11100000, p=001011. Нормалізація відповіді не потрібна.

3.2. Множення двійкових чисел

Для двійкової системи числення найбільш відомими є такі основні способи виконання операції множення:

1) множення починаючи з молодших розрядів множника;

2) множення починаючи зі старших розрядів множника.

У табл. 3.4 показано приклади.

В обох випадках операція множення складається з кількох послідовних операцій додавання часткових добутків. Операціями додавання управляють розряди множника. Якщо в якомусь розряді множника знаходиться одиниця, то до суми часткових добутків додається множене з відповідним зсувом (вліво або вправо), якщо в розряді множника – нуль, то множене не додається, але враховується, що в наступній операції аналізу розряду множника треба зробити додатковий зсув. Якщо, наприклад, у наступному після нульового розряду множника є 1, то множене зсувається на 2 розряди й додається до суми часткових добутків. Якщо підряд траплятиметься 0, то стільки додаткових зсувів

множеного треба буде зробити, коли в черговому розряді трапиться 1, а потім додавати множене до суми часткових добутків.

Таблиця 3.4

Способи виконання операції множення

Множення починається з молодших розрядів множника	Множення починається зі старших розрядів множника
A=1101 (множене) B=1101 (множник) 1101 x 1101 1101 + 0000 1101 1101 (часткові добутки) **10101001** Результат	A=1101 (множене) B=1101 (множник) 1101 x 1101 1101 + 1101 0000 1101 **10101001**

Таким чином, крім операції додавання чисел, для отримання добутку необхідна операція зсуву числа.

3.2.1. Множення чисел, представлених у формі з фіксованою комою

Множення двійкових чисел зазвичай виконують у прямому коді. Знак добутку визначають за знаковими розрядами множеного й множника відповідно до такого правила: **якщо знак операндів однаковий, то знак добутку – додатний; в іншому випадку знак добутку від'ємний**.

Знак добутку двох чисел не впливає на алгоритм виконання операції множення модулів цих чисел.

Розглянемо один з можливих варіантів алгоритму множення, коли операнди представлені в прямому коді. Перед виконанням самої процедури множення за звичними арифметичними правилами множення визначають і запам'ятовують знак добутку. Потім обидва операнди представляються в прямому коді й виконують процедуру множення одним з двох раніше описаних методів з обов'язковим контролем переповнення розрядної сітки.

У табл. 3.5 показано приклади операції множення для двох чисел з фіксованою комою.

Множення чисел, представлених у формі з фіксованою комою, можна організувати не лише на двійковому суматорі прямого коду, але й на двійкових суматорах оберненого або додаткового коду. У цьому випадку добуток додаткових або обернених кодів співмножників дорівнює додатковому або оберненому коду відповідно лише у випадку додатного

множника. Якщо ж множник від'ємний, то здійснюють корекцію результату.

Таблиця 3.5

Приклади операції множення

Приклад	Десяткові числа	Двійкові числа
1	$A=5_{10}$, $B=3_{10}$, $A \cdot B = 5_{10} \cdot 3_{10} = 15_{10}$ 5 x 3 **15**	A=0.0101, B= 0.0011 0.0101 x 0.0011 0101 + 0101 **0.1111 = 15_{10}**
2	$A=13_{10}$, $B=11_{10}$ $A \cdot B = 13_{10} \cdot 11_{10} = 143_{10}$ 13 x 11 13 13 **143**	A=1101, B=1011 1101 x 1011 1101 1101 0000 +1101 **10001111**

3.2.2. Множення чисел, представлених у формі з плаваючою комою

Під час виконання операції множення двох чисел, представлених у формі з плаваючою комою, їх мантиси множать, як числа з фіксованою комою, а порядки складають.

В обох випадках перевірка на переповнення є обов'язковою. Але оскільки реально складають характеристики чисел, то множення виконують за такою формулою:

$$A_1 \cdot A_2 = m_1 \cdot 2^{p_1} \cdot m_2 \cdot 2^{p_2} = ,$$
$$= (m_1 \cdot m_2) \cdot 2^{p_1 + p_2 - l}$$

де $p_1 + p_2 - l$ – характеристика результату.

Знак відповіді визначають звичайним способом. Якщо відповідь вийшла ненормалізована, то виконують процедури нормалізації й округлення відповіді.

Наявність денормалізації вліво визначають тими самими способами, що й переповнення розрядної сітки. Для денормалізації вправо характерне однакове значення розрядів мантиси по обидва боки від коми.

Приклад знаходження добутку двох чисел, представлених у показовій формі $A=(+0{,}10101\cdot 10^{100})_2$, $B=(-0{,}10001\cdot 10^{011})_2$. Мантиса добутку в денормалізованій формі – 0,0101100101 має від'ємний знак, порядок результату дорівнює 111. Після нормалізації значення порядку має бути зменшене на одиницю. Остаточно отримаємо – $0{,}101100101\cdot 10^{110}{}_2$, тобто результат представлений у нормальній формі.

За часом виконання операція множення належить до тривалих операцій. Витрати часу на множення двох чисел у прямому коді можна оцінити такою формулою (для випадку послідовного аналізу розрядів множника):

$$t_{множ} = \sum_{i=1}^{n} (t_{зсув} + P_i t_{сл}),$$

де $t_{зсув}$ – час виконання зсуву числа на один розряд;
$t_{сл}$ – час додавання на суматорі;
P_i – імовірність появи одиниці в розрядах множника;
n – кількість розрядів множника.

Існує кілька методів прискорення процедури множення: аналіз двох розрядів множника одночасно, аналіз довільної кількості розрядів множника, множення в системі числення з основою $q = 2^k$ і матричні методи множення.

У зв'язку з тим, що в цьому випадку в процесі виконання операції множення на кожному циклі операції аналізується одразу два розряди множника, то таких циклів знадобиться $n/2$, де n – довжина розрядної сітки множника без урахування знакового розряду. Ця кількість циклів записується в деякий лічильник *SC*. При встановленні на нульову позначку вмісту лічильника *SC* процедура множення зупиняється. Зазвичай аналізують два молодших розряди множника, тому наприкінці кожного циклу здійснюють одночасний зсув на два розряди вправо зображення суми окремих добутків і множника. Причому, таким чином, щоб при кожному такому зсуві черговий молодший розряд числа потрапляв у старший розряд мантиси множника.

3.3. Ділення двійкових чисел

3.3.1. Ділення двійкових чисел, представлених у формі з фіксованою комою

Ділення двійкових чисел багато в чому подібне до ділення десяткових чисел. Алгоритм ділення полягає в тому, що дільник на кожному кроці віднімається з діленого стільки разів (починаючи зі старших розрядів), скільки це можливо для отримання найменшої додатної остачі. Тоді в черговий розряд частки записується цифра, яка дорівнює числу дільників, що містяться в діленому на цьому кроці. Отже, під час ділення операцію віднімання повторюють до тих пір, поки зменшуване не стане меншим, ніж від'ємник. Кількість цих повторень показує, скільки разів від'ємник укладається в зменшуване. У табл. 3.6 показано приклади операції ділення для двох чисел з фіксованою комою.

Таблиця 3.6

Приклади операції ділення

Приклад	Ділене, дільник, результат
1	$A=45_{10}=1001101_2$, $B=5_{10}=101_2$, $A/B=45_{10}/5_{10}=9_{10}$ 001101 / 101 01 010 0101 1001
2	$A=204_{10}=11001100_{(2)}$, $B=12_{10}=1100_{(2)}$, $A/B=204_{10}/12_{10}=17_{10}$ 11001100 | 1100 1100 | 10001 остача 00001 - 0 11 - 0 110 - 0 1100 - 1100 0000

Пояснимо приклад 2, наведений у табл. 3.6. Двійкове, як і десяткове ділення, починається з аналізу діленого (A=11001100) і дільника (B=1100). Оскільки дільник укладається в 1100, то одиниця записується в старший

розряд поля частки. Дільник множиться на 1 і віднімається із 1100. Різниця дорівнює 0. Об'єднується 0 остачі зі значенням наступного розряду діленого, що дорівнює 1. Оскільки дільник (B=1100) 0 разів укладається в 1, записуємо 0 в наступний за старшинством розряд поля частки, а число 1 об'єднується з наступним розрядом діленого й т. д. до тих пір, поки ділене не виявляється вичерпаним.

У таблиці 3.7 наведено приклад операції ділення з відновленням остачі [5, 7, 8,14, 24, 27].

Таблиця 3.7

Приклад операції ділення з відновленням остачі

№	Операція	Коментар
1	0. 11001100 +1. 01000000 0. 00001100 перша остача	Починаємо віднімати дільник з діленого. Якщо остача виходить додатною, то в розряд частки записуємо 1, в іншому випадку – 0. Перший (старший) біт частки дорівнює 1, оскільки остача вийшла додатною: C = 1XXXX.
2	0. 00011000 +1. 01000000 1. 01011000 друга остача	Першу остачу зсуваємо на один розряд вліво і з неї віднімаємо дільник. Остача від'ємна, тому в наступний розряд частки записуємо 0, C=10XXX.
3	1. 01011000 + 0. 11000000 0. 00011000 зсунута перша остача	Біти дільника повертаємо першій остачі, тобто складаємо дільник (у прямому коді) і другу остачу.
4	0. 00110000 + 1. 01000000 1. 01110000 третя остача	Зсуваємо зсунуту першу остачу на один розряд вліво й віднімаємо з неї дільник. Третя остача від'ємна, отже, наступний (третій) розряд частки дорівнює 0, C = 100XX.
5	1. 01110000 + 0. 11000000 0.00110000 двічі зсунута перша остача	Повертаємо дільник третій остачі.
6	0. 01100000 1. 01000000 1.10100000 четверта остача	Зсуваємо двічі зсунуту першу остачу на один розряд вліво й віднімаємо дільник. Четверта остача знову від'ємна, тому C=1000X.
7	0.1100000 + 1.0100000 0.0000000 п'ята остача	Додаємо дільник до четвертої остачі, результат зсуваємо на один розряд вліво, а потім знову віднімаємо дільник. Остача додатна, отже C = 10001 = $17_{(10)}$.

Наприклад, розділимо $A=35_{10}= 0.100011_2$ на $B=5_{10}=101_2$ ($5_д = 1.011_д$). У регістрі С, як і в попередньому прикладі, формується частка.

0.100011	– ділене
+ 1.011000	– перше віднімання дільника
1.111011	– С = 0 відновлюємо остачу до діленого
+ 0.101000	
0.100011	– зсуваємо вліво остачу
1.00011	
+ 1.01100	
0.01111	– С = 01, зсуваємо вліво остачу
0.1111	
+ 1.0110	
0.0101	– С = 011, зсуваємо остачу
0.101	
+ 1.011	
0.000	**Отримуємо відповідь** – С = 0111 = 7_{10}.

Продемонструємо цей метод на тому ж прикладі, але спочатку дільник В=(1100) представимо в додатковому коді, що дозволить обмежитися додаванням в усіх випадках, коли треба виконувати додавання або віднімання: $B=1100_{пр}=1.0100_д$. Частка формується в деякому регістрі С, незаповнені розряди якого будемо позначати через Х.

Ділення чисел, представлених у формі з фіксованою комою, можна також здійснити на двійкових суматорах оберненого й додаткового коду. Перед виконанням самої процедури ділення чисел у форматі з фіксованою комою визначається й запам'ятовується знак частки. Далі обидва операнди представляються в прямому коді, а дільник ще й у додатковому для того, щоб віднімання дільника замінити додаванням, і виконується сама процедура ділення за описаним вище методом з обов'язковим контролем переповнення розрядної сітки. Якщо знак частки від'ємний, то відповідь, за необхідності, представляється в додатковому коді.

Розглянемо ще один приклад. Розділимо $A=506_{10}=0.111111010$ на $B=23_{10}=0.10111$.

Після кожного віднімання перевіряємо знак результату. При додатному знаку в молодший розряд частки записуємо 1 і переходимо до наступного кроку віднімання. При від'ємному знаку в молодший розряд частки записуємо 0 і відновлюємо остачу додаванням до неї дільника. Після цього виконуємо чергове віднімання. Кожне чергове віднімання виконується після зсуву дільника на один розряд вправо або діленого – на один розряд вліво. Після обробки всіх розрядів діленого останній результат віднімання становить остачу від ділення. У розглянутих прикладах остача

дорівнює нулю. Якщо число не ділиться без остачі, то віднімання здійснюють до отримання потрібного числа розрядів частки після коми.

```
  0.111111010          – ділене
+ 1.01001              – перше віднімання дільника
 10.010001           1 – результат додатний
+ 1.01001              – друге віднімання дільника
  1.11010            0 – результат від'ємний
+ 0.10111              – додавання дільника
 10.100010             – відновлена остача
+ 1.01001              – третє віднімання дільника
 10.010111           1 – результат додатний
+ 1.01001              – четверте віднімання дільника
 10.000000           1 – остача дорівнює нулю.
```

Розглянутий алгоритм ділення з відновленням остачі (додаванням до від'ємної остачі діленого) передбачає різну послідовність операцій для додатної й від'ємної остач. Розглянемо на одному з прикладів ще один метод, використовуваний у цифрових автоматах для операції ділення двійкових чисел з фіксованою комою – *метод ділення без відновлення остачі*. Як уже було відзначено, основою виконання ділення є операція віднімання з метою отримання остачі, знак якої визначає цифру частки. У цьому випадку знак остачі визначає не лише чергову цифру частки, але й характер наступної процедури: додавання дільника до зсунутої остачі, якщо ця остача менша 0, і віднімання дільника із зсунутої остачі, якщо остача більша або дорівнює 0. Цей метод ділення отримав назву *ділення без відновлення остачі.*

Розглянемо приклад. Розділимо $A=506_{10}=0.111111010$ на $B=23_{10}=0.10111$.

```
  0.111111010          – ділене додатне
+ 1.01001              – перше віднімання дільника
 10.010001           1 – результат додатний
+ 1.01001              – друге віднімання дільника
  1.110100           0 – результат від'ємний
+ 0.10111              – додавання дільника
 100.010111          1 – результат додатний
+ 1.01001              – третє віднімання дільника
 10.000000           1 – остача дорівнює нулю.
```

3.3.2. Ділення чисел, представлених у формі з плаваючою комою

При операції ділення чисел, представлених у формі з плаваючою комою, їх мантиси діляться, як числа з фіксованою комою, а порядки віднімаються. В обох випадках перевірка на переповнення є обов'язковою. Але оскільки реально віднімаються характеристики чисел, то ділення виконують за такою формулою [5, 10, 15]:

$$A_1 / A_2 = m_1 \cdot 2^{p1} / m_2 \cdot 2^{p2} = (m_1 / m_2) \cdot 2^{p_1 - p_2 + l},$$

де $p_1 - p_2 + l$ – характеристика результату.

Знак відповіді визначають звичайним способом. Якщо відповідь вийшла ненормалізованою, то виконують процедуру нормалізації й округлення відповіді.

Оскільки мантиси операндів нормалізовані, то можливі випадки, коли $|m_1| > |m_2|$, $|m_1| < |m_2|$. У першому випадку перед початком ділення треба відняти дільник з діленого й у цілу частину мантиси частки записати 1. Потім продовжувати ділити числа звичайним методом. Після отримання частки, очевидно, що вона не буде нормалізованою. Отже, треба нормалізувати частку, тобто в цьому випадку зсунути її на один розряд вправо, а до порядку частки додати 1.

Нагадаємо, що при реалізації алгоритмів математичних операцій у форматі з плаваючою комою щоразу, коли відбувається та чи інша процедура, яка зачіпає характеристики операндів або результату, здійснюється контроль над переповненням і зникненням порядку.

Розглянемо приклад. A=10_{10}=0.1010, $p_1=p_a=4$, B=2=0.1, $p_2=p_в=2$, $|m_1| > |m_2|$.

При першому відніманні m_2 з m_1 1 записуємо в цілу частину частки:

0.1010	
+ 1.1000	
0.0010	C=1.XX, далі будемо ділити методом ділення без
0.0100	відновлення остачі
+ 1.1000	
1.1100	C = 1.0
1.1000	
+ 0.1000	
0.0000	C = 1.01. **Зсуваємо вправо** C = 0.101, p=p+1=3.

На завершення огляду принципів організації арифметичних дій з двійковими числами зазначимо, що далі ми детально розглянемо алгоритми арифметичних процедур з числами у формах з фіксованою й плаваючою комою.

3.4. Виконання операцій над десятковими числами

3.4.1. Представлення десяткових чисел у Д-кодах

Операції над десятковими числами (десяткова арифметика) часто включено до складу основних команд інформаційних систем і калькуляторів. Тому розробникові сучасного програмного забезпечення необхідно знати алгоритми виконання арифметичних операцій у двійково-десяткових кодах.

Двійково-десяткова система числення набула значного поширення в сучасних інформаційних системах через легкість переведення в десяткову систему й назад. Її використовують там, де основну увагу приділено не простоті технічного рішення архітектури інформаційних систем, а зручності їх роботи. Двійково-десяткова система числення неекономічна з огляду реалізації технічної побудови машини (приблизно на 20% збільшується потрібне обладнання), але дуже зручна для програмування.

Двійково-десятковий код (Д-код) десяткового числа – це таке його представлення, у якому кожну десяткову цифру зображено чотирма двійковими розрядами (*тетрадою* з двійкових символів). Для представлення одної десяткової цифри використовують чотири двійкові. Тут, звичайно, є надмірність, оскільки чотири двійкові цифри (*подвійна тетрада*) можуть зобразити не 10, а 16 чисел, однак ця система, як ми вже відзначали, дуже приваблива для програмістів.

Для однозначності переведення чисел у *Д*-код і назад бажано, щоб розряди тетрад мали певну вагу. Тоді значення десяткової цифри a_i відповідає виразу:

$$a_i = a_4 d_4 + a_3 d_3 + a_2 d_2 + a_1 d_1,$$

де d_i – вага розряду тетради.

Максимально допустиме число в тетраді – 9. Якщо виникає число 10 і більше, то одиниця переходить у наступну старшу тетраду. Існують різні *Д*-коди. Як приклад розглянемо *Д*-код, вага розрядів тетради якого така: 8, 4, 2, 1 (див. табл. 3.8). Цей код називається також кодом *Д1* прямого заміщення. Але є *Д*-коди з тетрадами: 5, 1, 2, 1; 2, 4, 2, 1 і т. ін. Указані комбінації в цих *Д*-кодах дозволені. Усі інші комбінації – заборонені.

Наявність дозволених і заборонених комбінацій – дуже важлива властивість *Д*-кодів. Вона відрізняє їх від звичайних позиційних систем числення, у яких усі комбінації – дозволені.

В аналізованому *Д*-коді прямого заміщення (8421) дозволені комбінації відповідають двійковим еквівалентам десяткових цифр з вагами розрядів, рівних ступеням основи 2. Цей *Д*-код найчастіше використовують у різноманітних інформаційних системах, електронних касових апаратах та інших цифрових пристроях.

Таблиця 3.8

Приклади запису десяткових чисел у коді *Д1* прямого заміщення

Десяткові цифри	Код *Д1*	Десяткові цифри	Код *Д1*	Десяткові цифри	Код *Д1*
0	0000	10	0001 0000	20	0010 0000
1	0001	11	0001 0001	21	0010 0001
2	0010	12	0001 0010	22	0010 0010
3	0011	13	0001 0011	33	0011 0011
4	0100	14	0001 0100	34	0011 0100
5	0101	15	0001 0101	45	0100 0101
6	0110	16	0001 0110	56	0101 0110
7	0111	17	0001 0111	67	0110 0111
8	1000	18	0001 1000	78	0111 1000
9	1001	19	0001 1001	89	1000 1001

3.4.2. Формальні правила порозрядного додавання в Д-кодах

Спочатку розглянемо особливості, властиві *Д*-кодам.

Поява забороненої комбінації під час виконання якихось дій над числами свідчить про виникнення помилки або ж про необхідність ввести корекцію результату.

При додаванні тетрад виникає потетрадний перенос, тобто перенос 1 в старшую тетраду, замість десяткового порозрядного переносу.

При додаванні чисел у *Д*-коді можуть виникнути такі випадки:

1) при додаванні двох тетрад у цьому розряді числа утворюється сума менша 10, значить, не потрібна корекція результату. Наприклад, скласти дві тетради A = 0100, B = 0101 за умови, що немає переносу з молодшої тетради. C = 0100 + 0101 = 1001. Результат менший 10.

2) свідченням того, що результат додавання неправильний, є або поява забороненої комбінації, якщо $15 \geq C \geq 10$, або поява потетрадного

переносу p=16, що перевищує значення десяткового переносу на 6. Отже, потрібна корекція результату в цій тетраді введенням поправки, що дорівнює + 0110 = $6_{(10)}$. Наприклад, скласти дві тетради A= 0101, B = 1001 за наявності переносу з молодшої тетради – p = *1*/C=A+B+p, р = 1111 – заборонена комбінація й потрібна поправка. Тоді,

$$\begin{array}{r} 1111 \\ \underline{+\ 0110} \\ [1] \leftarrow 0101 \end{array}$$

тобто результат у цій тетраді дорівнює 0101 і необхідний перенос у старшу тетраду.

Розглянемо ще один приклад. Складемо $A = 279_{10}$ = 0010 0111 1001, B=581_{10} = 0101 1000 0001.

$$\begin{array}{rccc} & 0010 & 0111 & 1001 \\ + & 0101 & 1000 & 0001 \\ & 0111 & 1111 & 1010 \\ + & & \underline{0110} & \underline{0110} \quad \text{поправки} \\ C = 1000 & \leftarrow 0110 & \leftarrow 0000 & \end{array}$$

Стрілка указує передачу 1 десяткового переносу. У підсумку отримаємо C=100001100000=860_{10}.

Суматор одної тетради, призначений для формування суматорів чисел представлених у *Д*-коді прямого заміщення, показано на рис. 3.3. На рисунку символи s – сума в цій тетраді a_i+b_i і p – перенос у старшу тетраду.

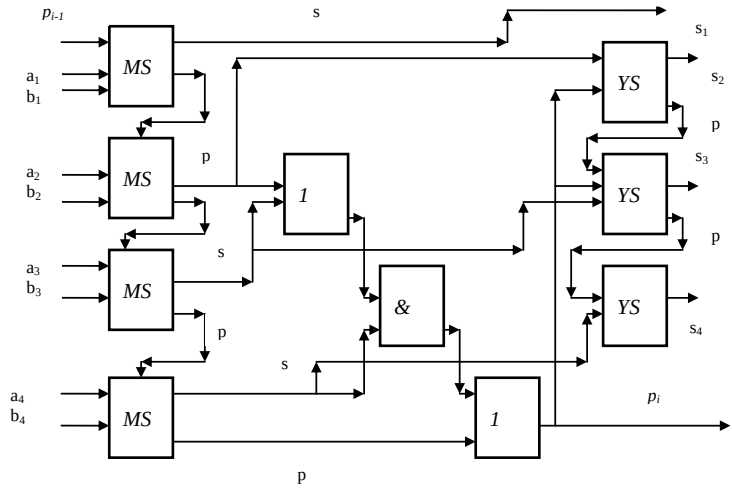


Рис. 3.3. Схема суматора тетради

3.4.3. Представлення від'ємних чисел у Д-кодах

Представлення *Д*-коду в розрядній сітці може бути здійснено у формі з фіксованою або плаваючою комою. При цьому від'ємні числа можуть бути представлені в прямому, оберненому або додатковому коді.

Для аналізованого *Д1*-коду не можна отримати обернений або додатковий код простим інвертуванням, оскільки інвертування набору тетрад означає отримання доповнення до 2^4-1=15. Отже, необхідно прибрати різницю. Один з використовуваних при цьому прийомів полягає в тому, що в усі цифрові тетради числа в *Д1*-коді додається 0110. Після цього здійснюється інвертування набору. Отримане зображення становить собою обернений код числа. Додатковий код отримуємо, як завжди, додаванням 1 до молодшого розряду молодшої тетради. Розглянемо приклади. Представимо число $A=-256_{10}$ = 0. 0010 0101 0110 в оберненому коді для коду *Д1*:

	1. 0010 0101 0110	
+	0110 0110 0110	у всі тетради додали 0110
	1000 1011 1100	після інвертування маємо

Отримаємо $A_{об}$ = 1. 0111 0100 0011.

Представити число $A = -398_{10}$ в додатковому коді для коду *Д1*:

	1. 0011 1001 1000	
	+ 0110 0110 0110	додаємо 0110
	1. 1001 1111 1110	інвертуємо
	0110 0000 0001	
	+1	
Отримаємо $A_д$ =	1.0110 0000 0010	

3.4.4. Виконання операцій додавання й віднімання чисел у Д-кодах

Операції додавання й віднімання в *Д*-кодах виконують над операндами за формальними правилами десяткової арифметики, описаними раніше.

При додаванні двох додатних чисел переведення в додатковий код виключено. У випадку появи потетрадного переносу або результату більше 9 здійснюють корекцію результату додаванням 6.

Додавання від'ємних чисел виконують аналогічно додаванню додатних з тією лише різницею, що результату присвоюють від'ємний знак.

При відніманні чисел здійснюють попереднє переведення в додатковий код, а потім додавання чисел.

Розглянемо приклад, коли одне число від'ємне, а друге додатне.

Складемо два числа в коді *Д1* А = -1000 0010 0101, В = 1001 0100 0110.

1. 0001 0111 0101
0. 1001 0100 0110
1. 1010 1011 1011
\+ 0110 0110 0110 поправки

Отримаємо С =0.←0001 ← 0010 ← 0001.

3.4.5. Множення й ділення чисел у Д-кодах

Операці множення у *Д*-кодах виконують за звичайною схемою. Множення чисел зводиться до послідовного додавання часткових добутків, отриманих під час множення множеного на чергову цифру множника. Оскільки кожна цифра множника представляється тетрадою, то множення супроводжується розшифровкою значення чергової тетради множника й зсувом на чотири розряди одразу. Розшифровку можна здійснити різними способами [2,24,35]. Найпростішим прикладом є послідовне віднімання 1 із значення тетради до отримання 0 і, відповідно, додавання множеного. Обов'язково враховуються проміжні переповнення.

Розглянемо приклад. Помножимо $A=25_{10}=00100101$ на $B=12_{10}=00010010$, часткові добутки формуємо в *Р*. Аналіз тетрад В починаємо з молодшої (0010). А·В= 0010 0101 · 0001 0010 = 0011 0000 0000 = 300_{10}.

P 0000 0000 0000
\+ A 0010 0101 0010 - 0001 = 0001 > 0, отже, треба ще раз В + Р
P 0000 0010 0101
\+ A 0010 0101 0010 - 0001 = 0, кінець аналізу молодшої тетради.
P 0000 0100 1010
\+ 0110 поправка
P 0000 0101 0000 зсуваємо В на чотири розряди (1 тетраду) вліво і складаємо з Р, аналізуючи старшу тетраду В.
P 0000 0101 0000
\+ A 0010 0101 0000 0001 - 0001 = 0
P 0010 1010 0000
\+ 0110 поправка
P 0011 0000 0000 = 300_{10} **Отримаємо** А·В = 0011 0000 0000 = 300_{10}.

Ділення десяткових чисел у *Д*-кодах виконують методом послідовного віднімання дільника з діленого на першому кроці та з остач – на наступних кроках. Віднімання на кожному кроці здійснюють до тих пір, поки не вийде від'ємна остача. Кожного разу при отриманні додатної остачі додається 1 в спеціальний лічильник, де накопичується чергова цифра частки. Потім здійснюється зсув на чотири двійкових розряди й додавання дільника до тих пір, поки не вийде додатна остача. Кількість додавань (без останнього) є доповненням відповідної цифри частки до 9, що заноситься в лічильник чергової цифри частки. Таким чином, процес ділення складається з послідовного чергування циклів додавання й віднімання із зсувами. Знак частки отримуємо як логічну суму за модулем 2 знаків чисел.

Наприклад, $A=48_{10}=01001000$, $B=2_{10}=00000010$, $A/B=24_{10}=00100100$. В С1 – формуємо старшу тетраду частки, а в С2 – молодшу.

A/B= 0100 1000 / 0010
- 0010
0010 > 0 C1= C1+1=1
- 0010
0000 C1=1+1=2= 0010
0010
-0010
+ 0010
0000 зсуваємо В на 4 розряди вправо й виконуємо ті самі дії:
0100 1000
- 0010
0110 > 0 C2=C2+1=1
- 0010
0100 > 0 C2=1+1=2
- 0010
0010 > 0 C2=2+1=3
- 0010
0000 C2=3+1=4= 0100

Отримаємо в підсумку C1+C2 = 0010 0000 + 0000 0100= 0010 0100 = 24_{10} .

3.4.6. Переведення чисел з Д-коду в двійковий і з двійкового коду в Д-код

Перетворення з двійково-десяткової системи в десяткову (і зворотні перетворення) не викликають труднощів і виконуються шляхом прямої заміни чотирьох двійкових цифр одною десятковою цифрою (або

навпаки). Дві двійково-десяткові цифри становлять один байт. Таким чином, за допомогою одного байта можна представити значення від 0 до 99, а не від 0 до 255 або від 0 до FF, як при використанні 8-розрядного двійкового слова або 2-розрядного шістнадцяткового числа. Використовуючи один байт, можна формувати двійково-десяткові числа з будь-яким потрібним числом десяткових розрядів. Наприклад, якщо число A=1001010100111000 розглядати як двійкове, то його десятковий еквівалент $(1001010100111000)_2=(38200)_{10}$ в чотири рази більший, ніж десятковий еквівалент двійково-десяткового числа $(1001010100111000)_{2\text{-}10}=(9538)_{10}$.

У загальному випадку, нехай задано 4-розрядне число в *Д1* коді – $A=a_4\,a_3\,a_2\,a_1$, кожна десяткова цифра якого має бути представлена у вигляді $a_i=\{4\ 3\ 2\ 1\}_i$. Число *A* можна представити в такому вигляді [15,23,34]:

$$A = a_4 \cdot 10^3 + a_3 \cdot 10^2 + a_2 \cdot 10^1 + a_1 \cdot 10^0.$$

Як видно з формули, при переведенні з коду *Д1* у двійковий код кожний *i*-й розряд коду *Д1* треба множити на 10^{i-1}, тобто $a_1 \cdot 1, a_2 \cdot 10, a_3 \cdot 100, a_4 \cdot 1000.$

Водночас $10 = 8 + 2 = 2^3+2^1$, отже, маємо: $a_1 = \{a_4\ a_3\ a_2\ a_1\}_1$; $a_2 = \{a_4\ a_3\ a_2\ a_1\}_2 \cdot (2^3 + 2^1)$; $a_3 = \{a_4\ a_3\ a_2\ a_1\}_3 \cdot (2^3 + 2^1)\cdot (2^3 + 2^1)$; $a_4 = \{a_4\ a_3\ a_2\ a_1\}_4 \cdot (2^3 + 2^1)\cdot (2^3 + 2^1)\cdot (2^3 + 2^1)$.

Таким чином, перша тетрада не множиться. Друга тетрада зсувається на три розряди вліво й зберігається, потім ця сама тетрада зсувається на один розряд вліво, додається до збереженої тетради й знову зберігається. Далі з третьою тетрадою виконуються ті самі процедури, але послідовно двічі, а з четвертою – тричі. Усі отримані результати потетрадно складаються. У результаті отримуємо число у двійковому коді.

Розглянемо приклад. Переведемо $A=25_{10} = 0010\ 0101_{2\text{-}10}$ у двійкову систему числення. Другу тетраду (0010) зсуваємо на три розряди вліво, отримуємо 0001 0000. Цю саму тетраду зсуваємо на один розряд вліво й складаємо з отриманою:

```
 0001 0000
+0000 0100
 0001 0100
+0000 0101                  (перша тетрада)
 0001 1001 = 25₁₀ .
```

Переведення з двійкової системи в *Д1*-код може здійснюватися різними способами. Зокрема, для ряду послідовних операцій над двійковим зображенням числа може бути використана процедура ділення на $1010_2=10_{10}$ цілих двійкових чисел. Десяткові цифри отримуємо послідовно

одну за одною починаючи зі старшого десяткового розряду. При дробових числах ця операція видозмінюється так, щоб при множенні на число 1010 можна було отримати відповідні цифри десяткових дробів.

Є більш простий спосіб переведення – це зсув вліво двійкового числа стільки разів, скільки розрядів у двійковому числі. Необхідно передбачити корекцію в тих тетрадах, значення яких перевищить 1010, або відбудеться потетрадний перенос. Наприклад, $A=49_{10}=110001_2$ переведемо в код *Д1*.

									110001
1) Зсув							1		10001
2) Зсув						1	1		0001
3) Зсув					1	1	0		001
4) Зсув і корекція				1	1	0	0		01
+0110				0	1	1	0		
Результат після корекції			1	0	0	1	0		01
5) Зсув		1	0	0	1	0	0		1
6) Зсув	1	0	0	1	0	0	1		
Відповідь	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	

Алгоритми переведення чисел з двійкової системи числення в *Д*-код і навпаки можуть бути реалізовані схемними або програмними способами [5, 7, 8, 10, 15 та ін.].

3.5. Оцінка точності виконання арифметичних операцій

Як уже було відзначено, для представлення чисел у будь-якій формі в цифрових автоматах відведено скінчену довжину розрядної сітки. Тому числа, у загальному випадку, через неминучі процедури округлення, представлено з деякою похибкою. У зв'язку з цим обчислення в цифровому автоматі виконуються з похибкою, яка в деяких випадках може суттєво накопичуватися й, отже, впливати на точність результату.

Наприклад, нехай величини A і B задані з абсолютними похибками. [A] і [B] – машинне представлення чисел A і B.

Можна показати, що відносна похибка алгебраїчної суми дорівнюватиме [9]:

$$\delta_{A\pm B} = \frac{[A]}{[A]\pm[B]}\delta A \pm \frac{[B]}{[A]\pm[B]}\delta B .$$

Відносна похибка добутку [14]: $AB = |A| + |B|$, а відносна похибка частки: $A/B = |A| + |B|$.

Як видно з наведених виразів, операції множення й ділення незначно збільшують відносну похибку, а віднімання майже рівних чисел може відчутно її збільшити. Дійсно, якщо прийняти, що [A] мало відрізняється від [B], то справедливим є співвідношення

$$\delta_{A-B} \approx \delta \cdot \frac{2[A]}{[A]-[B]}.$$

Через неминучі похибки машинних обчислень може спостерігатися порушення деяких основних законів математики. Наприклад, може не виконуватися точно закон асоціативності для множення. Закон дистрибутивності, що пов'язує операції множення й додавання, може значно порушуватися, тобто не буде суворо виконуватися рівність $A \cdot (B+C)=(A \cdot B)+(A \cdot C)$.

Тому треба дуже ретельно оцінювати конкретні умови обчислень, наприклад, з великими масивами, і за необхідності переходити до представлення чисел з подвійною точністю. Слід уживати інших заходів, щоб включити вплив накопиченої похибки на результат обчислень.

Оскільки в інформаційних системах доводиться вдаватися до округлення чисел, то неминучими є похибки округлення.

Залежно від того, як ураховано величину частини числа, яка не потрапила в розрядну сітку, існує кілька способів округлення.

Перший спосіб. Відкидання частини числа, яка не потрапила в розрядну сітку. У цьому випадку похибка округлення не залежить від величини самого числа, а залежить лише від кількості розрядів у машині для будь-якої системи числення.

Другий спосіб – симетричне округлення. При цьому аналізують величину частини числа, яка не потрапила в розрядну сітку. У цьому випадку помилка не перевищує половини одиниці молодшого розряду. Спосіб симетричного округлення найчастіше застосовують на практиці.

Третій спосіб – округлення за доповненням. У цьому випадку для округлення береться інформація, яка міститься в (n+1)-му розряді. При q = 2, якщо в $(n+1)$ розряді міститься 1, у n-й розряд додається 1; якщо ж там нуль, вміст розрядів правіше n-го відкидається.

Запитання для самоперевірки

1. Перемножити числа A=-0.11010101 і B=0.00101001.
2. Як здійснюються операції додавання й віднімання з двійковими числами?

3. Як здійснюються операції множення й ділення з двійковими числами?
4. Як здійснюються операції множення й ділення з двійковими числами, представленими у формі з плаваючою комою?
5. Якими показниками оцінюють точність виконання операцій у цифрових автоматах?
6. Дайте визначення Д-кодів.
7. Чому Д-коди широко використовують у інформаційних системах?
8. Переведіть число 32 в Д-код.
9. Як здійснюють операції додавання й віднімання в Д-кодах?
10. Як здійснюють операції множення й ділення в Д-кодах?
11. Наведіть способи округлення чисел у цифрових автоматах.
12. Поясніть метод ділення двійкових чисел з відновленням остачі.
13. Що таке модифіковане представлення чисел?
14. У яких випадках настає переповнення розрядної сітки при додаванні двійкових чисел?
15. Нарисуйте схему комбінаційних суматорів.
16. Переведіть числа $A=12_{10}$ і $B=18_{10}$ у Д-коди й виконайте операцію додавання.
17. Переведіть числа $A=32_{10}$ і $B=8_{10}$ у Д-коди й виконайте операцію ділення.

Література для самостійної підготовки за темою:

8, 14, 15, 23, 24, 25, 27, 33, 35.

Розділ 4. КОНТРОЛЬ РОБОТИ ЦИФРОВОГО АВТОМАТА

Розглянуті в попередньому розділі алгоритми виконання арифметичних операцій забезпечують отримання правильного результату лише в тому випадку, якщо цифровий автомат (ЦА) працює без порушень. При виникненні порушень у його роботі результат буде неправильним, однак користувач про це не дізнається, якщо не будуть передбачені заходи, які сигналізують про появу помилки.

Отже, при синтезі цифрового автомата мають бути передбачені заходи щодо виявлення і, якщо потрібно, то й виправлення помилки. Ці функції зазвичай покладено на систему контролю роботи цифрового автомата.

Система контролю – це сукупність методів і засобів, які забезпечують визначення правильності роботи автомата в цілому або окремих його вузлів, а також автоматичне виправлення помилки.

Тому система контролю повинна будуватися з таким розрахунком, щоб вона дозволяла виявити й по можливості виправити будь-які порушення.

Помилки результату можуть бути таких видів:

1) помилки, які виникають через похибки у вихідних даних;

2) помилки, обумовлені методичними похибками;

3) помилки, які з’являються через виникнення несправностей у роботі машини.

Перші два види не є предметом аналізу в цьому посібнику. Тому розглянемо третій вид помилок. Перевірка правильності функціонування окремих пристроїв цифрового автомата й виявлення несправностей може бути здійснена за двома напрямами:

1) профілактичний контроль, завдання якого – попередження появи можливих помилок у роботі;

2) оперативний контроль, завдання якого – контроль за помилками у роботі.

Вирішення всіх завдань контролю стає можливим лише за наявності певної надмірності інформації, яка супроводжує основну інформацію.

Надмірність може бути створена або апаратними (схемними) засобами (наприклад, 100% резервування), або логічними чи інформаційними засобами.

Відсутність логічного контролю в перших цифрових автоматах породило свого часу метод *„подвійнго рахунку”* – тобто задачу розв’язували двічі й результати порівнювали [6,7, 14, 15, 24, 27 та ін.].

Розглянемо, як удасться вирішити цю проблему за рахунок спеціального кодування станів цифрового автомата таким чином, щоб забезпечувався оперативний контроль правильності перетворення даних і виявлення місця помилки.

Оскільки більшість зовнішніх пристроїв цифрових автоматів, зокрема інформаційних систем, дозволяють людині спілкуватися з ними звичною для неї мовою слів і десяткових чисел, а інформаційна система розуміє лише мову фізичних станів, кодованих двійковими числами, то в обчислювальних пристроях (ОП) зазвичай здійснюється кодування (декодування) інформації, що пересилається в (з) обчислювальний пристрій. Слід також відзначити, що оскільки цифрові автомати мають достатньо багато механічних елементів, то вони менш надійні, ніж електронні. Причиною неправильного сприйняття коду символу можуть бути забруднення на зчитувальних головках накопичувачів на магнітних дисках, плями на лазерних дисках і т. ін. Для виявлення таких помилок (а іноді й для їх виправлення) коди символів перетворюють у відповідну стандартну форму, наприклад, додають ще один байт, щоб у новому коді було парне число одиниць. При передачі такого стандартного коду здійснюється певного виду контроль (наприклад, контроль парності), і за його результатам приймається рішення про використання отриманого символу.

У загальній постановці задача кодування інформації представляється як відповідне перетворення числових даних у заданій системі числення. При цьому вибирається така система числення, за допомогою якої можна створити надмірність при передачі інформації і за рахунок цього виявляти помилки.

Позиційні системи числення не мають надмірності, тому не придатні для цієї мети.

Ідеться про використання *систематичних кодів* – кодів, які містять у собі, крім інформаційних, контрольні розряди.

У контрольні розряди записується деяка інформація про вихідне число. Тому повний код числа складатиметься із самого числа плюс деяка його характеристика (наприклад, парне це число чи непарне).

4.1. Кодування за методом парності-непарності

Якщо в математичному коді виділено один контрольний розряд (k=1), то до кожного двійкового числа додається один надмірний розряд і в

нього записується 1 або 0 з такою умовою, щоб сума цифр у кожному числі за „mod 2” дорівнювала 0 для випадку парності або 1 для випадку непарності. Поява помилки в кодуванні виявиться за порушенням парності (непарності). При такому кодуванні допускається, що може виникнути лише одна помилка. Приклад реалізації методу парності представлено в табл. 4.1.

Таблиця 4.1

Приклад реалізації методу парності-непарності

Передача		**Прийом**
Число	Контрольний розряд	Перевірка
10101011	1	0
11001010	0	0
10010001	1	0
11001011	0	1 – порушення

Можна представити й дещо видозмінений спосіб контролю за методом парності-непарності. Довге слово розбивається на групи, кожна з яких містить n розрядів. Контрольні розряди k виділяються всім групам за рядками й за стовпцями відповідно до схеми (див. рис. 4.1) [15, 24]:

a_1	a_2	a_3	a_4	a_5	k_1
a_6	a_7	a_8	a_9	a_{10}	k_2
a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	k_3
a_{16}	a_{17}	a_{18}	a_{19}	a_{20}	k_4
a_{21}	a_{22}	a_{23}	a_{24}	a_{25}	k_5
k_6	k_7	k_8	k_9	k_{10}	

Рис. 4.1. Схема виділення контрольних розрядів

Збільшення надмірності інформації приводить до того, що з’являється можливість не лише виявити помилку, але й виправити її.

Контроль за методом парності-непарності широко використовують в ЕОМ для контролю запису, зчитування інформації в ЗП, а також під час виконання арифметичних операцій.

4.2. Коди Хеммінга

Коди, запропоновані американським ученим Р. Хеммінгом у 1948 р., дозволяють не лише виявити, але й виправити поодинокі помилки. Ці коди – систематичні. Хеммінг уперше ввів поняття кодової відстані [6, 8, 34].

Кодовою відстанню між двома словами називається кількість розрядів, у яких символи слів не збігаються.

Мінімальною кодовою відстанню (d_{min}) цього коду називається мінімальна відстань між двома будь-якими словами в цьому коді. Якщо довжина слова *n*, то кодова відстань може приймати значення від 1 до *n*. Якщо є хоч одна пара слів, що відрізняються в одному розряді, то мінімальна кодова відстань дорівнює 1. Для систематичних кодів $d_{min}>1$. У загальному випадку, щоб код дозволяв виявляти помилки кратністю *r*, має бути виконано умову: $d_{min} \geq r + 1$.

Код Хеммінга будується так, що до наявних інформаційних розрядів слова додається певна кількість контрольних розрядів, які формуються перед записом слова й разом з інформаційними розрядами слова записуються в пам'ять ЦА. При зчитуванні слова контрольна апаратура утворює з прочитаних інформаційних і контрольних розрядів коригувальне число, яке дорівнює „0" за відсутності помилки, або вказує місце помилки. Помилковий розряд автоматично коригується зміною його стану на протилежний. Розглянемо приклад. Нехай $d_{min}=3$. Якщо в молодшому розряді коригувального числа з'явиться „1", то це означає помилку в одному з тих розрядів слова, порядкові номери яких мають „1" в молодшому розряді (тобто розрядів з непарними номерами). Введемо перший контрольний розряд, якому присвоїмо непарний порядковий номер і який установимо при кодуванні таким чином, щоб сума одиниць усіх розрядів з непарними порядковими номерами дорівнювала „0". Ця операція може бути записана у вигляді:

$$K_1 = x1+x3+x5+ \ldots =0,$$

де *x1, x3, x5* – двійкові символи, розміщені в розрядах з порядковими номерами 1, 3,.. і т. д.

Поява „1" в другому розряді (справа) коригувального числа означає помилку в одному з тих розрядів слова, порядкові номери яких (2,3,6,7,10...) мають „1" в другому справа розряді. Тому друга операція кодування, яка дозволяє знайти другий контрольний розряд, має вигляд:

$$K_2 = x2+x3+x6+x7+\ldots=0.$$

Розмірковуючи аналогічно, можна визначити всі інші контрольні розряди, тобто знайти K_3, K_4,...

Після прийому кодового слова (разом зі сформованими контрольними розрядами) виконуються ті самі операції підрахунку, які було описано вище, і число $K_kK_m - \ldots K_3K_2K_1$, що утворюється, вважається коригувальним.

За відсутності помилок K_k ...K_2K_1=0, за наявності помилки не дорівнюватимуть нулю ті суми K_i, в утворенні яких брав участь помилковий розряд; коригувальне число при цьому дорівнюватиме порядковому номеру помилкового розряду.

Вибір місця для контрольних розрядів здійснюється так, щоб контрольні розряди брали участь лише в одній операції підрахунку парності. У нашому прикладі такі позиції – це розряди з номерами, що є ступенями двійки: 1, 2, 4, 8, 16 і т. д.

Контроль за кодом Хеммінга реалізується за допомогою набору схем підрахунку парності, розглянутих вище, які при кодуванні визначають контрольні розряди, а при декодуванні формують коригувальне число.

Припустимо, до n_0 інформаційних розрядів у коді Хеммінга додається K контрольних розрядів для автоматичного визначення місця розташування помилкового розряду. Загальна кількість символів $n=n_0+k$. Здійснюється k перевірок на парність (за кількістю контрольних розрядів) і записується k-розрядне двійкове число, яке визначає номер позиції коду з помилкою. Якщо n – загальна кількість розрядів, то

$$n+1=n_0+k+1\le 2^k \text{ звідси } n_0\le 2^k-k-1.$$

Звідси випливає, що, наприклад, 5 контрольних розрядів дозволяють передавати 26 інформаційних розрядів.

Інформаційні й контрольні розряди необхідно розміщувати на певних місцях для знаходження помилок. Зазвичай контрольні розряди прийнято розміщати на позиціях коду 2^0, 2^1, 2^2, ... (1, 2, 4, 8, 16 ...), у яких у записі є лише одна 1 (для зручності). При цьому нумерація позицій іде зліва направо. А нумерація позицій для запису інформаційних розрядів – справа наліво. Розглянемо приклад переведення двійкового коду числа в код Хеммінга. Нехай треба передати інформацію, для якої виділено три контрольних розряди, тобто K = 3, тоді n_0= 4, тобто для представлення числа нам відводиться чотири розряди. У підсумку передається 7-розрядний код. Через I позначимо інформаційні розряди, а через K – контрольні.

1	2	3	4	5	6	7
K_1	K_2	I_4	K_3	I_3	I_2	I_1
0	0	0	0	1	1	0

тобто передається $0110_2 = 6_{10}$. Для перевірки на парність складаємо позиції 1, 3, 5, 7 (відлік іде зліва направо), тобто порядкові номери яких мають 1 в молодшому розряді (у двійковому представленні). Отримуємо: 0+0+1+0=1, сума дорівнює 1, значить у K_1 записуємо 1 (для парності). Далі аналогічно складаємо позиції 2, 3, 6, 7 (порядкові номери містять 1 в другому розряді), отримуємо: 0+0+1+0=1, значить у K_2 записуємо теж 1. Далі складаємо

позиції 4, 5, 6, 7 (порядкові номери містять 1 у третьому розряді), отримуємо: 0 + 1 + 1 + 0 = 0, в K_3 записується 0. Таким чином, отримано такий код:

1	2	3	4	5	6	7
K_1	K_2	I_4	K_3	I_3	I_2	I_1
1	1	0	0	1	1	0

Це і є 6 у коді Хеммінга. Слід відзначити, що загальна кількість 1 в коді має бути парною.

Тепер розглянемо приклад коригування отриманого кодованого в коді Хеммінга числа, у якому є збій: число 0111000. Треба визначити позицію, у якій відбувся збій. Для цього сумуємо позиції 1, 3, 5, 7, отримуємо: K_1 = 0+1+0+0=1. Далі сумуємо позиції 2, 3, 6, 7, отримуємо: K_2 =1+1+0+0=0. Сумуємо позиції 4, 5, 6, 7, отримаємо K_3=1+0+0+0=1. Контрольний код дорівнює $101_2 = 5_{10}$, значить помилка на 5-й позиції коду, тобто $0111100_2 = 12_{10}$.

Розглянемо ще один приклад. На рис. 4.2 показано код виправлення для 16-бітового слова даних. До цього слова додано ще п’ять бітів парності, які розташовані в 1, 2, 4, 8 і 16 розрядах слова, а в розрядах 3, 5 – 7, 9 – 15, 17 – 21 розміщено біти даних (див. рис. 4.2 а).

Кожен біт парності використовується для контролю лише певних розрядів розширеного слова. Номери контрольованих розрядів для кожного біта парності наведено в табл. 4.2. Нагадаємо, що в число контрольованих розрядів внесено й той розряд, де розташований сам біт парності. При цьому вміст біта парності встановлюється таким чином, щоб сумарне число одиниць у контрольованих ним розрядах було парним.

Таблиця 4.2

Номери бітів парності й контрольованих ними розрядів слів у коді Хеммінга

Контрольні розряди	**Контрольовані розряди**																	
1	**1**	**3**	**5**	**7**	**9**	**11**	**13**	**15**	**17**	**19**	**21**	**23**	**25**	**27**	**29**	**31**	**32**	**..**
2	**2**	**3**	**6**	**7**	**10**	**11**	**14**	**15**	**18**	**19**	**22**	**23**	**26**	**27**	**30**	**31**	**34**	**..**
4	**4**	**5**	**6**	**7**	**12**	**13**	**14**	**15**	**20**	**21**	**22**	**23**	**28**	**29**	**30**	**31**	**36**	**..**
8	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**40**	**..**
16	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**48**	**..**
32	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**	**..**
…	**..**	**..**	**..**	**..**	**..**	**..**	**..**	**..**	**..**	**..**	**..**	**..**	**..**	**..**	**..**	**..**	**..**	**..**

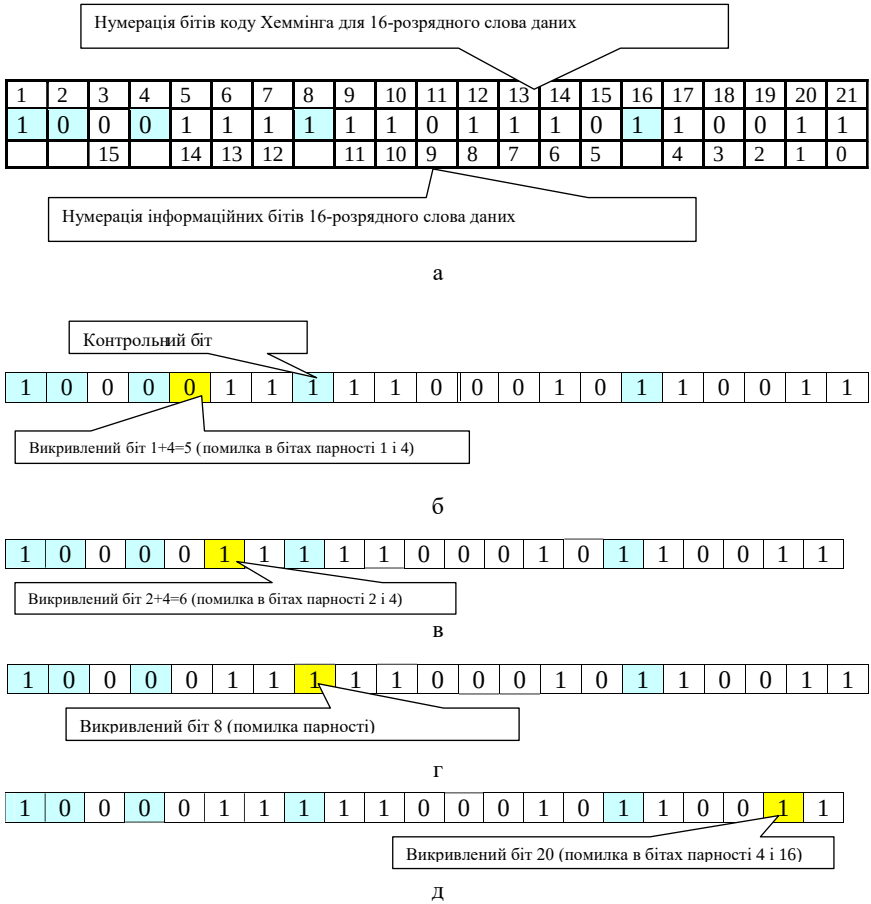


Рис. 4.2. Код виправлення для 16-бітного слова даних

Розглянемо приклад, показаний на рис. 4.2. Спочатку перевіримо код числа, наведеного на рис. 4.2 а. У ньому біти 1, 3, 5, 7, 9, 11, 13, 15, 17, 19 і 21 містять шість одиниць, тобто біт парності 1 – правильний. Біти 2, 3, 6, 7, 10, 11, 14, 15, 18 і 19 містять чотири одиниці, тобто біт парності 2 – правильний. Біти 4, 5, 6, 7, 12, 13, 14, 15, 20 і 21 містять шість одиниць, тобто біт парності 4 – правильний. Біти 8, 9, 10, 11, 12, 13, 14 і 15 містять чотири одиниці, тобто біт парності 8 – правильний. Біти 16, 17, 18, 19, 20 і 21 містять чотири одиниці, тобто біт парності 16 теж правильний. Проаналізуємо, що відбудеться, якщо через виникнення помилки зникне

одиниця в п’ятому розряді цього числа (див. рис. 4.2 б). Перевірка коду, що утворився, дає такий результат:

- біт парності 1 неправильний (біти 1, 3, 5, 7, 9, 11, 13, 15, 17, 19 і 21 містять п’ять одиниць);
- біт парності 2 правильний (біти 2, 3, 6, 7, 10, 11, 14, 15, 18 і 19 містять чотири одиниці);
- біт парності 4 неправильний (біти 4, 5, 6, 7, 12, 13, 14, 15, 20 і 21 містять п’ять одиниць);
- біт парності 8 правильний (біти 8, 9, 10, 11, 12, 13, 14 і 15 містять чотири одиниці);
- біт парності 16 правильний (біти 16, 17, 18, 19, 20 і 21 містять чотири одиниці).

Отримання неправильного значення біта парності 1 указує на те, що помилка має бути в одному з контрольованих ним бітів: 1, 3, 5, 7, 9, 11, 13, 15, 17, 19 або 21. Оскільки біт парності 2 правильний, то правильні й контрольовані ним непарні біти 3, 7, 11, 15 і 19, так що помилка відбулася не в них. Правильність контрольного біта 8 виключає виникнення помилки в бітах 9 і 13, а правильність контрольного біта 16 – у бітах 17 і 21. Таким чином, підозрілими залишаються біти 1 і 5. Оскільки неправильним є й біт парності 4, який не контролює біт 1, але контролює біт 5, можна зробити висновок про помилковість біта 5. Інвертування цього біта виправляє становище, тобто всі біти парності стають правильними.

Схожий аналіз можна застосувати й до кодів, показаних на рис. 4.2 в, г, д. Ці коди є викривленими значеннями коду на рис. 4.2 а. Алгоритм аналізу можна сформулювати так:

а) перевірити всі біти парності;

б) якщо всі біти парності правильні, то перейти до пункту д);

в) обчислити суму номерів усіх неправильних бітів парності;

г) інвертувати (змінити значення з 0 на 1) вміст біта, номер якого дорівнює сумі, обчисленій у пункті в);

д) передати правильний код числа, виключивши з нього всі біти парності.

Перевірочна матриця будь-якого коду Хемінга завжди містить мінімум три лінійно залежних стовпці, тому кодова відстань коду дорівнює трьом.

Якщо стовпці перевірочної матриці представляють упорядкований запис десяткових чисел, тобто 1, 2, 3... у двійковій формі, то обчислений синдром

$$S_i(1,0) = S_{r-1} \ldots S_1 S_0 = \nu_i(1,0) \cdot \mathrm{H}^{\mathrm{T}}_{(n,k)},$$

однозначно вказує на номер позиції перекрученого символу.

Для (7,4) - коду Хемінга перевірочна матриця в упорядкованому виді має вид

$$\mathrm{H}_{(7,4)} = \begin{vmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{vmatrix}.$$

Нехай передане кодове слово $\nu(1,0) = 1100001$, а прийняте слово - $\nu^1(0,1) = 1110001$.

Синдром, що відповідає прийнятому слову буде дорівнює

$$S_i(0,1) = \nu^1(0,1) \cdot \mathrm{H}^{\mathrm{T}}_{(7,4)} = [1110001] = \begin{vmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{vmatrix} = [101]$$

Обчислений синдром вказує на помилку в п'ятій позиції. Перевірочна матриця в упорядкованому виді представляє сукупність перевірочних рівнянь, у яких перевірочні символи займають позиції з номерами $2^i (i = 0, 1, 2\ldots).$

Для (7,4) - коду Хемінга перевірочними рівняннями будуть

$$\nu_1 = \nu_3 + \nu_5 + \nu_7; \quad \nu_2 = \nu_3 + \nu_6 + \nu_7; \quad \nu_4 = \nu_5 + \nu_6 + \nu_7,$$

де ν_1, ν_2, ν_4 – перевірочні символи.

Елементи синдрому визначаються з наступних виразів

$$S_0 = \nu_1 + \nu_3 + \nu_5 + \nu_7; \quad S_1 = \nu_2 + \nu_3 + \nu_6 + \nu_7; \quad S_2 = \nu_4 + \nu_5 + \nu_6 + \nu_7.$$

Корегуючи здатність коду Хемінга може бути збільшена введенням додаткової перевірки на парність. У цьому випадку перевірочна матриця для розглянутого (7, 4) - коду буде мати наступний вигляд

$$H_{(7,4)} = \begin{vmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{vmatrix},$$

а кодова відстань коду $d_0 = 4$.

Перевірочні рівняння використовуються для побудови кодера, а синдромні - декодера коду Хемінга.

Існує модифікований код Хеммінга, у якому додається ще один розряд загальної парності. Такий код дозволяє виявляти й усувати подвійні помилки. Наприклад, записуємо число 1011010, а зчитуємо зі збоєм: 1**1**110**0**0. Слід інвертувати зчитане число, послати його на вхід і знову зчитати (знову з таким же збоєм): *A*=1**1**110**0**0=*A*=0000111, посилаємо й читаємо *B* = 0**1**001**0**1, складаємо *A* і *B*, отримуємо: *C* = 0**1**000**1**0, у якому одиниці показують, у яких розрядах відбувся збій.

На рис. 4.3 показана блок-схема алгоритму роботи декодуючого пристрою модифікованого (7,3) коду Хемінга в режимі корекції помилок.

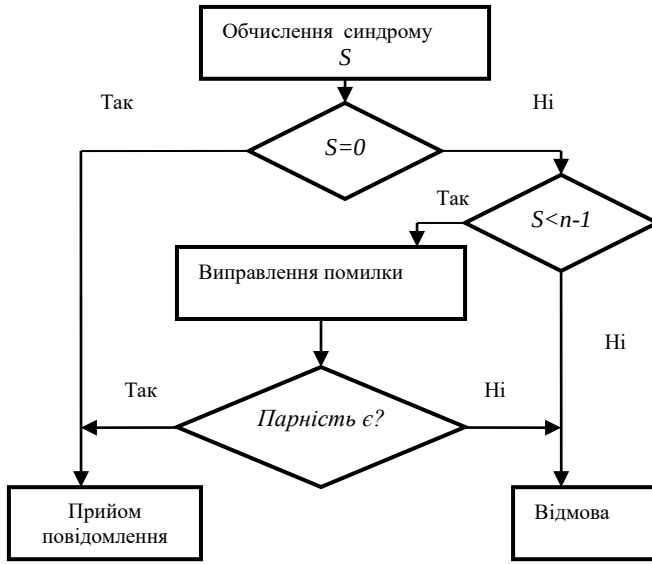


Рис. 4.3. Блок-схема алгоритму роботи декодуючого пристрою модифікованого (7,3) коду Хемінга в режимі корекції помилок

4.3. Контроль за модулем

Контроль виконання арифметичних і логічних операцій можна здійснювати за допомогою контрольних кодів, що становлять собою остачі від ділення чисел на деякий модуль, наприклад „mod 2”, „mod 3”. Такий контроль називається контролем за модулем. Для двійкових чисел цей модуль зазвичай дорівнює або більше 3. Розрізняють *числовий* і *цифровий контроль за модулем*.

При *числовому методі* код заданого числа визначається як найменша додатна остача від ділення числа на вибраний модуль.

Наприклад, визначити контрольний код чисел A = 125 і B = 89 за модулем 11. Отримаємо такі результати: 125/11=12 (4), тобто контрольний код дорівнює 4; 89/11 = 8 (1), тобто контрольний код дорівнює 1.

При *цифровому методі* контролю контрольний код числа утворюється діленням суми цифр числа на вибраний модуль. У цьому варіанті можливі два шляхи отримання контрольного коду:

1) ділення суми цифр на модуль;

2) сумування цифр за вибраним модулем.

Наприклад, визначимо контрольний код для чисел 153 і 41 за модулем 3.

Сума цифр 153 дорівнює 9, а сума цифр 41 – 5. Розділимо ці суми на 3. Отримаємо відповідно контрольні коди 0 і 2.

Контроль по модулю визнаний одним з основних методів функціональної діагностики цифрових автоматів. Проте реалізація контролю по модулю для сучасних ЦА є складною задачею, що пов'язане з високими вимогами до продуктивності інформаційних систем, діапазону представлення чисел і точності обчислень.

Такому рівню вимог відповідають матричні цифрові автомати, які працюють із числами представленими у форматі з плаваючою комою, в яких мантиси чисел обробляються із збереженням єдиних форматів представлення операндів і результатів. Витрати обладнання і часу на обчислення в матричних ЦА знаходяться відповідно в квадратичній і лінійній залежності від розрядності оброблюваних чисел.

Для зменшення цих витрат доцільно використовувати методи скороченого виконання операцій. Проте скорочення обчислень суперечить використовуванню традиційного контролю по модулю, вимагаючи його розвитку.

4.4. Ітеративні коди

Ітеративні коди використовують при контролі передач масивів кодів між зовнішнім запам'ятовуючим пристроєм та цифровим автоматом, між двома цифровими автоматами та у інших випадках. Ітеративний код утворюється шляхом додавання додаткових розрядів по парності до кожного рядка і кожного стовпця масиву слів (двовимірний код) яки передаються. Крім того, парність може визначатися і по діагональних елементах масиву слова (багатовимірний) код.

Етапи перевірки за допомогою ітеративних кодів можуть бути сформульовані наступним чином.

1. Інформаційні символи a_{ij} повідомлень розташовуються у вигляді прямокутника, див. табл. 4.3.
2. Формуються додаткові елементи коду $b_1, b_2, ..., b_{m_1}$, які забезпечують перевірку відповідних рядків на парність, див. табл. 4.4.

Таблиця 4.3

Інформаційні символи a_{ij} повідомлень

a_{11}	a_{12}	⋯	a_{1i}	⋯	a_{1,m_2}
a_{21}	a_{22}	⋯	a_{2i}	⋯	a_{2,m_2}
⋮	⋮	⋯	⋮	⋯	⋮
a_{j1}	a_{j2}	⋯	a_{ji}	⋯	a_{j,m_2}
⋮	⋮	⋯	⋮	⋯	⋮
$a_{m_1,1}$	$a_{m_1,2}$	⋯	$a_{m_1,i}$	⋯	a_{m_1,m_2}

3. Формують додаткові елементи коду $c_1, c_2, ..., c_{m_2}$, які забезпечують перевірку відповідних стовпців на парність, див. табл. 4.5.

Таблиця 4.4

Додаткові елементи b_{m_1}

a_{11}	a_{12}	$\cdots$	a_{1i}	$\cdots$	a_{1,m_2}	b_1
a_{21}	a_{22}	$\cdots$	a_{2i}	$\cdots$	a_{2,m_2}	b_2
$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$
a_{j1}	a_{j2}	$\cdots$	a_{ji}	$\cdots$	a_{j,m_2}	b_j
$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$
$a_{m_1,1}$	$a_{m_1,2}$	$\cdots$	$a_{m_1,i}$	$\cdots$	a_{m_1,m_2}	b_{m_1}

4. Формується, так звана – «перевірка перевірок» - d , яка доповнює до парного значення коди рядків та стовбців, див. табл. 4.6.

Таким чином, в таблиці 4.6 побудовано ітеративний код, який є добутком двох кодів з перевіркою на парність.

Таблиця 4.5

Додаткові елементи c_{m_2}

a_{11}	a_{12}	$\cdots$	a_{1i}	$\cdots$	a_{1,m_2}	b_1
a_{21}	a_{22}	$\cdots$	a_{2i}	$\cdots$	a_{2,m_2}	b_2
$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$
a_{j1}	a_{j2}	$\cdots$	a_{ji}	$\cdots$	a_{j,m_2}	b_j
$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$
$a_{m_1,1}$	$a_{m_1,2}$	$\cdots$	$a_{m_1,i}$	$\cdots$	a_{m_1,m_2}	b_{m_1}
c_1	c_2	$\cdots$	c_i	$\cdots$	c_{m_2}	

Таблиця 4.6

Формування «перевірка перевірок» - d

a_{11}	a_{12}	$\cdots$	a_{1i}	$\cdots$	a_{1,m_2}	b_1
a_{21}	a_{22}	$\cdots$	a_{2i}	$\cdots$	a_{2,m_2}	b_2
$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$
a_{j1}	a_{j2}	$\cdots$	a_{ji}	$\cdots$	a_{j,m_2}	b_j
$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$
$a_{m_1,1}$	$a_{m_1,2}$	$\cdots$	$a_{m_1,i}$	$\cdots$	a_{m_1,m_2}	b_{m_1}
c_1	c_2	$\cdots$	c_i	$\cdots$	c_{m_2}	d

Розглянемо наступний приклад. Потрібно побудувати ітеративний код для наступних чотирьох повідомлень, див. табл. 4.7.

Таблиця 4.7

Повідомлення для перевірки за допомогою ітеративного коду

0	0	0	1	0
1	1	0	0	1
1	0	0	1	1
1	0	0	0	0

У даному випадку $m_1 = 4, \quad m_2 = 5$. У відповідності до розглянутої методики, побудуємо ітеративний код, див. табл. 4.8.

Таблиця 4.8

Повідомлення для перевірки за допомогою ітеративного коду

0	0	0	1	0	1
1	1	0	0	1	1
1	0	0	1	1	1
1	0	0	0	0	1
1	1	0	0	0	0

Мінімальна кодова відстані цього коду $d_1 = 2$. Для початкового коду (табл. 4.7) мінімальна кодова відстані також *2*. Таким чином, обидва коди мають можливість знаходити окремі помилки.

Розглянемо процедуру декодування ітеративного коду. Наприклад, під час передачі повідомлення у цифровому автоматі виникла помилка (0→1), див. табл. 4.9.

Таблиця 4.9

Приклад процедури декодування ітеративного коду

0	0	0	1	0	1
1	1	**1**	0	1	1
1	0	0	1	1	1
1	0	0	0	0	1
1	1	0	0	0	0

Під час передачі повідомлення викривлений елемент опиниться на перехресті другого рядку та третього стовпцю.

Для виправлення помилки слід зробити наступні дії.

1. Перевіримо рядки на парність. З цією метою для кожного рядка обчислимо суму елементів коду за модулем 2.

$$S_j = a_{j1} \oplus a_{j2} \oplus a_{j3} \oplus a_{j4} \oplus a_{j5} \oplus b_j; \quad j = \overline{1,5}. \qquad (4.1)$$

Скориставшись виразом (4.1), отримаємо $S_1 = 0$, $S_2 = 1$, $S_3 = 0$, $S_4 = 0$, $S_5 = 0$. Парному значенню кількості одиниць відповідає нульове значення S_j. Отже, помилка у другому рядку, $S_2 = 1$.

2. Перевіримо стовбці на парність. З цією метою для кожного стовпця обчислимо суму елементів коду за модулем 2.

$$R_i = a_{1i} \oplus a_{2i} \oplus a_{3i} \oplus a_{4i} \oplus a_{5i} \oplus c_i; \quad j = \overline{1,6}. \qquad (4.2)$$

Отримаємо $R_1 = 0$, $R_2 = 0$, $R_3 = 1$, $R_4 = 0$, $R_5 = 0$, $R_6 = 0$. Помилка у третьому стовбці, $R_3 = 1$.

3. На перехресті другого рядку та третього стовпця розташований елемент a_{23}. Отже, цей елемент порушує парність рядка та стовпця, див. табл. 4.10.

Таблиця 4.10

Результат декодування

0	0	0	1	0	1
1	1	**1**	0	1	1
1	0	0	1	1	1
1	0	0	0	0	1
1	1	0	0	0	0

$S_2 = 1,\ R_3 = 1.$

4. Для відновлення кодової комбінації елемент a_{23} слід інвертувати (1→0).

Розглянутий алгоритм використовування ітеративних кодів дозволяє корегувати будь яку одиноку помилку.

4.5. Коди Ріда - Соломона

Одними з найбільш перспективних з точки зору практичного використання є блокові коди Ріда-Соломона (РС-коди).

На практиці, з-поміж двох кодів із приблизно рівними коректувальними властивостями перевага буде віддана коду з менш складною технічною реалізацією кодера. Для переважної більшості практично цікавих кодів, технічна складність кодера визначається складністю декодера, оскільки алгоритм декодування на порядок складніше алгоритму кодування.

Код Ріда-Соломона має мінімальну відстань, і є кодом з максимально досяжною кодовою відстанню, тобто при фіксованих *n* і *k* не існує коду, у якого мінімальна відстань більша, ніж у коду Ріда-Соломона

$$d_{min} = n - k + 1.$$

У цей час відомі декілька алгебраїчних алгоритмів декодування РС-кодів, заснованих на розв'язанні системи перевірочних рівнянь, що задають розташування і значення помилок. При цьому багато питань

розробки неалгебраїчних (кореляційних) алгоритмів декодування, заснованих на урахуванні структурних властивостей коду, розглядалися недостатньо повно. Зокрема, не досліджені властивості циклічності по частоті РС-кодів і можливості їхнього використання для побудови неалгебраїчних алгоритмів кодування і кореляційного декодування, що допускають побудову економічних схем кодерів.

РС-код довжини $N = q - 1$, розмірності $1 \le k \le q - 2$ і потужності

$$J = q^k \text{ над } GF(q), \quad q = p^m,$$

де m - степінь розширення простого поля, а p - просте число, є циклічним кодом з породжуючим поліномом вигляду [50]:

$$g(x) = \prod_{i=0}^{d-2} (x - \alpha^{m_0(m_1+i)}),$$

де $d = r + 1$- мінімальна кодова відстань $PC(N, K)$ - коду;

$r = N - K$ - число перевірочних символів;

α - один з первісних елементів $GF(q)$;

m_0, m_1 - цілочисельні параметри: $1 \le m_0 \le q - 2$, $0 \le m_1 \le q - 1$.

Для низькошвидкісних кодів, що володіють високою коректувальною здатністю, використання неалгебраїчних методів дозволяє спростити декодер.

Дослідження схем захисту інформації на кодах Ріда-Соломона показали, що зміна будь-якого з параметрів (n, α, m) породжувального многочлену коду Ріда-Соломона призводить до утворення нового суміжного класу коду. В цьому випадку, якщо на приймальній стороні не відомий закон зміни параметрів , то декодування є складним обчислювальним завданням. Крім того, коди Ріда-Соломона мають добрі ансамблеві структурні властивості; змінюючи *q*-ічну основу алфавіту, можна виправляти як одиночні, так і пакети помилок.

Відмітною властивістю кодів Ріда-Соломона є те, що вони допускають відносно просту схемотехнічну реалізацію схемотехніки.

4.6. Контроль арифметичних операцій

Арифметичні операції можна представити у вигляді послідовності таких елементарних операцій:

- передача слова;
- зсув;
- взяття оберненого коду;
- додавання.

Операцію зсуву можна представити як передачу слова з i-го розряду в ($i+x$)-і або ($i-x$)-і розряди. Отже, контроль зсуву можна здійснити за методом парності-непарності.

Контроль виконання арифметичних операцій (додавання, віднімання, множення) можна здійснити методом контролю за модулем. Одночасно з виконанням операції над числами та сама операція проводиться над їхніми контрольними кодами, і контрольний код результату основної операції порівнюється з результатом операції над контрольними кодами вихідних чисел. При незбігу фіксується помилка.

Розглянемо приклад. Знайти контрольні коди чисел $A=571_{10}=1000111011$ і $B=329_{10}=0101001001$ і контрольний код їх суми за методом контролю за модулем 3. При діленні числа A на 3 маємо остачу 01_2 (1_{10}), а при діленні числа B на 3 – остача дорівнює 10_2 (2_{10}). Просумуємо ці числа та їхні остачі:

$$\begin{array}{r} A = 1000111011\ (01) \\ \underline{B = 0101001001\ (10)} \\ A + B = 1110000100\ (11) \end{array}$$

Якщо розділити суму на 3, остача буде 0, і якщо розділити суму контрольних кодів на 3, остача також буде 0. Отже, операція пройшла без збою.

Виконаємо цифровий контроль на тому самому прикладі. Сума цифр числа $A=1000111011$ дорівнює 6, отже, контрольний код дорівнює 0. Сума цифр $B=0101001001$ дорівнює 4, отже, контрольний код цього числа є 1. Таким чином, числа A і B з урахуванням контрольних кодів запишуться так:

$$\begin{array}{r} A = 1000111011\ (00) \\ \underline{B = 0101001001\ (01)} \\ A + B = 1110000100\ (01) \end{array}$$

Якщо скласти цифри результату, отримаємо 4, отже, контрольний код буде 1 і сума остач теж 1.

Запитання для самоперевірки

1. Як виконується контроль виконання арифметичних і логічних операцій у цифрових автоматах?
2. Які Ви знаєте помилки результату в цифрових автоматах? Перерахуйте їх і поясніть їхнє походження.
3. Що таке коди Хеммінга?
4. Чому необхідно кодувати інформацію під час введення її в цифрові автомати?
5. Які причини викривлення інформації під час її обробки в цифрових автоматах, зокрема інформаційних системах?
6. Поясніть призначення бітів парності.
7. Що становить собою цифровий метод контролю?
8. Що таке мінімальна кодова відстань?
9. Що таке систематичні коди?
10. Що таке система контролю цифрового автомата?
11. Чому необхідна надмірність інформації для контролю роботи цифрового автомата?

Література для самостійної підготовки за темою:
8, 14, 15, 23, 24, 27, 35.

Розділ 5. ОСНОВИ АЛГЕБРИ ЛОГІКИ

5.1. Основні поняття алгебри логіки

Логіка – це наука про закони й форми мислення. *Математична логіка* – наука про застосування математичних методів для розв'язання логічних задач. Логічною основою цифрових автоматів або інформаційних системах є *алгебра логіки (булева алгебра)* – одна з основних частин математичної логіки.

Усі цифрові автомати побудовані на елементах, які виконують ті або інші логічні операції. Одні елементи забезпечують обробку двійкових символів, які представляють інформацію, інші – комутацію каналів, якими інформація передається, треті – управління та ін. Електричні сигнали, які діють на входах і виходах перерахованих елементів, мають зазвичай два різних рівні, отже, вони можуть бути представлені двійковими символами, наприклад, 1 і 0. Позначають здійснення якоїсь події, наприклад, високої напруги в певній точці схеми цифрового автомата, символом 1. Цей символ називається *логічною одиницею* (не те саме, що двійкова 1). Відсутність події позначається символом 0 і називається *логічним нулем.*

Розглянемо наступні фізичні форми представлення інформації в інформаційних системах. Найбільш поширеними способами фізичного представлення інформації є імпульсний і потенційний:

- імпульс або його відсутність;
- високий або низький потенціал електричної напруги;
- високий потенціал або його відсутність.

При імпульсному способі відображення код одиниці ідентифікується наявністю електричного імпульсу, відповідно код нуля – його відсутністю. Імпульс характеризується амплітудою й тривалістю. Тривалість імпульсу повинна бути менша за часовий такт інформаційної системи. Форма й амплітуда сигналу до уваги не беруться. З вищесказаного випливає, що для аналізу й синтезу схем у цифрових автоматах, наприклад, інформаційних системах, може бути використаний апарат алгебри логіки, який також оперує двома поняттями – *істина* або *хибність*.

Таким чином, кожному сигналу на вході чи виході двійкового елемента ставиться у відповідність *логічний аргумент*, який може

приймати одне з двох значень: стан логічної одиниці (подія істинна) і стан логічного нуля (подія хибна).

Теорія логічних основ цифрових автоматів використовує значну кількість специфічних термінів і понять, які будуть наведені у цьому і наступних розділах.

Функція $f(x_1, x_2,...x_n)$ називається логічною (перемикальною), або булевою, якщо вона, так само як і її аргументи x_i, може приймати лише два значення: 0 або 1.

Алгебра логіки є алгеброю станів, а не алгеброю чисел, тому цю алгебру називають також алгеброю висловлювань.

Висловлюванням називається твердження, про яке можна з упевненістю сказати, істинне воно чи хибне. Якщо висловлювання істинне, то говорять, що його значення істинності дорівнює одиниці. Якщо ж висловлювання хибне – його значення істинності дорівнює нулю. Висловлювань одночасно істинних і хибних не буває.

Приклади висловлювань:

- „Зараз падає дощ" – це твердження може бути істинним або хибним;
- „Земля – центр всесвіту" – хибне твердження;
- „Київ – столиця України" – істинне твердження.

Висловлювання бувають простими й складними. Прості окремі висловлювання – це логічні аргументи, їх прийнято позначати буквами латинського алфавіту, наприклад, якщо просте висловлювання x істинне, то $x=1$, якщо ж хибне, то $x=0$.

Висловлювання з різним змістом позначають різними буквами й вважають різними. Два висловлювання називають *еквівалентними*, якщо істинності їх однакові. Еквівалентність висловлювань позначають знаком рівності або тотожності, наприклад, запис $x=y$ означає, що висловлювання x і y або істинні, або хибні одночасно.

Знаки, що об'єднують логічні аргументи в складні висловлювання, тобто в логічні функції, є знаками логічних дій, точніше *логічних зв'язок*, а не математичних дій.

Сукупність значень аргументів логічної функції називається *набором (або точкою)* і може позначатися, зокрема, як $x_1, x_2, ...x_n$, де x_i дорівнює нулю або одиниці $(i=1, 2,...n)$. Очевидно, що набір значень аргументів фактично становить собою деяке двійкове число. Кожному набору значень аргументів приписується номер, який дорівнює двійковому числу, що відповідає значенню цього набору. Наприклад, для чотирьох аргументів 0, 0, 0, 0 – нульовий набір; 0, 0, 0, 1 – перший набір; 0, 0, 1, 0 – другий набір; 1, 0, 1, 0 – десятий набір і т. д.

Таким чином, логічна функція (функція алгебри логіки) – це функція $y=f(x_1, x_2,...x_n)$, яка приймає значення 0 або 1 на наборі логічних аргументів

x_1, x_2,...x_n. Кожній логічній функції цього набору аргументів також прийнято приписувати номер: 0, 1, 2,.....

Будь-яку булеву функцію можна задати за допомогою таблиці, у якій з усіма можливими наборами значень двійкових аргументів зіставлено відповідні їм значення функції. Така таблиця називається таблицею істинності, оскільки вона визначає істинність або хибність складного висловлювання залежно від істинності або хибності складових висловлювань. Для функцій одного аргументу може існувати всього чотири різні булеві функції F_1, F_2, F_3 і F_4, представлені в табл. 5.1.

Таблиця 5.1

Таблиця істинності для функцій одного аргументу

x	F_1	F_2	F_3	F_4
0	0	0	1	1
1	0	1	0	1

З таблиці випливає, що функції F_1 і F_4 не залежать від аргументу і є відповідно константами 0 і 1, а функція F_2 повторює значення аргументу, тобто $F_2=x$. Функція F_3 називається запереченням або інверсією аргумента x.

Для функцій двох аргументів може існувати 16 (і тільки 16) різних функцій. Таблицю істинності цих функцій наведено нижче.

Таблиця 5.2

Таблиця істинності для функцій двох аргументів

x_1	x_2	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Яким би складним не був логічний зв’язок між логічною функцією та її аргументами, цей зв’язок завжди можна представити у вигляді найпростіших логічних операцій:

- заперечення (операція **„НЕ”**);
- логічне додавання (операція **„АБО”**).
- логічне множення (операція **„І”**).

Розглянемо ці операції детально.

Запереченням називається такий логічний зв'язок між вхідним логічним аргументом *x* і вихідним логічним аргументом *y*, при якій *y* істинний лише тоді, коли *x* хибний, і, навпаки, *y* хибний лише тоді, коли істинний *x*. Цю функціональну залежність представлено в табл. 5.3. Такі таблиці, що відображають відповідність усіх можливих комбінацій значень двійкових аргументів значенням логічної функції, називають таблицями істинності.

За допомогою логіко-математичної символіки логічна функція **„НЕ"** аргументу *y* записується як $y = \overline{x}$ (читається – „*y є не x"*). Якщо, наприклад, *x* – твердження про наявність сигналу в точці схеми ЦА, то *y* відповідає твердженню – сигнал відсутній.

Таблиця 5.3

Операція „НЕ"

x	*y*
0	**1**
1	**0**

Функція *f(x)*, яка приймає значення, протилежне значенню *x*, – *логічне заперечення (інверсія)*, або функція **„НЕ" (NOT)**, може позначатися одним з таких способів:

$$f(x) = \overline{x} = \neg x .$$

Звернімо увагу на наступні функції.

Логічним додаванням кількох аргументів називається така функція, яка хибна лише тоді, коли одночасно хибні й усі аргументи, що додаються. Таблицю істинності операції логічного додавання наведено нижче (див. табл. 5.4).

Таблиця 5.4

Операція „АБО"

x_1	x_2	*y*
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**1**

Логічне додавання також називається *диз'юнкцією* й позначається так:

$$y(x_1, x_2) = x_1 + x_2 = x_1 \vee x_2.$$

Наприклад, вираз $y=x_1 \vee x_2$ читається так: „*y є x_1 або x_2*".

Логічним множенням кількох аргументів називається така функція, яка істинна лише тоді, коли одночасно істинні всі аргументи, що множаться. Таблицю істинності операції логічного множення наведено нижче (див. табл. 5.5).

Таблиця 5.5.

Операція „І"

x_1	x_2	y
0	**0**	**0**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**

Логічне множення також називається *кон'юнкцією* й позначається так:

$$y(x_1, x_2) = x_1 \cdot x_2 = x_1 \wedge x_2 = x_1 \,\&\, x_2.$$

Наприклад, вираз $y=x_1 \cdot x_2$ читається так: „*y є x_1 і x_2*".

Функція **„НЕ-І"** *(штрих Шеффера* **NAND***)* – це функція, яка хибна тоді, коли всі аргументи істинні. Умовна позначка цієї функції:

$$y(x_1, x_2) = x_1 \mid x_2.$$

Це читається так: „функція у хибна, тобто дорівнює 0, коли обидва аргументи x_1 і x_2 одночасно істинні, тобто дорівнюють одиниці, і функція істинна, тобто дорівнює одиниці, коли або обидва аргументи одночасно хибні, або ж хоча б один з них хибний.

Функція **„НЕ-АБО"** (стрілка Пірса або **NOR**) – це функція, яка істинна лише тоді, коли всі аргументи хибні. Умовна позначка цієї функції:

$$y(x_1, x_2) = x_1 \downarrow x_2 .$$

Це читається так: „функція у хибна, тобто дорівнює 0, коли хоча б один з її аргументів x_1 або x_2 істинний, або ж обидва одночасно істинні, тобто дорівнюють одиниці, і функція істинна, тобто дорівнює одиниці, коли обидва аргументи одночасно хибні".

Функція **„ЯКЩО-ТО”** (**IF-THEN** *імплікація*) – це функція, яка хибна тоді й лише тоді, коли x_1 істинний і x_2 хибний. Аргумент x_1 називається *посилкою*, а x_2 – *наслідком*. Її умовна позначка:

$$y(x_1, x_2) = x_1 \rightarrow x_2.$$

Функція **„АБО”**, що виключає **(XOR)** – це функція $y(x_1, x_2)$, яка позначається знаком $\forall$. Ця операція реалізує функцію *нерівнозначності,* тобто фактично реалізується процедура **додавання за модулем 2**, яка позначається знаком $\oplus$:

$$y(x_1, x_2) = x_1 \ \forall x_2 = x_1 \oplus x_2.$$

З усіх наведених вище визначень зрозуміло, що в алгебрі логіки всі знаки дій: $\wedge$ або &, $\vee$ або +, $\rightarrow$, $\forall$ і т. д., на відміну від звичайної алгебри, є знаками *логічних зв’язок*, тобто *логічних дій*, а не знаками арифметичних дій.

У табл. 5.6 наведено приклади всіх елементарних логічних функцій від двох аргументів x_1 і x_2.

Таблиця 5.6

Перелік елементарних логічних функцій від двох аргументів

№ функції		Значення			Приклад написання
		x_1	x_2	y	
f_0	константа нуль	0	0	0	*f0*
f_1	кон’юнкція	0	0	0	$x_1 \wedge x_2$
f_2	заборона зворотної імплікації	0	0	0	$\overline{x_2 \rightarrow x_1}$
f_3	функція дорівнює x_1	1	0	1	$x_1 \wedge \bar{x}_2 \vee x_1 \wedge x_2 = x_1$
f_4	заборона імплікації	0	0	0	$\overline{x_1 \rightarrow x_2}$
f_5	функція дорівнює x_2	0	1	1	$\bar{x}_1 \wedge x_2 \vee x_1 \wedge x_2 = x_2$
f_6	додавання за модулем 2	0	1	1	$x_1 \oplus x_2$
f_7	диз’юнкція	0	1	1	$x_1 \vee x_2$
f_8	функція Пірса	1	0	0	$x_1 \downarrow x_2$
f_9	рівнозначність	1	0	0	$x_1 \equiv x_2$
f_{10}	інверсія x_2	1	0	1	$\bar{x}_2$
f_{11}	зворотня імплікація	1	0	1	$x_2 \rightarrow x_1 = x_1 \leftarrow x_2$
f_{12}	інверсія x_1	1	1	0	$\bar{x}_1$
f_{13}	імплікація	1	0	0	$x_1 \rightarrow x_2$
f_{14}	функція Шеффера	1	1	0	$x_1 \mid x_2$
f_{15}	константа одиниця	1	1	1	*f1*

Наведемо наступні поняття алгебри логіки. Дві функції вважаються *рівносильними* одна одній, якщо вони приймають на всіх можливих наборах їх аргументів одні й ті самі значення.

Логічний аргумент x_i є *дійсним*, якщо значення логічної функції $f(x_1, x_2, ..., x_n)$ змінюється при зміні x_i. В протилежному випадку такий аргумент є *фіктивним*, тобто не є її *дійсним* аргументом.

Необхідність введення двох останніх понять обумовлена наступним. При аналізі деякої невідомої логічної функції *(логічної схеми)*, для якої необхідно сформувати аналітичний вираз, не всі логічні аргументи, що підключаються для цього аналізу, можуть бути аргументами цієї функції, що й виявляється в підсумку проведеного аналізу. У подальшому буде показано, що будь-які логічні операції над логічними аргументами можна звести до певної сукупності елементарних логічних функцій, наприклад, таких як **„І”, „АБО”, „НЕ-АБО”**.

Елементарні логічні операції в цифрових автоматах виконуються також над двійковими числами, *порозрядно*. У табл. 5.7 наведено кілька прикладів виконання логічних операцій над двома двійковими числами.

Таблиця 5.7

Приклади логічних операцій над двома двійковими числами

Логічне множення **„І”**	Логічне додавання **„АБО”**	Заперечення **„НЕ”**
01011010 11110000 01010000	0101010 1111000 1111010	01011010 10100101

Підстановка в логічну функцію замість її аргументів інших логічних функцій називається *суперпозицією*.

Система логічних функцій називається *функціонально повною*, якщо за допомогою функцій, які входять у цю систему, застосовуючи операції суперпозиції й підстановки, можна отримати складну логічну функцію.

Елементи, які реалізують найпростіші логічні функції, схематично представляються у вигляді прямокутників, на полі яких зображується символ, що позначає функцію, виконувану цим елементом. На рис. 5.1 показано умовні позначення елементів, які реалізують логічні функції **„І”, „АБО”, „НЕ”.** Вхідні змінні прийнято зображувати зліва, а вихідні – справа. Вважається, що передача інформації відбувається зліва направо.

При переході від логічних функцій до логічних схем зазвичай приймається, що логічній одиниці (1) відповідає імпульсний сигнал стандартної амплітуди, наприклад, високого рівня, а логічному нулю (0) –

низького рівня й зазвичай фіксованої тривалості. Причому, всі вхідні сигнали мають надходити на кожен елемент одночасно.

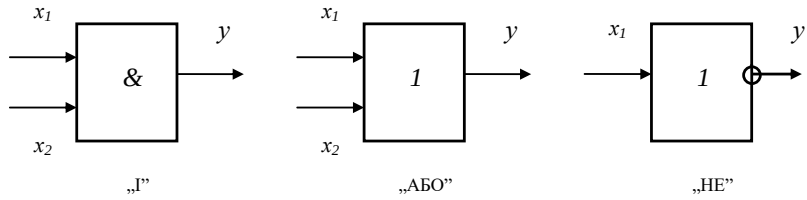


Рис. 5.1. Умовні позначки елементів, які реалізують логічні функції „І”, „АБО”, „НЕ”

Як будь-яку логічну функцію можна реалізувати за допомогою відповідної комбінації елементарних логічних функцій, так будь-яку логічну схему можна сформувати за допомогою відповідної комбінації логічних елементів.

5.2. Властивості елементарних функцій алгебри логіки

У цьому підрозділі ми розглянемо основні закони, аксіоми й теореми алгебри логіки.

В алгебрі логіки є чотири основні закони: *переставний* (властивості комутативності); *сполучний* (властивості асоціативності); *розподільний* (властивості дистрибутивності), *інверсії* (правило де Моргана).

Деякі закони звичайної алгебри можуть бути застосовані й до алгебри логіки.

Переставний закон:
для множення $(A \cdot B) = (B \cdot A)$;
для додавання $(A + B) = (B + A)$.

Сполучний закон:
для множення $A \cdot (B \cdot C) = (A \cdot B) \cdot C$;
для додавання $A + (B + C) = (A + B) + C$.

Розподільний закон:
$A \cdot (B + C) = A \cdot B + A \cdot C$;
$(A + B) \cdot C = A \cdot C + B \cdot C$.

Алгебра логіки має специфічні аксіоми й теореми, основні з яких, необхідні для аналізу й синтезу логічних ланцюгів або схем, наведено в табл. 5.8.

Таблиця 5.8

Специфічні аксіоми й теореми алгебри логіки

1)	$A = 1$, якщо $A \neq 0$	$A = 0$, якщо $A \neq 1$
2)	Якщо $A = 0$, то $\overline{A} = 1$	Якщо $A=1$, то $\overline{A} = 0$
3)	$0 + 0 = 0$	$0 \cdot 0 = 0$
4)	$0 + 1 = 1$	$1 \cdot 0 = 0$
5)	$1 + 1 = 1$	$1 \cdot 1 = 1$
6)	$\overline{0} = 1$	$\overline{1} = 0$
7)	$A \vee 0 = A$	$A \cdot 1 = A$;
8)	$A \vee 1 = 1$	$A \cdot 0 = 0$;
9)	$A \vee A = A$	$A \cdot A = A$;
10)	$A \vee \overline{A} = 1$	$\overline{A} \cdot A = 0$
11)	$\overline{A + B} = \overline{A} \cdot \overline{B}$	$\overline{A \cdot B} = \overline{A} + \overline{B}$
Правило де Моргана		
12)	$\overline{A \wedge B} = \overline{A} \vee \overline{B}$ і $\overline{A \vee B} = \overline{A} \wedge \overline{B}$	
Правило подвійного заперечення		
13)	$\overline{(\overline{A})} = A$	
Закон склеювання		
14)	$A \cdot B \vee A \cdot \overline{B} = A$	$(A \vee B) \cdot (A \vee \overline{B}) = A$

Аксіоми й теореми, записані в другій колонці (тобто зліва), називаються двоїстими аксіомам і теоремам, записаним у третій колонці (тобто справа).

Двоїстість визначається як зміна всіх знаків операції **„І”** на знаки операції **„АБО”**, усіх знаків операції **„АБО”** на знаки операції **„І”**, усіх нулів на одиниці й усіх одиниць на нулі.

Двоїстість є однією з основних властивостей алгебри логіки й означає, що якщо *f(A, B, C)* і *f′(A, B, C)* – двоїсті функції, то

$$f(\overline{A,B,C}) = f'(\overline{A},\overline{B},\overline{C}).$$

Закони де Моргана є однією з ілюстрацій властивості подвійності й, як уже було відзначено, можуть бути сформульовані у вигляді:

$$\overline{A \cdot B \cdot C} = \overline{A} + \overline{B} + \overline{C};$$
$$\overline{A + B + C} = \overline{A} \cdot \overline{B} \cdot \overline{C}.$$

Із законів Моргана випливає, що є можливість виражати кон'юнкцію через диз'юнкцію й заперечення або диз'юнкцію – через кон'юнкцію й заперечення. Закони де Моргана й висновки з них справедливі для будь-якої кількості аргументів.

Функція додавання за модулем 2 представляється так:

$$A \otimes B = \overline{A} \cdot B + A \cdot \overline{B}.$$

Для цієї функції справедливими є такі аксіоми:

$$A \otimes A = 0; \quad A \otimes A \otimes A = A; \quad A \otimes \overline{A} = 1; \quad A \otimes 1 = \overline{A}; \quad A \otimes 0 = A.$$

На основі розглянутих аксіом і властивостей елементарних логічних функцій можна, вивести закон поглинання:

$$A \vee (A \cdot B) = A;$$
$$A \cdot (A + B) = A.$$

Функції **„І", „АБО", „НЕ"** через функцію Шеффера виражаються так:

$$\overline{A \cdot B} = A \mid B;$$
$$A + B = \overline{A \cdot B} = \overline{A} \mid \overline{B}.$$

Функцію Пірса може бути описано такими виразами:

$$A \downarrow B = \overline{A + B} = \overline{A} \cdot \overline{B}.$$

Для цієї функції справедливими є аксіоми:

$$A \downarrow A = \overline{A}; \quad A \downarrow 0 = \overline{A}; \quad A \downarrow \overline{A} = 0; \quad A \downarrow 1 = 0.$$

Функції **„І", „АБО", „НЕ"** виражаються через функцію Пірса так:

$$A \cdot B = (A \downarrow A) \downarrow (B \downarrow B); \quad A + B = (A \downarrow B) \downarrow (A \downarrow B); \quad \overline{A} = A \downarrow A.$$

Слід зазначити, що логічні вирази, які містять операції диз'юнкції й кон'юнкції, можна перетворювати (розкривати дужки, виносити спільний множник, переставляти місцями члени й т. ін.) за правилам алгебри, вважаючи формально диз'юнкцію операцією додавання, а кон'юнкцію – операцією множення**.** При цьому слід пам'ятати, що в алгебрі логіки, на відміну від звичайної алгебри, знак + або знак $\vee$ означають логічну зв'язку **„АБО"**, а знак множення „·" або знаки $\wedge$, і & – логічну зв'язку **„І"** [8, 14,

24]. *Булевий вираз* становить собою формулу, що складається з логічних констант і логічних аргументів, з'єднаних знаками логічних операцій.

5.3. Аналітичне представлення функцій алгебри логіки

Існують різні способи представлення логічних функцій.

Найпростішим варіантом є словесний опис функції, наприклад: *„функція трьох аргументів приймає значення 1, якщо два будь-яких аргументи чи всі три дорівнюють 1. В усіх інших випадках функція дорівнює 0".*

Можна використовувати табличний спосіб представлення функції. При цьому способі функція представляється своєю таблицею істинності (див. табл. 5.3 – 5.5). Нижче наведено приклад такої таблиці (див. табл. 5.9) для деякої логічної функції трьох аргументів *f(A, B, C).*

Таблиця 5.9

Приклад таблиці логічної функції трьох аргументів

№ набору	Аргументи функції *f*			Значення функції
	A	B	C	*f*
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	0
Примітка	Зазвичай у таблиці істинності стовпець з номером набору не наводять			

Табличний спосіб є наочним, але у випадку *складних функцій алгебри логіки* (ФАЛ) стає некомпактним. Простіше виглядає аналітичний запис у вигляді формул. До розгляду аналітичної форми представлення ФАЛ введемо кілька нових понять.

Аргументи та їх інверсії часто називають *літералами*.

Терм – це група логічних аргументів у прямій або інверсній формі, тобто група літералів, об'єднаних одним і тим самим знаком логічної зв'язки: логічного додавання або ж логічного множення. У термі кожний

аргумент або його заперечення трапляється лише один раз, тобто в терм може входити або аргумент, або його заперечення.

Диз'юнктивний терм (макстерм) – це логічна функція, яка зв'язує всі аргументи в прямій або інверсній формі, тобто літерали, знаком диз'юнкції.

Наприклад, $F_1 = A + \overline{B} + C + D; \quad F_2 = A \vee B.$

Макстерм називають також *конституантою нуля*, оскільки ця логічна функція дорівнює нулю лише тоді, коли всі її аргументи дорівнюють нулю одночасно [8, 14, 15, 22, 24, 25, 31 та ін.].

Кон'юнктивний терм (мінтерм) – це логічна функція, яка зв'язує аргументи в прямій або інверсній формі, тобто літерали, знаком кон'юнкції, наприклад, $F_1 = \overline{A} \,\&\, B \,\&\, \overline{C} \,\&\, D; \quad F_2 = A \wedge B \wedge C$.

Мінтерм називають також *конституантою одиниці*, оскільки ця функція дорівнює одиниці лише тоді, коли всі її аргументи одночасно дорівнюють одиниці.

Ранг терма – *r*, визначається кількістю літералів, що входять у цей терм, наприклад, для *мінтерма* $F = \overline{A} \cdot B \cdot \overline{C} \cdot D \cdot G$ ранг терма дорівнює п'яти ($r=5$), а для *макстерма* $G = \overline{A} + B + \overline{C}$ - $r = 3$.

Будь-яка задана таблицею функція алгебри логіки може бути представлена аналітично у вигляді диз'юнкції скінченного числа *мінтермів*, на кожному з яких функція дорівнює одиниці [8, 9, 15, 27 та ін.]:

$$f(x_1, x_2, \ldots, x_n) = F_1 \vee F_2 \vee \ldots \vee F_n = \Sigma F_i = \vee F_i, \qquad (5.1)$$

де i – номери наборів, на яких функція дорівнює 1;

Σ – знак диз'юнкції, який об'єднує всі *мінтерми* F_i.

Розглянемо приклад, у якому необхідно записати функцію в аналітичному вигляді, якщо її задано таблицею (див. табл. 5.10).

Таблиця 5.10

Таблиця істинності функції трьох аргументів

A	B	C	*f(A, B, C)*
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Згідно з виразом (5.1) отримаємо:

$$f(A,B,C) = F_1(0,0,0) + F_4(0,1,1) + F_5(1,0,0) = \overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot \overline{C}.$$

Слід відзначити, що функція, представлена таблицею істинності, може бути визначена не лише її одиничними значеннями, але й нульовими.

Наприклад, для функції, заданої таблицею (5.10) відповідно маємо:

$$\bar{f}(A,B,C) = F_2(0,0,1) + F_3(0,1,0) + F_6(1,0,1) + F_7(1,1,0) + F_8(1,1,1).$$

Будь-яка задана таблицею функція алгебри логіки може бути задана аналітично у вигляді кон'юнкції скінченного числа *макстермів*, на кожному з яких функція дорівнює нулю:

$$f(x_1, x_2, \ldots, x_n) = H_1 \wedge H_2 \wedge \ldots \wedge H_n = \bigwedge H_i = \prod H_i.$$

Наприклад, використовуючи правила двоїстості, результат наведеного рішення можна представити в такому вигляді:

$$f(A,B,C) = (A + B + \overline{C}) \,\&\, (A + \overline{B} + C) \,\&\, (\overline{A} + B + \overline{C}) \,\&\, (\overline{A} + \overline{B} + C) \,\&\, (\overline{A} + \overline{B} + \overline{C}).$$

Для запису тієї самої функції алгебри логіки можна використовувати багато різних форм. Форми, які представляють суми *елементарних* добутків, називаються *нормальними диз'юнктивними формами (НДФ)*, наприклад, $f(A,B,C) = C + A \cdot \overline{B} + B \cdot C + \overline{A} \cdot \overline{B} \cdot C.$

Під *елементарним* розуміється такий добуток, у якому співмножниками є лише окремі аргументи або їх заперечення, наприклад, вираз $f(A,B,C) = \overline{B} \cdot C \vee A \cdot \overline{B}$ містить два елементарних добутки, кожен з яких складається з двох співмножників.

Нормальна кон'юнктивна форма (НКФ) – це кон'юнктивне об'єднання *макстермів*, що містить *макстерми* різних рангів, наприклад, $f(A,B,C) = (A + \overline{B}) \cdot (B + C) \cdot (\overline{A} + \overline{B} + C).$

Кожна логічна функція в загальному випадку може мати кілька нормальними диз'юнктивними формами або нормальна кон'юнктивна форма.

Розрізняють також *мінімальні нормальні диз'юнктивні формами* і нормальна кон'юнктивна форма логічних функцій.

Нормальна диз'юнктивна форма заданої функції називається *мінімальною*, якщо кількість букв (літералів), які вона містить, буде не більша, ніж у будь-якій іншій нормальній диз'юнктивній формамі тієї самої функції (**саме букв, а не аргументів**) [24], наприклад, нормальна

диз'юнктивна формама $(A \cdot B \vee A \cdot \overline{B} \cdot D \vee B \cdot A)$ містить сім літералів, але три аргументи – *A, B, D.*

Мінімальна форма представлення функції алгебри – це така форма, яка містить мінімальну кількість термів, що мають мінімальні ранги.

Вище (див. табл. 5.10) розглянуто процедуру формування аналітичної форми функції алгебри логіки за її таблицею істинності. Можна виконати й зворотну процедуру, тобто побудувати таблицю істинності за логічним виразом. Якщо в логічному виразі є *n* аргументів, то для них у таблиці передбачено *n* колонок й одну колонку для значень функції. Після цього, починаючи з нульового набору, кожен набір вписують у таблицю й для нього обчислюють значення функції, яке також заносять у таблицю.

5.4. Досконалі нормальні форми

Нормальні кон'юнктивна й диз'юнктивна форми не дають однозначного уявлення про функцію. Таке уявлення отримуємо лише з використанням досконалих нормальних форм (ДНФ). Досконалою нормальною формою називають також стандартною або ж канонічною нормальною формиою [2, 7, 8, 14, 15, 22-25, 27 та ін.].

Нормальна канонічна форма відрізняється від нормальної форми тим, що завжди містить терми лише максимального рангу й дає однозначне уявлення про функцію [15].

Будь-яка функція алгебри логіки, крім абсолютно істинної функції, може бути представлена в єдиній досконалій кон'юнктивній нормальній формі (ДКНФ або ДНКФ) або в єдиній досконалій диз'юнктивній нормальній формі (ДДНФ або ДНДФ).

Досконалою нормальною кон'юнктивною формою визначається як сума елементарних добутків, у яких кожен аргумент трапляється рівно один раз або із запереченням, або без нього. Розглянемо наступний приклад.

Для перетворення функції $f(A,B,C) = \overline{B} \cdot C \vee A \cdot \overline{B}$, необхідно додати кожний елементарний добуток аргументів, яких не вистачає, так, щоб тотожність перетворення не було порушено, тобто

$$f(A,B,C) = \overline{B} \cdot C \vee A \cdot \overline{B} = (A \vee \overline{A}) \cdot \overline{B} \cdot C \vee A \cdot \overline{B} \cdot (C \vee \overline{C}).$$

У першому випадку елементарний добуток доповнився співмножником $(A \vee \overline{A})$, що дорівнює одиниці, у другому – співмножником $(C \vee \overline{C})$.

Після розкриття дужок і приведення подібних членів отримаємо функцію, записану в досконалій диз'юнктивній нормальній формі:

$$f(A,B,C) = A \cdot \overline{B} \cdot C \vee \overline{A} \cdot \overline{B} \cdot C \vee A \cdot \overline{B} \cdot \overline{C}.$$

В отриманому виразі кожний аргумент (або його заперечення) міститься по одному разу в кожному елементарному добутку. Отримана функція перетворюється в логічну одиницю при трьох різних комбінаціях значень вхідних аргументів:

A=1, B=0, C=1 – перша комбінація;
A=0, B=0, C=1 – друга комбінація;
A=1, B=0, C=0 – третя комбінація.

Кожній комбінації відповідає елементарний добуток, що дорівнює одиниці, для всіх інших комбінацій вхідних аргументів – нулю.

Таблиця істинності такої функції (див. табл. 5.11) містить три рядки, у яких функція дорівнює одиниці. Кожному з цих рядків відповідає по одній з перерахованих вище комбінацій вхідних аргументів, тобто таблиця істинності функції має стільки рядків, де функція перетворюється в одиницю, скільки елементарних добутків містить її досконала диз'юнктивна нормальна форма.

Отже, досконала нормальна кон'юнктивна форма – це стандартний або канонічний добуток *макстермів* максимального рангу певної функції, а досконала нормальна диз'юнктивна форма – стандартна або канонічна сума *мінтермів* максимального рангу певної функції.

Таблиця 5.11

Таблиця істинності функції

A	B	C	*f(A, B, C)*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Запис досконалої диз'юнктивної нормальної форми за її таблицею істинності можна здійснити так. Необхідно для всіх комбінацій вхідних аргументів, що обертають функцію в одиницю, записати елементарні

добутки, інвертуючи аргументи, які приймають на цій комбінації нульове значення, а всі отримані елементарні добутки з'єднати знаками логічного додавання. Застосувавши це правило до функції, представленої в табл. 5.11, отримаємо три елементарні добутки:

$\overline{A} \cdot \overline{B} \cdot C$ – перший елементарний добуток;
$A \cdot \overline{B} \cdot \overline{C}$ – другий елементарний добуток;
$A \cdot \overline{B} \cdot C$ – третій елементарний добуток.

Якщо з'єднати ці добутки знаками логічного додавання, то отримаємо вихідний вираз, з якого розпочали розгляд цього прикладу – $\overline{A} \cdot \overline{B} \cdot C \vee A \cdot \overline{B} \cdot \overline{C} \vee A \cdot \overline{B} \cdot C$.

Отже, досконала нормальна диз'юнктивна форма функції знаходиться так. Виписують ряд добутків, тобто *мінтермів*, усіх аргументів і з'єднують їх знаками диз'юнкції. Кількість добутків має дорівнювати кількості наборів, на яких задана функція дорівнює одиниці. У кожному *мінтермі* над аргументом, значення якого в цьому наборі дорівнювало нулю, ставиться знак від'ємності.

Досконалу кон'юнктивну нормальну форму функції формують так. Формується добуток диз'юнкцій, тобто *макстермів*, усіх аргументів з кількістю співмножників, що дорівнює кількості наборів, на яких задана функція обертається в нуль. У кожному *макстермі* над аргументом, що дорівнює одиниці в цьому наборі, ставиться знак від'ємності.

Довільна нормальна диз'юнктивна форма переводиться в досконалу нормальну диз'юнктивну форму таким чином.

Нехай $f_{ндф} = F_1$ – *мінтерм* функції, представленої в нормальній диз'юнктивній формі. Тоді цей *мінтерм* перетворюється так:

$$f_{сндф} = F_1 \cdot x_i \vee F_1 \cdot \overline{x}_i = F_1 \cdot (x_i \vee \overline{x}_i),$$

де x_i – аргумент, який не входить у *мінтерм* F_1.

Якщо *максимальний ранг* для функції дорівнює r, а *мінімальний ранг j-го мінтерма* дорівнює k, то попереднє перетворення необхідно застосувати до *j-го мінтерма* $(r - k)$ раз.

Розглянемо приклад. Перетворимо логічну функцію, задану в нормальній диз'юнктивній формі:

$$f(A,B,C,D) = A \cdot \overline{B} \vee B \cdot \overline{C} \cdot D \vee \overline{A} \cdot \overline{C} \cdot D \vee A \cdot B \cdot C \cdot D.$$

$F_1 \qquad F_2 \qquad F_3 \qquad F_4$

Оскільки *терм* F_4 має максимальний ранг і в нього входять усі аргументи функції, то скористаємося прийомом перетворення $f_{снф} = F_1 \cdot x_i \vee F_1 \cdot \bar{x}_i = F_1 \cdot (x_i + \bar{x}_i)$ по черзі до інших трьох термів:

$$F_1 = A \cdot \overline{B} \cdot (C \vee \overline{C}) = A \cdot \overline{B} \cdot C \vee A \cdot \overline{B} \cdot \overline{C}.$$

Обидва члени отриманого виразу помножимо на $(D \vee \overline{D})$.

У результаті отримаємо:

$$F_1 = A \cdot \overline{B} \cdot C \cdot D \vee A \cdot \overline{B} \cdot \overline{C} \vee A \cdot \overline{B} \cdot \overline{C} \cdot D \vee A \cdot \overline{B} \cdot C \cdot \overline{D} \vee A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}.$$

Аналогічно розглянемо терми F_1 і F_3.

$$F_2 = B \cdot \overline{C} \cdot D \cdot (A \vee \overline{A}) = A \cdot \overline{B} \cdot C \cdot D \vee \overline{A} \cdot B \cdot \overline{C} \cdot D.$$

$$F_3 = \overline{A} \cdot \overline{C} \cdot D \cdot (B \vee \overline{B}) = \overline{A} \cdot B \cdot \overline{C} \cdot D \vee \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot D.$$

Після приведення подібних членів визначаємо досконалу нормальну диз'юнктивну форму цієї функції:

$$f(A,B,C,D) = A \cdot B \cdot \overline{C} \cdot D \vee A \cdot \overline{B} \cdot C \cdot D \vee A \cdot \overline{B} \cdot C \cdot \overline{D} \vee A \cdot \overline{B} \cdot \overline{C} \cdot D \vee$$
$$\overline{A} \cdot \overline{B} \cdot \overline{C} \cdot D \vee A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} \vee \overline{A} \cdot B \cdot \overline{C} \cdot D \vee A \cdot B \cdot C \cdot D.$$

Довільна нормальну кон'юнктивну форму переводиться в досконалу нормальну кон'юнктивну форму шляхом такого перетворення [15]. Нехай задано *макстерм* $f_{нкф} = H_1$.

Тоді $f_{снкф} = H_1 \vee x_i \cdot \bar{x}_i = (H_1 \vee x_i)(H_1 \vee \bar{x}_i)$. Перетворимо в досконалу нормальну кон'юнктивну форму в логічну функцію:

$$f(A,B,C) = (A \vee B) \cdot (B \vee \overline{C}) \cdot (A \vee B \vee C).$$

H_1 H_2 H_3

Застосовуємо правило перетворень по черзі до *макстермів* H_1 і H_2, оскільки *макстерм* H_3, має максимальний ранг:

$$H_1 = (A \cdot B) \vee C \cdot \overline{C} = (A \vee B \vee C) \cdot (A \vee B \vee \overline{C}).$$

$$H_2 = (B \vee \overline{C}) \vee A \cdot \overline{A} = (A \vee B \vee \overline{C}) \cdot (\overline{A} \vee B \vee \overline{C}).$$

Після спрощень досконалої нормальної кон'юнктивної форми функції прийме остаточний вигляд:

$$f(A,B,C) = (A \vee B \vee C) \cdot (A \vee B \vee \overline{C}) \cdot (\overline{A} \vee B \vee \overline{C}).$$

У табл. 5.12 наведено досконалу нормальну кон'юнктивну форму і досконалу нормальну диз'юнктивну форму для елементарних логічних функцій, див. також табл. 5.6.

Таблиця 5.12

Досконала нормальна кон'юнктивна форма і досконала диз'юнктивна нормальна форма для елементарних логічних функцій

№ функції	Функції Алгебри логіки	*ДНДФ* функції	*ДНКФ* функції
f_0	0	y	$(x \vee y) \cdot (x \vee \overline{y}) \cdot (\overline{x} \vee y) \cdot (\overline{x} \vee \overline{y})$
f_1	$x \wedge y$	$x \wedge y$	$(x \vee y) \cdot (x \vee \overline{y}) \cdot (\overline{x} \vee y)$
f_2	$x \wedge \overline{y}$	$x \wedge \overline{y}$	$(x \vee y) \cdot (x \vee \overline{y}) \cdot (\overline{x} \vee \overline{y})$
f_3	x	$x \cdot \overline{y} \vee x \cdot y$	$(x \vee y) \cdot (x \vee \overline{y})$
f_4	$\overline{x} \wedge y$	$\overline{x} \wedge y$	$(x \vee y) \cdot (\overline{x} \vee y) \cdot (\overline{x} \vee \overline{y})$
f_5	y	$(\overline{x} \cdot y) \vee (x \cdot y)$	$(x \vee y) \cdot (\overline{x} \vee y)$
f_6	$x \otimes y$	$(\overline{x} \cdot y) \vee (x \cdot \overline{y})$	$(x \vee y) \cdot (\overline{x} \vee \overline{y})$
f_7	$x \vee y$	$(\overline{x} \cdot y) \vee (x \cdot \overline{y}) \vee (x \cdot y)$	$x \vee y$
f_8	$x \downarrow y$	$\overline{x} \cdot \overline{y}$	$(x \vee \overline{y}) \cdot (\overline{x} \vee y) \cdot (\overline{x} \vee \overline{y})$
f_9	$x \equiv y$	$(\overline{x} \cdot \overline{y}) \vee (x \cdot y)$	$(x \vee \overline{y}) \cdot (\overline{x} \vee y)$
f_{10}	$\overline{y}$	$(\overline{x} \cdot \overline{y}) \vee (x \cdot \overline{y})$	$(x \vee \overline{y}) \cdot (\overline{x} \vee \overline{y})$
f_{11}	$y \to x$	$(\overline{x} \cdot \overline{y}) \vee (x \cdot \overline{y}) \vee (x \cdot y)$	$x \vee \overline{y}$
f_{12}	$\overline{x}$	$(\overline{x} \cdot \overline{y}) \vee (\overline{x} \cdot y)$	$(\overline{x} \vee y) \cdot (\overline{x} \vee \overline{y})$
f_{13}	$x \to y$	$(\overline{x} \cdot \overline{y}) \vee (\overline{x} \cdot y) \vee (x \cdot y)$	$\overline{x} \vee y$
f_{14}	$x \mid y$	$(\overline{x} \cdot \overline{y}) \vee (x \cdot \overline{y}) \vee (\overline{x} \cdot y)$	$\overline{x} \vee \overline{y}$
f_{15}	1	$(x \cdot \overline{y}) \vee (\overline{x} \cdot y) \vee (x \cdot y) \vee (\overline{x} \cdot \overline{y})$	y

5.5. Системи функцій алгебри логіки

Довільна логічна функція може бути представлена багаточленом вигляду [15, 24, 25]:

$$f(x_1,x_2,..,x_n)=a_0 \oplus a_1 \cdot x_1 \oplus a_2 \cdot x_2 \oplus ... \oplus a_n \cdot x_n \oplus a_{n+1} x_1 \cdot x_2 \oplus ... \oplus a_{n+2} x_1 \cdot x_3 \oplus .. \oplus a_N \cdot x_1 \cdot x_2 \cdot x_3 .. x_n,$$

де $a_0, a_1,..., a_N$ – деякі константи, що дорівнюють нулю або одиниці;

$\oplus$ – знак операції додавання за модулем два.

У записі конкретної логічної функції у вигляді багаточлена коефіцієнти $a_0, a_1,..., a_N$ випадають, оскільки члени, при яких ці коефіцієнти дорівнюють нулю, можна опустити, а коефіцієнти, що дорівнюють одиниці, не писати.

Щоб логічну функцію, задану таблицею її значень, представити у вигляді багаточлена, слід цю функцію записати у вигляді суми конституант одиниці (рівних одиниці на тих самих наборах, на яких дорівнює одиниці задана функція). Потім усі аргументи, що входять в отриманий вираз із запереченням, замінити за допомогою співвідношення $\bar{x} = x \otimes 1$, розкрити дужки й привести подібні члени з урахуванням, що в цьому випадку $x \oplus x \oplus x .. \oplus x = x$, якщо n непарне й $x \oplus x \oplus x ... \oplus x = 0$, якщо n парне.

Розглянемо приклад. Представити функцію Шеффера у вигляді багаточлена.

$$f(x, y) = (\bar{x} \cdot \bar{y}) \otimes (\bar{x} \cdot y) \otimes (x \cdot \bar{y}).$$

Використовуючи співвідношення $x \otimes 1 = \bar{x}$ і $x \cdot (y \otimes z) = (x \cdot y) \otimes (x \cdot z)$, після перетворень отримаємо такий вираз:

$$f(x, y) = 1 \otimes x \otimes x \otimes y \otimes y \otimes xy \otimes xy \otimes xy.$$

Приводячи подібні члени, остаточно отримаємо:

$$f(x, y) = 1 \otimes xy.$$

Слід зазначити, що існує кілька класів функцій алгебри логіки, які також важливі для логічного аналізу [2].

Клас лінійних функцій *($K_л$)*. Логічна функція називається *лінійною*, якщо вона представляється поліномом першого ступеня:

$$f(x_1, x_2, ...x_n) = k_0 \oplus k_1 x_1 \oplus k_2 x_2 \oplus .... \oplus k_n x_n,$$

де коефіцієнти k_i дорівнюють нулю або одиниці.

Функції двох аргументів, які є лінійними, наведено в табл. 5.13.

Таблиця 5.13

Лінійні логічні функції

Логічна функція	Приклад написання
Константа нуль	$f_0(x_1,x_2) = 0$
Аргумент x_1	$f_3(x_1,x_2) = x_1$
Аргумент x_2	$f_5(x_1,x_2) = x_2$
Додавання за модулем 2	$f_6(x_1,x_2) = x_1 \oplus x_2$
Рівнозначність	$f_9(x_1,x_2) = x_1 \equiv x_2$
Інверсія x_2	$f_{10}(x_1,x_2) = \overline{x}_2$
Інверсія x_1	$f_{12}(x_1,x_2) = \overline{x}_1$
Константа одиниця	$f_{15}(x_1,x_2) = 1$

Клас функцій, що зберігають нуль *(K_0)*. Якщо функція на нульовому наборі аргументів дорівнює нулю, то говорять, що *функція зберігає нуль*:

$$f(0, 0, ..., 0) = 0.$$

Логічні функції двох аргументів, що зберігають нуль, наведено в табл. 5.14.

Таблиця 5.14

Логічні функції двох аргументів, що зберігають нуль

Логічна функція	Приклад написання
Константа нуль	$f_0(x_1,x_2) = 0$
Кон'юнкція	$f_1(x_1,x_2) = x_1 \wedge x_2$
Заборона зворотної імплікації	$f_2(x_1,x_2) = \overline{x_2 \rightarrow x_1}$
Аргумент x_1	$f_3(x_1,x_2) = x_1$
Заборона імплікації	$f_4(x_1,x_2) = \overline{x_1 \rightarrow x_2}$
Аргумент x_2	$f_5(x_1,x_2) = x_2$
Додавання за модулем 2	$f_6(x_1,x_2) = \oplus x_2$
Диз'юнкція	$f_7(x_1,x_2) = x_1 \vee x_2$

Клас функцій, що зберігають одиницю *(K_1)*. Якщо функція на одиничному наборі аргументів дорівнює одиниці, то говорять, що така *функція зберігає одиницю*:

$$f(1, 1, ..., 1) = 1.$$

Клас монотонних функцій *($K_м$)*. Функція називається *монотонною,* якщо при будь-якому зростанні набору значення цієї функції не зменшується. Розрізняють монотонно зростаючі і монотонно спадаючі функції.

Якщо значення кожного аргументу одного набору більше або дорівнює значенню того же аргументу другого набору, то говорять, що перший набір не менший за другий. Такі набори називаються *порівнянними*, наприклад: 1,1,0,1 > 1,1,0,0; 0,1,1,0,1 > 0,1,0,0,0; 1,0,1,1 > 1,0,0,0.

Якщо ж деякі із значень аргументів першого набору більші або рівні, а інші менші за значення тих самих аргументів другого набору, то такі набори називаються *незрівнянними*. Приклади незрівнянних наборів: 0,1 і 1,0; 1,0,0 і 0,0,1; 1,1,1,0 і 1,0,0,1.

Для функцій двох аргументів монотонними є такі функції: $f_0(x_1,x_2)$, $f_1(x_1,x_2)$, $f_3(x_1,x_2)$, $f_5(x_1,x_2)$, $f_7(x_1,x_2)$, $f_{15}(x_1,x_2)$.

Клас самоподвійних функцій *(K_C)*. Логічна функція називається *самоподвійною,* якщо на кожній парі протилежних наборів вона приймає протилежні значення, тобто

$$f(\overline{x_1, x_2, x_3, ...x_n}) = f(\bar{x}_1, \bar{x}_2, \bar{x}_3, ...\bar{x}_n).$$

Два набори називаються *протилежними,* якщо всі значення аргументів одного набору протилежні значенням аргументів іншого набору, наприклад: 1,0,1,1,0,0 і 0,1,0,0,1,1; 1,1,1,1 і 0,0,0,0, тобто для того, щоб отримати протилежний набір, достатньо замінити в цьому наборі нулі одиницями, а одиниці – нулями.

Для функцій двох аргументів самоподвійними є такі функції: $f_3(x_1,x_2)$, $f_5(x_1,x_2)$, $f_{10}(x_1,x_2)$, $f_{12}(x_1,x_2)$.

Усі перераховані класи логічних функцій мають таку властивість: будь-яка логічна функція, отримана за допомогою суперпозиції й підстановки з функцій одного класу, обов'язково належатиме до цього ж класу [2, 6, 7, 14, 18, 20, 24, 27 та ін.].

Функції $f_8(x_1,x_2)$ (функція Пірса) і $f_{14}(x_1,x_2)$ (функція Шеффера) не належать до жодного з указаних класів.

Для того щоб система функцій алгебри логіки була повною, необхідно й достатньо, щоб вона містила хоча б одну функцію:

- яка не зберігає нуль,
- яка не зберігає одиницю,
- яка не є лінійною,
- яка не є монотонною,
- яка не є самоподвійною.

Базисом називається функціонально повна система елементарних логічних функцій, за допомогою якої будь-яка ФАЛ може бути представлена суперпозицією вихідних функцій.

До базису належить система функцій **„І”, „АБО”, „НЕ”** (базис 1). Базисами є також системи, що містять функції **„І”, „НЕ”** (базис 2), **„АБО”, „НЕ”** (базис 3), складаються з функції Шеффера (**„І-НЕ”**) (базис 4) чи функції Пірса **„АБО-НЕ”** (базис 5) [2, 22]. Цей перелік показує, що базиси можуть бути *надмірними* (базис 1) і *мінімальними* (базиси 4 і 5). Базис є *мінімальним*, якщо видалення хоча б одної функції перетворює систему функції алгебри логіки в неповну.

Проблема найпростішого представлення логічних функцій зводиться до вибору не лише базису, але й форми найбільш економного представлення цих функцій.

Функціонально повними будуть п’ять систем елементарних функцій, п’ять базисів, наведених у табл. 5.15.

Таблиця 5.15

Функціонально повні системи елементарних логічних функцій (базиси)

№ базису	**Система елементарних логічних функцій**		Примітка
1	$y = \bar{x}$; $y = x_1 \cdot x_2$; $y = x_1 \vee x_2$.	**„НЕ”** **„І”** **„АБО”**	Надмірний Базис
2	$y = \bar{x}$; $y = x_1 \cdot x_2$.	**„НЕ”** **„І”**	
3	$y = \bar{x}$; $y = x_1 \vee x_2$.	**„НЕ”** **„АБО”**	
4	$y = \overline{x_1 \cdot x_2}$.	Штрих Шеффера – **„І-НЕ”**.	
5	$y = \overline{x_1 + x_2}$.	Стрілка Пірса – **„АБО-НЕ”**.	

Кожна з перерахованих простих логічних функцій апаратно реалізується за допомогою певних електронних логічних елементів [9, 10, 15, 18, 30].

Функції, яких не вистачає, можна отримати на основі відомих правил алгебри логіки. Таким чином, достатньо мати один тип логічного елемента **„І-НЕ”** чи **„АБО-НЕ”**, щоб на його основі побудувати всю багатоманітність логічних схем [2, 7, 8, 14, 15, 22, 23, 24, 25, 27, 30, 31 та ін.].

5.6. Числове й геометричне представлення логічних функцій

Для спрощення запису логічних функцій замість повного перерахування *термів* часто використовують лише номери наборів, для яких функція приймає значення одиниці. Наприклад, якщо деяка задана таблицею функція *f(A,B,C)* приймає значення одиниці на наборах з номерами 0, 3, 4 і 6, то її можна представити так:

$$f(A,B,B) = \vee F(0, 3, 4, 6), \text{ або } f(A, B, C) = \Sigma F(0, 3, 4, 6).$$

Якщо ця ж функція на наборах 1, 2, 5, 7 приймає значення 0, то її можна представити так:

$$f(A,B,C) = \wedge F(1, 2, 5, 7) \text{ або } f(A,B,C) = \Pi F(1, 2, 5, 7).$$

Таку форму запису називають *числовою.*

Перший вид такої форми використовують, коли функція представляється в досконалій нормальній диз'юнктивній формі, а другий – у досконалій нормальній кон'юнктивній формі.

Багато перетворень, виконуваних над логічними функціями, іноді зручно інтерпретуються з використанням їх геометричних представлень, наприклад, функцію двох аргументів можна інтерпретувати як певну площину, задану в системі координат *x, y*. Якщо відкласти по кожній осі одиничні відрізки *x* і *y*, то вийде квадрат, вершини якого відповідають комбінаціям аргументів (див. рис. 5.2).

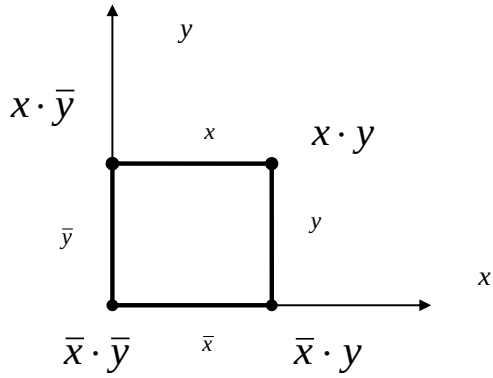


Рис. 5.2. Геометричне представлення логічних функцій

З рисунка випливає, що дві вершини, які належать тому самому ребру, є сусідніми й з'єднуються по аргументу, який змінюється вздовж цього ребра, наприклад, $\bar{x} \cdot \bar{y} + x \cdot \bar{y} = \bar{y}$, оскільки з властивостей логічних функцій випливає таке тотожне перетворення – $f(x, y) = \bar{x} \cdot \bar{y} + x \cdot \bar{y} = (\bar{x} + x) \cdot \bar{y} = 1 \cdot \bar{y} = \bar{y}$.

Для функцій трьох аргументів геометричне представлення виконують у вигляді тримірного куба. Ребра куба поглинають вершини. Грані куба поглинають свої ребра й, отже, вершини.

У випадку аналізу функції, яка містить чотири аргументи, будують чотиримірний куб [7, 8, 22]. У геометричному розумінні кожен набір аргументів x_1, x_2, x_3,...., x_n можна розглядати як n-мірний вектор, що визначає точку n-мірного простору. Тому всю множину наборів, на яких визначено функцію n аргументів, представляють у вигляді вершин n-мірного куба. Координати вершин куба указуються в порядку, що відповідає порядку перерахування аргументів у записі функції. Відзначаючи точками вершини, у яких функція приймає значення, що дорівнює одиниці, отримуємо геометричне представлення функції алгебри логіки [7,8, 14, 15, 27].

Запитання для самоперевірки

1. Що таке математична логіка?
2. Що таке логічний нуль і логічна одиниця?
3. Дайте визначення операціям заперечення, логічного множення й логічного додавання.
4. Перерахуйте основні закони алгебри логіки.
5. Що таке геометричне представлення логічної функції?
6. Як виконують тотожні перетворення булевих виразів?
7. Що таке мінімізація в алгебрі логіки?
8. Дайте визначення нормальної диз'юнктивної форми.
9. Сформулюйте правило запису досконалої нормальної диз'юнктивної форми.
10. Назвіть основні відмінні особливості логічних аругемнтів.
11. Які способи представлення логічних функцій?
12. Якщо в таблиці істинності логічної функції **„I"** двох аргументів *(A, B)* значення всіх елементів не інверсні, то яка вийде логічна функція двох аргументів?
13. Запишіть довільну кон'юнкцію для логічної функції чотирьох аргументів.

14. Перетворіть логічні функції до нормальної диз'юнктивної форми:

$$f_1(A,B,C) = (\overline{A \cdot B \vee A \cdot C}) \cdot \overline{A \cdot C};$$
$$f_2(A,B,C) = (\overline{A \vee B \vee \overline{C}}) \vee \overline{A} \cdot \overline{B}.$$

Запишіть досконалу нормальну диз'юнктивну форму логічної функції виду:

$$f(A,B,C) = (\overline{A} \cdot C) \vee B.$$

15. Наведіть умовні позначки основних логічних елементів.
16. Дайте визначення термінів кон'юнкція і диз'юнкція.

Література для самостійної підготовки за темою:

2,7,8, 14, 15,22-25, 27, 30, 31.

Розділ 6. МІНІМІЗАЦІЯ ЛОГІЧНИХ ФУНКЦІЙ

Розглянуті в попередньому розділі тотожні перетворення дозволяють суттєво спростити вирази логічних функцій. Оскільки кожна логічна функція реалізується за допомогою певного набору пристроїв, то, отже, чим менше елементів містить вираз, тим простіша електронна схема цифрового автомату, яка реалізує відповідну йому функцію. Таким чином, значний інтерес становить вивчення методів мінімізації логічних функцій.

Розрізняють аналітичні й табличні методи мінімізації.

У принципі будь-яка логічна функція може бути спрощена безпосередньо за допомогою аксіом і теорем логіки, але такі перетворення вимагають певних обґрунтувань і перевірки на наявність помилок. Тому більш доцільно використовувати спеціальні алгоритмічні методи мінімізації, що дозволяють проводити спрощення функції просто, швидко й безпомилково. До таких методів належать, наприклад, *метод Квайна, метод карт Карно* та ін. Ці методи найбільш придатні для звичайної технічної практики [8, 10, 12, 14, 15, 21-27, 30, 35].

6.1. Аналітичні методи

Аналітичний метод мінімізації логічних функцій полягає в послідовному застосуванні до деякої формули законів і правил алгебри логіки. Однак цей метод не піддається чіткій алгоритмізації. Дії, використовувані під час реалізації цього методу, визначаються видом вихідного виразу, що перетворюється, а також досвідом виконавця. Відсутність алгоритмізації процесу мінімізації значно підвищує ймовірність появи помилок і можливість отримання не цілком мінімізованої логічно функції.

Метод безпосередніх перетворень найбільш придатний для простих формул, коли послідовність перетворень очевидна.

Розглянемо приклад застосування методу безпосередніх перетворень на прикладі мінімізації функції трьох аргументів, заданої її досконалою нормальною диз'юнктивною формою*:*

$$y = f(A,B,C) = (\overline{A}\cdot\overline{B}\cdot C)\vee(A\cdot\overline{B}\cdot\overline{C})\vee(A\cdot\overline{B}\cdot C)\vee(A\cdot B\cdot\overline{C})\vee(A\cdot B\cdot C).$$

Об'єднаємо попарно перший і третій, другий і третій, а також четвертий і п'ятий елементарні добутки:

$$y = (\overline{A} \cdot \overline{B} \cdot C \vee A \cdot \overline{B} \cdot C) \vee (A \cdot \overline{B} \cdot \overline{C} \vee A \cdot \overline{B} \cdot C) \vee (A \cdot B \cdot \overline{C} \vee A \cdot B \cdot C).$$

Слід зазначити, що один і той самий елементарний добуток при об'єднанні можна використати кілька разів. У розглядуваному прикладі елементарний добуток $A \cdot \overline{B} \cdot C$ використано двічі. Після винесення за дужки отримаємо:

$$y = \overline{B} \cdot C \cdot (A \vee \overline{A}) \vee A \cdot \overline{B} \cdot (\overline{C} \vee C) \vee A \cdot B \cdot (C \vee \overline{C}) = \overline{B} \cdot C \vee A \cdot \overline{B} \vee A \cdot B.$$

Отримана логічна функція простіша за вихідну досконалу нормальну диз'юнктивну форму, однак вона не є мінімальною. Об'єднавши другий і третій елементарні добутки, після винесення за дужку A остаточно отримаємо:

$$y = \overline{B} \cdot C \vee A \cdot (\overline{B} \vee B) = \overline{B} \cdot C \vee A.$$

Неважко помітити, що в кожній парі об'єднувані елементарні добутки відрізняються лише одним аргументом, який входить у перший добуток із запереченням, а в другий – без заперечення. Такі елементарні добутки називають сусідніми. До сусідніх добутків застосовувана операція склеювання (тобто $A \cdot B \vee A \cdot \overline{B} = A$), у результаті якої зменшується кількість добутків, що сумуються, і на одиницю скорочується кількість аргументів.

Розглянемо ще один приклад. Необхідно вибрати найменше значення з трьох аргументів *A, B, C*.

Перший варіант аналізу дає такі результати:

якщо *A*<*B* і *A*<*C*, то найменше = *A*,
якщо *A*>=*B* і *B*<*C*, то найменше = *B*,
якщо *A*>=*C* і *B*>=*C*, то найменше = *C*.

Проведений аналіз дав неоптимальний результат. Скоротимо його, винісши за дужки деякі порівняння:

```
якщо A<B, то
        якщо A<C, то найменше = A,
        інакше найменше = C.
інакше
        якщо B<C, то найменше = B,
        інакше найменше = C.
```

В оптимізованому варіанті прикладу лише три порівняння замість шести. Більше того, якщо в першому прикладі для досягнення результату

завжди виконується шість порівнянь, то в другому прикладі – лише два порівняння.

У процесі мінімізації логічної функції враховано, у якому базисі ефективніше реалізувати її мінімальну форму за допомогою електронних схем.

Тепер введемо поняття *накриття* для логічних функцій [24, 35].

Нехай на якомусь наборі аргументів функція *f приймає значення* a_1, а функція φ на цьому ж наборі приймає значення a_2, тоді говорять, що функція f на цьому наборі накриває значення a_2 функції φ своїм значенням a_1.

Так, у досконалій нормальній диз'юнктивній формі кожна одиниця заданої логічної функції накривається одиницею лише одного *мінтерма*. Тому кількість *мінтермів*, що входять у досконалу нормальну диз'юнктивну форму, дорівнює кількості наборів, на яких функція дорівнює одиниці.

Якщо деяка логічна функція φ (в окремому випадку елементарний добуток) дорівнює нулю на тих самих наборах, на яких дорівнює нулю інша функція *f*, то говорять, що функція φ входить у функцію *f*. Інакше кажучи, функція φ входить у функцію *f* тоді, коли вона накриває нулями всі нулі функції *f*, а одиниці функції *f* можуть бути накриті й нулями, й одиницями функції φ. Входження позначається: $\varphi \subset f$. Наприклад, у функцію додавання за модулем 2 ($f_6(x_1,x_2)$ (див. табл. 5.6) входитимуть усі функції, які приймають нульові значення на наборах 0,0 і 1,1, тобто функції $f_4(x_1, x_2) = \bar{x}_1 \cdot x_2$, $f_2(x_1, x_2) = x_1 \cdot \bar{x}_2$ і $f_0(x_1, x_2) = 0$. Очевидно, що константа нуль входить у всі функції, а в константу одиниця входять усі функції.

Функцію φ, що входить у цю функцію *f*, називають її *імплікантою*.

Простими *імплікантами* логічної функції f називають такі елементарні добутки або елементарні суми, які самі входять у цю функцію, але ніяка власна частина цих добутків не входить у функцію *f*.

Власною частиною називають добуток, отриманий шляхом виключення з цього добутку одного чи кількох співмножників, наприклад, добуток $x \cdot \bar{y} \cdot z$ має такі власні частини: $x \cdot \bar{y}$, $\bar{y} \cdot z$, $x \cdot z$, x, $\bar{y}$, z.

Прості імпліканти становлять собою найкоротші елементарні добутки або найкоротші суми, що входять у цю логічну функцію.

Для того щоб знайти прості імпліканти логічної функції, треба виписати всі елементарні добутки, що входять у цю функцію, і вибрати з них ті, власні частини яких у цю функцію не входять.

Розглянемо, наприклад, логічну функцію трьох аргументів, задану таблицею 6.1.

У цю функцію входять чотири елементарні добутки – $\bar{x}\cdot y\cdot z$, $x\cdot\bar{y}\cdot\bar{z}$, $x\cdot\bar{y}\cdot z$, $x\cdot\bar{y}$. Простою підстановкою значень аргументів можна перевірити, що ці добутки обертаються в нуль на всіх наборах, на яких дорівнює нулю задана функція. Простими *імплікантами* будуть лише два добутки – $\bar{x}\cdot y\cdot z$ і $x\cdot\bar{y}$. А власна частина $x\cdot\bar{y}$ добутків, що залишилися, входить у задану функцію, тому вони не є елементарними.

Будь-яка логічна функція дорівнює диз'юнкції всіх її простих імплікант. Диз'юнкція всіх простих імплікант називається *скороченою нормальною диз'юнктивною формою* логічної функції.

Таблиця 6.1

Логічна функція трьох аргументів

x	*y*	*z*	*f(x,y,z)*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Прості імпліканти накривають одиницями не одну, а кілька одиниць заданої функції, наприклад, проста імпліканта x_1 функції $f_{11}(x_1,x_2)$, обертається разом з цією функцією в одиницю на наборах 1,0 і 1,1, а проста імпліканта $\bar{x}_2$ – на наборах 0,0 і 1,0. Тому прості імпліканти x_1 і $\bar{x}_2$ спільно накривають одиницями всі одиниці функції f_{11}, яка, унаслідок цього, може бути представлена у формі $f_{11}(x_1,x_2)=x_1\vee\bar{x}_2$, що є скороченою нормальною диз'юнктивною формою функції $f_{11}(x_1,x_2)$.

У табл. 6.2 наведено прості імпліканти й скорочені нормальні диз'юнктивні форми логічних функцій двох аргументів.

Розглянемо процес спрощення логічних виразів з використанням положень теорем і правил алгебри логіки.

Найбільш ефективними для цього процесу є закон поглинання й закон склеювання. Крім того, застосовують *теорему де Моргана, винесення спільних членів за дужки, метод випробування членів* і т. ін.

Розглянемо два простих приклади.

Дано деяку функцію в досконалій нормальній диз'юнктивній формі:

$$f(A,B,C)=\bar{A}\cdot B\cdot C+A\cdot\bar{B}\cdot C+A\cdot B\cdot\bar{C}+A\cdot B\cdot C\,.$$

Таблиця 6.2

Прості імпліканти логічних функцій двох аргументів

Логічні функції	Елементарні добутки, що входять у функцію	Прості імпліканти	Скорочена диз'юнктивна форма
$f_0(x,y)$	y	y	0
$f_1(x,y)$	$x \cdot y$	$x \cdot y$	$x \cdot y$
$f_2(x,y)$	$x \cdot \bar{y}$	$x \cdot \bar{y}$	$x \cdot \bar{y}$
$f_3(x,y)$	$x, x \cdot \bar{y}, x \cdot y$	x	x
$f_4(x,y)$	$\bar{x} \cdot y$	$\bar{x} \cdot y$	$\bar{x} \cdot y$
$f_5(x,y)$	$\bar{x} \cdot y,\ x \cdot y,\ y$	y	y
$f_6(x,y)$	$\bar{x} \cdot y,\ x \cdot \bar{y}$	$\bar{x} \cdot y,\ x \cdot \bar{y}$	$\bar{x} \cdot y \vee x \cdot \bar{y}$
$f_7(x,y)$	$\bar{x} \cdot y,\ x \cdot \bar{y}, x \cdot y, x, y$	$x,\ y$	$x \vee y$
$f_8(x,y)$	$\bar{x} \cdot \bar{y}$	$\bar{x} \cdot \bar{y}$	$\bar{x} \cdot \bar{y}$
$f_9(x,y)$	$\bar{x} \cdot \bar{y}, x \cdot y$	$\bar{x} \cdot \bar{y}, x \cdot y$	$\bar{x} \cdot \bar{y} \vee x \cdot y$
$f_{10}(x,y)$	$\bar{x} \cdot \bar{y}, x \cdot \bar{y}, \bar{y}$	$\bar{y}$	$\bar{y}$
$f_{11}(x,y)$	$\bar{y}, x, x \cdot y, \bar{x} \cdot \bar{y}, x \cdot \bar{y}$	$x, \bar{y}$	$x \vee \bar{y}$
$f_{12}(x,y)$	$\bar{x} \cdot \bar{y}, \bar{x} \cdot y, \bar{x}$	$\bar{x}$	$\bar{x}$
$f_{13}(x,y)$	$\bar{x}, \bar{x} \cdot \bar{y}, y,\ \bar{x} \cdot y, x \cdot y$	$\bar{x}, y$	$\bar{x} \vee y$
$f_{14}(x,y)$	$\bar{x}, \bar{y}, \bar{x} \cdot \bar{y},\ \bar{x} \cdot y, x \cdot \bar{y}$	$\bar{x}, \bar{y}$	$\bar{x} \vee \bar{y}$
$f_{15}(x,y)$	$\bar{x}, \bar{y}, x, y, \bar{x} \cdot y, x \cdot \bar{y},\ \bar{x} \cdot \bar{y}, x \cdot y$	$x, y,\ \bar{x}, \bar{y}$	1

Четвертий доданок є сусіднім з будь-яким з перших трьох, тому є сенс додати цей доданок ще двічі. Тоді отримаємо такий вираз:

$$f(A,B,C) = \bar{A} \cdot B \cdot C + A \cdot \bar{B} \cdot C + A \cdot B \cdot \bar{C} + A \cdot B \cdot C + A \cdot B \cdot C + A \cdot B \cdot C .$$

Шляхом попарного склеювання отримаємо:

$$f(A,B,C) = A \cdot B + B \cdot C + A \cdot C .$$

Цей вираз є тупиковою формою, оскільки не містить надмірних доданків.

Розглянемо другий приклад.

Спростимо вираз $f(A,B,C) = \bar{A} \cdot \bar{B} + A \cdot \bar{B} + A \cdot B + B \cdot C .$ Використовуючи теореми й правила алгебри логіки, отримаємо викладки, представлені в табл. 6.3.

Таким чином, отримаємо $\bar{A} \cdot \bar{B} + A \cdot \bar{B} + A \cdot B + B \cdot C = A + \bar{B} + C .$

У функції, представленій у досконалій нормальній диз'юнктивній формі, відшукують усі сусідні доданки й здійснюють їх склеювання.

Можна застосувати й групове склеювання, тобто у виразі відшукують доданки, що мають спільне застосуванні, які виносять за дужки, а в дужках залишається група доданків більш низького рангу. Якщо ця група складається з 2^k доданків *k-го* рангу, то вона становить собою повну досконалу форму для *k* аргументів і, отже, дорівнює одиниці. У результаті залишається один доданок, що містить аргументи, винесені за дужки. Для зручності склеювання, у тому числі й групового, якщо це необхідно, додають доданки, які вже є у вихідному виразі.

Після того, як можливості *склеювання* вичерпано, роблять спробу виключити надмірні доданки застосуванням *закону поглинання* й, за необхідності, *методу випробовування членів* [14, 15, 23, 35 та ін.].

Таблиця 6.3

Мінімізація логічної функції трьох аргументів

Перетворення	Примітка
Вихідна функція $\overline{A}\cdot\overline{B}+A\cdot\overline{B}+A\cdot B+B\cdot C=$	
$=\overline{A}\cdot\overline{B}+A\cdot\overline{B}+A\cdot\overline{B}+A\cdot B+B\cdot C=$	За правилом повторення $x\vee x=x$
$=\overline{B}\cdot\overline{A}+\overline{B}\cdot A+A\cdot\overline{B}+A\cdot B+B\cdot C=$	За законом переміщення $x\cdot y=y\cdot x$
$=\overline{B}\cdot(\overline{A}+A)+A\cdot(\overline{B}+B)+B\cdot C=$	За законом розподілення $x\cdot(y+z)=x\cdot y+x\cdot z$
$=\overline{B}\cdot(A+\overline{A})+A\cdot(B+\overline{B})+B\cdot C=$	За законом переміщення $x+y=y+x$
$=\overline{B}\cdot 1+A\cdot 1+B\cdot C=$	За правилом додатковості $x+\bar{x}=1$
$=\overline{B}+A+B\cdot C=$	За правилом незмінності $x\cdot 1=x$
$=A+\overline{B}+B\cdot C=$	За законом переміщення $x+y=y+x$
$=A+\overline{B}+C$ Результат	За теоремою $x+\bar{x}\cdot y=x+y$ або $\bar{x}+x\cdot y=\bar{x}+y$

Метод полягає в тому, що попередньо виконують усі можливі згадані процедури склеювання й поглинання, і тим самим виходить скорочена нормальна диз'юнктивна форма. Далі виконується сам метод випробування кожного члена отриманого виразу. Щоб випробувати деякий член, слід виключити його із *скороченої* нормальної диз'юнктивної форми і підставити у вираз, що залишився, такі значення аргументів, при яких виключений член перетворюється на одиницю. Якщо при цьому вираз теж тотожно дорівнює одиниці, то випробовуваний член є зайвим. Після того, як буде відкинуто таким чином всі зайві доданки, виходить *тупикова нормальна диз'юнктивна форма*.

Знайдемо, наприклад, тупикові форми логічної функції, заданої в *скороченій нормальній диз'юнктивній формі* $f(A,B,C)=\overline{A}\cdot\overline{B}\vee A\cdot C\vee\overline{B}\cdot C.$

Відповідно до викладеного вище спочатку випробуємо член $\overline{A}\cdot\overline{B}$, для чого підставимо у вираз $A\cdot C\vee\overline{B}\cdot C$ значення аргументів $A=0,B=0$ (при цьому $\overline{A}\cdot\overline{B}=\overline{0}\cdot\overline{0}=1$). Таким чином, отримаємо $0\cdot C\vee\overline{0}\cdot C=C$. Оскільки цей вираз не дорівнює тотожно 1, то член $\overline{A}\cdot\overline{B}$ виключати не можна.

Аналогічно випробуємо член $A\cdot C$. Підставляючи у вираз $\overline{A}\cdot\overline{B}\vee\overline{B}\cdot C$ значення *A = 1* і *C=1* (при цьому $A\cdot C=1\cdot 1=1$). Таким чином, цей член, також виключити не можна, оскільки $\overline{1}\cdot\overline{B}\vee\overline{B}\cdot 1=\overline{0}\cdot\overline{B}\vee\overline{B}\cdot 1=\overline{B}$.

Далі випробуємо член $\overline{B}\cdot C$, для чого підставимо у вираз $\overline{A}\cdot\overline{B}\vee A\cdot C$ значення аргументів *B=0* і *C=0* (при цьому $\overline{B}\cdot C=\overline{0}\cdot 1=1\cdot 1=1$). Отримаємо – $\overline{A}\cdot\overline{0}\vee A\cdot 1=\overline{A}\cdot 1\vee A\cdot 1=\overline{A}\vee A=1.$ Отриманий вираз тотожно дорівнює 1, тому член $\overline{B}\cdot C$ можна виключити.

Отже, після всіх викладок отримаємо – функція $f(A,B,C)=\overline{A}\cdot\overline{B}\vee A\cdot C\vee\overline{B}\cdot C$ має одну тупикову форму, яка і є мінімальною $f(A,B,C)=\overline{A}\cdot B\vee A\cdot C.$

Виключення надмірних доданків може бути здійснене різними способами. Тому для тієї самої функції може існувати кілька тупикових форм. Якщо всі їх знайдено, то вибирається мінімальна.

Слід зазначити, що із збільшенням кількості аргументів складність мінімізації шляхом застосування тотожних перетворень зростає. Впоратися зі складними функціями можуть лише досвідчені фахівці в галузі алгебри логіки.

6.2. Метод Квайна та імплікантні матриці

Метод отримання скороченої нормальної диз'юнктивної форми логічної функції називається *методом Квайна.*

При мінімізації за методом Квайна в базисі 1 передбачено, що вихідна функція задана в досконалій нормальній диз'юнктивній формі.

Нагадаємо, що *імпліканта функції* – це логічна функція, яка перетворюється на нуль при наборі аргументів, на яких сама функція також дорівнює нулю.

Тому будь-який *мінтерм* у складі досконалій нормальній диз'юнктивній формі або група *мінтермів*, з'єднаних знаками диз'юнкції, є імплікантами вихідної нормальної диз'юнктивної форми.

Первинна або проста імпліканта функції – це імпліканта типу елементарної кон'юнкції деяких аргументів, причому жодна частина якої вже не є імплікантою цієї функції.

Диз'юнкція простих імплікант, жодну з яких виключити не можна, називається *тупиковою нормальною диз'юнктивною формою функції*. Деякі функції мають кілька тупикових форм. Тупикові форми, що містять найменшу кількість букв, є мінімальними.

Задача мінімізації за методом Квайна зводиться до того, щоб попарно порівняти всі імпліканти, які входять у досконалій нормальній диз'юнктивній формі, з метою виявлення можливості поглинання якогось аргументі, тобто $y \cdot x_i \vee y \cdot \bar{x}_i = y$.

Таким чином, удається знизити ранг термів. Ця процедура проводиться до тих пір, поки не залишиться жодного члена, що допускає поглинання з якимось іншим термом. Терми, які піддалися поглинанню, відзначаються. Терми, які не було відзначено, становлять собою первинні імпліканти [14, 24, 25, 27 та ін.].

Отриманий логічний вираз не завжди виявляється мінімальним. Тому досліджують можливість подальшого спрощення. Для цього складають таблицю, у рядках якої записують знайдені первинні імпліканти, а в стовпцях указують терми вихідного рівняння. Клітини цієї таблиці відзначають у випадку, якщо первинна імпліканта входить до складу якогось терма. Після цього задача спрощення зводиться до того, щоб знайти таку мінімальну кількість первинних імплікант, яка покриває всі стовпці.

У цьому методі використовують операції неповного склеювання $(x \cdot y \vee x \cdot \bar{y} = x \vee x \cdot y \vee x \cdot \bar{y})$ і поглинання.

Теорема Квайна. Якщо в досконалій нормальній диз'юнктивній формі логічної функції провести всі операції неповного склеювання й потім усі операції поглинання, то в результаті виходить скорочена нормальна диз'юнктивна форма цієї функції, тобто диз'юнкція всіх її простих імплікант [14, 24, 25].

Метод Квайна виконується в кілька етапів, і скорочену нормальну диз'юнктивну форму зручно знаходити в такій послідовності:

1) провести в досконалу нормальну диз'юнктивну форму функції всі можливі операції склеювання конституант одиниці. У результаті цього утворюються добутки, які містять (*n-1*) букв (склеюватися можуть лише добутки з однаковою кількістю букв);

2) виконати операцію поглинання;

3) здійснити всі можливі склеювання членів з (*n-1*) буквою;

4) провести поглинання членів з (*n-1*) буквою й знову виконати операцію склеювання членів з кількістю букв, що дорівнює (*n-2*), і т. д.

Як приклад розглянемо процес мінімізації логічної функції трьох аргументів за допомогою метода Квайна. Функцію задана таблицею (див. табл. 6.4).

Таблиця 6.4

Логічна функція трьох аргументів

A	*B*	*C*	*f(A,B,C)*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Представимо функцію в досконалій диз'юнктивній нормальній формі

$$f(A,B,C) = \overline{A} \cdot B \cdot C \vee A \cdot \overline{B} \cdot \overline{C} \vee A \cdot \overline{B} \cdot C .$$

У правій частині отриманого виразу можна виконати лише одне склеювання: другого члена з третім за аргументом *C*. При цьому отримаємо:

$$f(A,B,C) = \overline{A} \cdot B \vee \overline{A} \cdot B \cdot C \vee A \cdot \overline{B} \cdot \overline{C} \vee A \cdot \overline{B} \cdot C$$

Добуток $(\overline{A} \cdot B)$ поглинає члени $(A \cdot \overline{B} \cdot \overline{C})$ *и* $(A \cdot \overline{B} \cdot C)$:

$$f(A,B,C) = A \cdot \overline{B} \vee \overline{A} \cdot B \cdot C.$$

Цей вираз і є *скороченою диз'юнктивною нормальною формою заданої логічної функції*, оскільки подальше застосування склеювання й поглинання неможливе.

Тепер розглянемо процедуру мінімізації більш складної функції з використанням імплікативних матриць.

Припустимо, що треба знайти мінімальну нормальну диз'юнктивну форму логічної функції:

$$f(A,B,C,D) = \overline{A} \cdot B \cdot C \cdot D \vee \overline{A} \cdot \overline{B} \cdot C \cdot D \vee A \cdot \overline{B} \cdot C \cdot D \vee$$
$$\vee A \cdot \overline{B} \cdot \overline{C} \cdot D \vee A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} \vee \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}.$$

Побудуємо для цієї функції *імплікантну матрицю*, яка становить собою таблицю, у вертикальні й горизонтальні входи якої записано всі конституанти одиниці, тобто всі мінтерми й усі прості імпліканти заданої функції відповідно.

Для знаходження простих імплікант ми повинні провести спочатку процедуру неповного склеювання. Щоб швидше знаходити члени, які склеюються один з одним, нагадаємо, що склеюватися можуть лише сусідні члени, тобто такі, у яких кількість аргументів із запереченнями відрізняється на одиницю.

Проведемо операції склеювання в такому порядку:

1) виконаємо всі можливі склеювання 1-го члена з іншими;
2) виконаємо всі можливі склеювання 2-го члена з іншими, крім 1-го;
3) виконаємо всі можливі склеювання 3-го члена з іншими, крім 1-го й 2-го, і т. д.

Запишемо результат:

$1^* - 2^* = \overline{A} \cdot B \cdot C \cdot D \vee \overline{A} \cdot \overline{B} \cdot C \cdot D = \overline{A} \cdot C \cdot D \cdot (\overline{B} \vee B) = \overline{A} \cdot C \cdot D$ (за B);

$2 - 3^* = \overline{B} \cdot C \cdot D$ (за A);

$3 - 4^* = A \cdot \overline{B} \cdot D$ (за C);

$4 - 5^* = A \cdot \overline{B} \cdot \overline{C}$ (за D);

$5 - 6^* = \overline{B} \cdot \overline{C} \cdot \overline{D}$ (за A).

Отримали прості імпліканти. Після процедури неповного склеювання вираз набуде такого вигляду:

$$f(A,B,C,D) = \overline{A} \cdot B \cdot C \cdot D^* + \overline{A} \cdot C \cdot D + \overline{A} \cdot \overline{B} \cdot C \cdot D^* + \overline{B} \cdot C \cdot D +$$
$$+ A \cdot \overline{B} \cdot C \cdot D^* + A \cdot \overline{B} \cdot D + A \cdot \overline{B} \cdot \overline{C} \cdot D^* + A \cdot \overline{B} \cdot \overline{C} +$$
$$A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}^* + \overline{B} \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}^*.$$

Зірочками позначено ті члени, які поглинаються добутками, що утворилися після склеювання.

Далі проводять операцію поглинання. Для кожної імпліканти знаходять конституанти одиниці, тобто мінтерми, які нею поглинаються, тобто ті конституанти, власною частиною яких є ця імпліканта, наприклад, імпліканта $\overline{A} \cdot C \cdot D$ поглинає конституанти $A \cdot B \cdot C \cdot D$, $\overline{A} \cdot \overline{B} \cdot C \cdot D$, імпліканта $\overline{B} \cdot C \cdot D$ – конституанти $\overline{A} \cdot \overline{B} \cdot C \cdot D$, $A \cdot \overline{B} \cdot C \cdot D$ і т. д.

$$\overline{A}\cdot B\cdot C\cdot D^{*}+\overline{A}\cdot C\cdot D=\overline{A}\cdot C\cdot D\cdot(1+B)=\overline{A}\cdot C\cdot D,$$

$$\overline{A}\cdot\overline{B}\cdot C\cdot D^{*}+A\cdot\overline{B}\cdot C\cdot D^{*}+\overline{B}\cdot C\cdot D=\overline{B}\cdot C\cdot D\cdot(1+A+\overline{A})=\overline{B}\cdot C\cdot D,$$

$$A\cdot\overline{B}\cdot\overline{C}\cdot D^{*}+A\cdot\overline{B}\cdot\overline{C}=A\cdot\overline{B}\cdot\overline{C}\cdot(1+D)=A\cdot\overline{B}\cdot\overline{C},$$

$$A\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}^{*}+\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}^{*}+\overline{B}\cdot\overline{C}\cdot\overline{D}=B\cdot C\cdot D\cdot(1+A+\overline{A})=\overline{B}\cdot\overline{C}\cdot\overline{D}.$$

Тепер розглянемо принцип побудови й використання імплікантної матриці на прикладі матриці, наведеної в табл. 6.5.

Таблиця 6.5

Імплікантна матриця

№	Прості імпліканти	Конституанти одиниці					
		$\overline{A}\cdot B\cdot C\cdot D$	$\overline{A}\cdot\overline{B}\cdot C\cdot D$	$A\cdot\overline{B}\cdot C\cdot D$	$\overline{A}\cdot B\cdot\overline{C}\cdot D$	$A\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}$	$\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}$
		1	2	3	4	5	6
1	$\overline{A}\cdot C\cdot D$	*	*				
2	$\overline{B}\cdot C\cdot D$		*	*			
3	$A\cdot\overline{B}\cdot D$			*	*		
4	$A\cdot\overline{B}\cdot\overline{C}$				*	*	
5	$\overline{B}\cdot\overline{C}\cdot\overline{D}$					*	*

Клітини імплікантної матриці, утворені перетином рядків з імплікантами й колонок з поглинутими ними конституантами, позначимо знаком „*”.

Щоб отримати мінімальну нормальну диз’юнктивну форму заданої функції, достатньо знайти мінімальне число імплікант, які спільно накривають зірочками колонки імплікантної матриці.

З таблиці випливає, що в мінімальну форму обов’язково повинні ввійти імпліканти $\overline{A}\cdot C\cdot D$ і $\overline{B}\cdot\overline{C}\cdot\overline{D}$, оскільки лише вони накривають зірочками й шосту колонки таблиці.

Крім того, імпліканта $\overline{A}\cdot C\cdot D$ накриває другу, а імпліканта $\overline{B}\cdot\overline{C}\cdot\overline{D}$ – п’яту колонки. Тому залишається накрити лише третю й четверту колонки таблиці. Для цього можна вибрати пари імплікант $\overline{B}\cdot C\cdot D$ і $A\cdot\overline{B}\cdot D$; $A\cdot\overline{B}\cdot D$ і $A\cdot\overline{B}\cdot\overline{C}$ або одну імпліканту $A\cdot\overline{B}\cdot D$. Якщо вибрати вказані вище пари імплікант, члени $\overline{B}\cdot C\cdot D$ і $A\cdot\overline{B}\cdot\overline{C}$ виявляються зайвими, оскільки імпліканта $A\cdot\overline{B}\cdot D$ одна накриває третю й четверту колонки таблиці. Таким чином, вибравши імпліканту $A\cdot\overline{B}\cdot D$, отримаємо мінімальну нормальну диз’юнктивну форму заданої функції:

$$f(A,B,C,D)=\overline{A}\cdot C\cdot D\vee A\cdot\overline{B}\cdot D\vee\overline{B}\cdot\overline{C}\cdot\overline{D}.$$

На підставі вищевикладеного сформулюємо алгоритм отримання мінімальних нормальих диз'юнктивних форм заданої логічної функції:

1) логічну функцію представляють у досконалій нормальній диз'юнктивій формі, застосовуючи або запис „за одиницями" функції, якщо функцію задано таблицею, або операції розгортання, правила де Моргана та інші формули алгебри логіки, якщо функцію задано в довільній аналітичній формі;

2) в отриманій досконалій нормальній диз'юнктивій формі проводять усі операції неповного склеювання й поглинання. У результаті виходить скорочена нормальна диз'юнктива форма заданої функції;

3) знаходять мінімальні нормальні диз'юнктиві форми за імплікантною матрицею. Якщо кількість членів у скороченій нормальній диз'юнктивій формі невелика, то можна знайти тупикові форми методом випробування членів і вибрати серед них мінімальні.

Слід зазначити, що в окремих випадках мінімальна нормальна диз'юнктивна форма збігається зі скороченою.

Для того щоб знайти вираз заданої логічної функції, найбільш зручний для синтезу логічної схеми, слід, крім мінімальні нормальні диз'юнктиві форми функції, отримати також її мінімальну нормальну кон'юнктивну форму і вибрати з них ту, при технічній реалізації якої потрібна найменша кількість логічних елементів.

Розглянемо ще один алгоритм отримання мінімальної нормальної диз'юнктивної форми функції:

1) тим або іншим способом формують досконалу нормальну кон'юнктивну форму функції;
2) знаходять скорочену нормальну кон'юнктивну форму;
3) визначають тупикові нормальні кон'юнктивні форми і за ними знаходять мінімальні нормальні кон'юнктивні форми заданої функції.

Якщо функцію задано таблицею, то досконала нормальна кон'юнктивна форма формується „за нулями" функції. В усіх же подальших процедурах мінімізації має бути враховано, що в цьому випадку використовуються макстерми, тобто елементарні суми, а не елементарні добутки. Імпліканта, у тому числі й проста, – також елементарна сума.

Операції неповного склеювання й поглинання для кон'юнктивної форми визначаються, відповідно, такими співвідношеннями [14,15,24,27]:

$$(x + y)\cdot(x + \bar{y}) = x\cdot(x + y)\cdot(x + \bar{y}),$$
$$x\cdot(x + y) = x,$$

а формули розгортання мають вигляд:

$$x = (x + y) \cdot (x + \bar{y}),$$
$$(x + y) = (x + y + z) \cdot (x + y + \bar{z}).$$

Випробовуючи члени скороченої нормальної кон'юнктивної форми, виключають випробовуваний член й у вираз, що залишився, підставляють такі значення аргументів, які обертають виключений член у нуль. Якщо при цьому вираз, що залишився, також дорівнюватиме нулю, то випробовуваний член є зайвим. Таким чином, отримують тупикові нормальні кон'юнктивні форми, з яких вибирають мінімальні нормальні кон'юнктивні форми. Якщо отримана скорочена нормальна кон'юнктивна форма містить велику кількість членів, то мінімальні нормальні кон'юнктивні форми отримують за допомогою імплікантних матриць за методикою, аналогічною до використовуваної для отримання мінімальної нормальної диз'юнктивної форми.

Можна використати й інший спосіб отримання мінімальні нормальні кон'юнктивні форми. Попередньо знаходять мінімальні нормальні диз'юнктивної форми функції. Потім від отриманої мінімальної нормальної диз'юнктивної форми береться заперечення, і після перетворення за формулами де Моргана отримують мінімальної нормальної кон'юнктивної форми заданої функції.

6.3. Метод Карно

Прагнення до алгоритмізації пошуку елементарних добутків зумовило розробку табличних методів мінімізації логічних функцій. Одним з них є метод, що ґрунтується на використанні карт Карно.

Карти Карно (їх різновидом є карти Вейча, побудовані як розгортки куба на площині), є *графічним представленням таблиць істинності*. Тому їх будують або за таблицею істинності аналізованої функції, або ж за її досконалій нормальній диз'юнктивній формі.

Як вже відмічалось вище, кожен рядок таблиці істинності, для якої функція дорівнює одиниці, відповідає *мінтерму* функції, представленої в досконалій нормальній диз'юнктивній формі. Рядок, для якого функція дорівнює нулю, є *макстермом* функції, представленої в досконалій нормальній кон'юнктивній формі.

Карта Карно становить собою прямокутник, розбитий на квадрати, кількість яких дорівнює загальній кількості наборів для цієї функції n аргументів, тобто воно дорівнює 2^n. Так, для функції чотирьох аргументів квадратів буде 16, для п'яти аргументів – 32 і т. д. Кожен квадрат відповідає певному набору або терму, причому набори розташовано так,

щоб *сусідні набори або терми й за горизонталлю, і за вертикаллю відрізнялися б лише значенням одного аргументу: в одному квадраті він з інверсією, а в іншому, сусідньому – без*. Функцію в досконалій нормальній диз'юнктивній формі наносять на карту, відзначаючи, наприклад, знаком 1 квадрати, що відповідають тим наборам, на яких функція дорівнює одиниці, тобто в досконалій нормальній диз'юнктивній формі функції є відповідний *мінтерм*. Інші квадрати відзначають знаком 0. Іноді в куті квадрата ставлять номер набору. Такий спосіб використовують, якщо функцію задано числовим способом, але він незручний для процедури мінімізації.

Розглянемо приклади побудови карт Карно за таблицями істинності для двох аргументів.

Нижче наведено таблицю істинності (див. табл. 6.6) і структуру карти Карно (див. табл. 6.7) для логічної функції двох аргументів.

Таблиця 6.6

Таблиця істинності логічної функції двох аргументів

A	*B*	*f(A,B)*
0	0	*f*(0,0)
0	1	*f*(0,1)
1	0	*f*(1,0)
1	1	*f*(1,1)

Таблиця 6.7

Структура карти Карно функції двох аргументів

	B = 0	*B = 1*
A = 0	*f*(0,0)	*f*(0,1)
A = 1	*f*(1,0)	*f*(1,1)

У табл. 6.8 наведено таблицю істинності й структуру карти Карно для логічної функції трьох аргументів; у табл. 6.9 – карту Карно для логічної функції чотирьох аргументів.

Таким чином, карту Карно можна оформляти будь-яким прийнятним способом. При цьому слід пам'ятати, що кожен квадрат має відповідати певному набору або терму.

Треба врахувати, що квадрати, розташовані на протилежних кінцях кожного рядка або стовпця, також є сусідніми.

Перерахуємо послідовність дій, виконуваних для мінімізації функції за методом Карно:

1) вихідна функція, яка підлягає мінімізації, має бути представлена в нормальній диз'юнктивій форм. Потім її треба представити в досконалій нормальній диз'юнктивій формі. Або ж складають таблицю істинності мінімізованої функції. Як уже було відзначено, між рядками таблиці істинності й клітинами (гніздами) на карті Карно існує взаємно однозначна відповідність. Коли карта Карно складається за досконалою нормальною диз'юнктивою формою мінімізованою функцією, то очевидно, що кожний аргумент без заперечення заміняється її значенням 1, а із запереченням – 0;

Таблиця 6.8

Таблиці істинності й структура карти Карно для логічної функції трьох аргументів

Таблиця істинності логічної функції трьох аргумент				**Карта Карно логічної функції трьох аргументів**				
Аргумент			**Функція**		*BC*			
A	*B*	*C*	*f(A,B,C)*		**00**	**01**	**11**	**10**
0	**0**	**0**	*f*(0,0,0)	*A*	*f*(0,00)	*f*(0,01)	*f*(0,11)	*f*(0,10)
0	**0**	**1**	*f*(0,0,1)					
0	**1**	**0**	*f*(0,1,0)					
0	**1**	**1**	*f*(0,1,1)					
1	**0**	**0**	*f*(1,0,0)		*f*(1,00)	*f*(1,01)	*f*(1,11)	*f*(1,10)
1	**0**	**1**	*f*(1,0,1)					
1	**1**	**0**	*f*(1,1,0)					
1	**1**	**1**	*f*(1,1,1)					

Таблиця 6.9

Структура карти Карно для логічної функції чотирьох аргументів

f(A,B,C,D)		*CD*			
		00	**01**	**11**	**10**
AB	**00**	*f*(0,0,0,0)	*f*(0,0,0,1)	*f*(0,0,1,1)	*f*(0,0,1,0)
	01	*f*(0,1,0,0)	*f*(0,1,0,1)	*f*(0,1,1,1)	*f*(0,1,1,0)
	11	*f*(1,1,0,0)	*f*(1,1,0,1)	*f*(1,1,1,1)	*f*(1,1,1,0)
	10	*f*(1,0,0,0)	*f*(1,0,0,1)	*f*(1,0,1,1)	*f*(1,0,1,0)

2) потім будують карту Карно за принципом, описаним вище. Представимо систему координат, у якій, наприклад, для функції двох

аргументів на горизонтальній осі відкладаються значення одного аргументу, а на вертикалі – іншого. На перетині відповідних координат отримуємо гніздо, куди записуємо значення функції (0 або 1), що відповідає цьому набору. Якщо функцію представлено в досконалій нормальній диз'юнктивній формі, то в гнізді, що відповідає існуючому *мінтерму,* записуємо 1, а в гнізді неіснуючого *мінтерма* – 0;

3) після цього суміжні гнізда, у яких записано одиниці, об'єднують у групи в такий спосіб: об'єднують обов'язково парну кількість сусідніх гнізд з одиницями, і за вертикаллю, і за горизонталлю. Причому, кожне гніздо з 1 може потрапити одночасно в дві групи, отримані внаслідок об'єднання одиниць і за вертикаллю, і за горизонталлю;

4) кожній групі ставлять у відповідність новий *мінтерм* для зображення вихідної функції у формі мінімальної нормальної диз'юнктивної формі;

5) зображення кожного нового *мінтерма* формують за таким алгоритмом:

а) аргумент, який в кожному гнізді утвореної групи має значення тільки 0, зображують її інверсією;

б) аргумент, який в кожному гнізді утвореної групи має значення тільки 1, зображують без інверсії;

в) аргумент, який в межах утвореної групи змінює своє значення, не зображують, тобто відкидають.

Таким чином, карту Карно можна розглядати як графічне представлення сукупності всіх (існуючих і неіснуючих) *мінтермів* функції в досконалій нормальній диз'юнктивній формі певної кількості логічних аргументів.

Розглянемо таблиці 6.8 і 6.9. Карта Карно для функції трьох аргументів має $2^n=2^3=8$ гнізд, а для чотирьох аргументів – $2^4=16$ гнізд, виділених у таблиці кольором.

Карту розмічено системою координат, що відповідають значенню вхідних аргументів логічної функції, наприклад, верхній рядок карти для функції трьох аргументів відповідає нульовому значенню аргументів *A,* а нижній – її одиничному значенню. Кожен стовпець цієї карти характеризується значеннями двох аргументів – *B* і *C*.

Комбінація цифр, якими відзначено кожен стовпець, показує, для яких значень аргументів *B* і *C* обчислюють функцію, розміщувану в гніздах цього стовпця. Так, для випадку карти Карно функції чотирьох аргументів функція, розташована в гніздах стовпця з координатами 01, обчислюється при значеннях аргументів *C=0* і *D=1*. Функція, розташована в гнізді на перетині цього стовпця й рядка з координатами 11, визначається при наборі вхідних аргументів *A=1, B=1, C=0, D=1.*

Якщо на вказаному наборі аргументів функція дорівнює одиниці, то її досконала нормальна диз'юнктивна форма обов'язково містить елементарний добуток $A \cdot B \cdot \overline{C} \cdot D$, який приймає на цьому наборі одиничне значення. Таким чином, гнізда карти Карно, які представляють функцію, містять стільки одиниць, скільки елементарних добутків міститься в її досконалій нормальній диз'юнктивій формі, причому кожній одиниці відповідає один з елементарних добутків.

Координати рядків і стовпців у карті Карно слідують не в природному порядку зростання двійкових кодів, а в порядку 00, 01, 11, 10. Порядок слідування наборів змінено для того, щоб сусідні набори були сусідніми в геометричному розумінні.

Розглянемо табл. 6.10, у якій уміщено таблицю істинності й карту Карно функції $y = f(A,B,C) = \overline{A} \vee \overline{B} \cdot C$. Гнізда, у яких функція приймає значення, що дорівнюють одиниці, заповнено одиницями. В інші гнізда записано нулі. Процес мінімізації полягає у формуванні прямокутників, що містять по 2^k гнізда, де k – ціле число. У прямокутники об'єднано сусідні гнізда, які відповідають сусіднім елементарним добуткам. Наприклад, об'єднано гнізда з координатами 001 і 101. При об'єднанні цих гнізд утворився прямокутник, у якому аргумент A змінює своє значення. Отже, вона зникає при склеюванні відповідних елементарних добутків $\overline{A} \cdot \overline{B} \cdot C$ і $A \cdot \overline{B} \cdot C$. Гнізда, розташовані в першому рядку карти Карно, містять одиниці і є сусідніми. Тому всі їх об'єднано в прямокутник, що містить 2^2=4 гнізда.

Сукупність прямокутників, які покривають усі одиниці, називають *покриттям*. Слід відзначити, що одне й те саме гніздо (наприклад, з координатами 001) може покриватися два й кілька разів.

Аргументи B і C у межах прямокутника змінюють своє значення, отже, вони зникнуть з результуючого елементарного добутку. Аргумент A залишається незмінним й дорівнює нулю. Таким чином, елементарний добуток, отриманий у результаті об'єднання гнізд першого рядка, містить лише один елемент $\overline{A}$. Це, зокрема, випливає з того, що чотирьом гніздам першого рядка відповідає сума чотирьох елементарних добутків:

$$\overline{A} \cdot \overline{B} \cdot \overline{C} \vee \overline{A} \cdot \overline{B} \cdot C \vee \overline{A} \cdot B \cdot C \vee \overline{A} \cdot B \cdot \overline{C} =$$
$$= \overline{A} \cdot \overline{B} \cdot (\overline{C} \vee C) \vee \overline{A} \cdot B \cdot (C \vee \overline{C}) =$$
$$= \overline{A} \cdot \overline{B} \vee \overline{A} \cdot B = \overline{A} \cdot (B \vee \overline{B}) = \overline{A}.$$

Таблиця 6.10

Таблиця істинності та структура карти Карно для логічної функції трьох аргументів $y = f(A,B,C) = \overline{A} \vee \overline{B} \cdot C.$

Таблиця істинності логічної функції трьох аргументів				**Карта Карно логічної функції трьох аргументів** $y = f(A,B,C) = \overline{A} \vee \overline{B} \cdot C.$				
Аргументи			**Функція**		*BC*			
A	*B*	*C*	*f(A,B,C)*		**00**	**01**	**11**	**10**
0	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**1**
0	**0**	**1**	**1**	*A*				
0	**1**	**0**	**1**					
0	**1**	**1**	**1**	**1**				
1	**0**	**0**	**0**		**0**	**1**	**0**	**0**
1	**0**	**1**	**1**					
1	**1**	**0**	**0**					
1	**1**	**1**	**0**					

Таким чином, формула, яку отримуємо в результаті мінімізації логічної функції за допомогою карт Карно, містить суму стількох елементарних добутків, скільки прямокутників є в покритті. Чім більше гнізд у прямокутнику, тим менше аргументів міститься у відповідному йому елементарному добутку. Наприклад, для карти Карно, зображеної на рис. 6.1 а, прямокутнику, який містить чотири гнізда, відповідає елементарний добуток $\overline{C} \cdot D$ двох аргументів, а квадрату, що складається лише з одного гнізда, – елементарний добуток $\overline{A} \cdot B \cdot C \cdot \overline{D}$, що містить усі чотири аргументи. Функція, що відповідає покриттю, показаному на рис. 6.1 а, має вигляд

$$y = f(A,B,C,D) = \overline{C} \cdot D \vee \overline{A} \cdot B \cdot C \cdot \overline{D}.$$

Незважаючи на те, що карти Карно зображують на площині, сусідство квадратів установлюють на поверхні тора. Верхня й нижня межі карти Карно ніби склеюються, утворюючи поверхню циліндра. При склеюванні бокових меж утворюється поверхня у вигляді тора. Таким чином, гнізда з координатами 1011 і 0011, зображені на рис. 6.1 б, є

сусідніми у об'єднуються в прямокутник. Указаним гніздам відповідає сума елементарних добутків

$$A \cdot \overline{B} \cdot C \cdot D \vee \overline{A} \cdot \overline{B} \cdot C \cdot D = (A \vee \overline{A}) \cdot \overline{B} \cdot C \cdot D = \overline{B} \cdot C \cdot D.$$

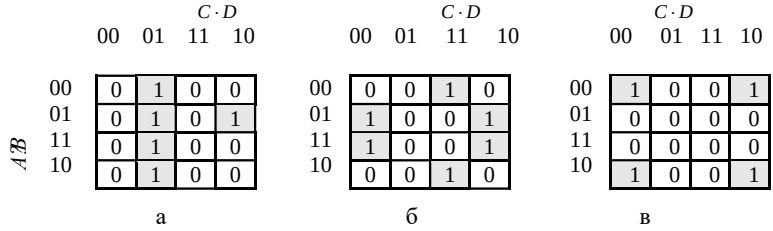


Рис. 6.1. Карти Карно функції чотирьох аргументів

Аналогічно об'єднуються й інші чотири одиничні гнізда. У результаті їх об'єднання отримуємо елементарний добуток $B \cdot \overline{D}$. Остаточно функція, що відповідає покриттю, зображеному на рис 6.3 б, має вигляд $f(A,B,C,D) = B \cdot \overline{D} \vee \overline{B} \cdot C \cdot D.$

Розглянемо ще один приклад. Карта Карно, наведена на рис. 6.1 в, містить одиничні гнізда, розташовані в кутах. Усі чотири гнізда є сусідніми й після об'єднання дадуть елементарний добуток $\overline{B} \cdot \overline{D}$.

При формуванні на базі карти Карно нових мінтермів для представлення функції в мінімальній нормальній диз'юнктивній формі або ж у мінімальній нормальній кон'юнктивній формі значення відповідних аргументів, що дорівнюють 1, заміняються зображенням аргументів без заперечення, а при значеннях, що дорівнюють 0, – із запереченням.

Розглянемо процес мінімізації на прикладі функції, заданої таким логічним рівнянням:

$$f(A,B,C,D) = B \cdot C \cdot D + \overline{A} \cdot B \cdot D + \overline{B} \cdot C \cdot D + A \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{C} \cdot D + \overline{B} \cdot \overline{C} \cdot D + \overline{A} \cdot B \cdot C + \overline{A} \cdot \overline{B} \cdot \overline{C}.$$

Представимо цю функцію в досконалій диз'юнктивній нормальній формі:

$$\begin{aligned} f = {} & B \cdot C \cdot D \cdot (A + \overline{A}) + \overline{A} \cdot B \cdot D \cdot (C + \overline{C}) + B \cdot C \cdot D \cdot (A + \overline{A}) + \\ & + A \cdot \overline{B} \cdot \overline{C} \cdot (D + \overline{D}) + A \cdot \overline{C} \cdot D \cdot (B + \overline{B}) + \\ & + \overline{B} \cdot \overline{C} \cdot D \cdot (A + \overline{A}) + \overline{A} \cdot B \cdot C \cdot (D + \overline{D}) + \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot (D + \overline{D}) = \\ & A \cdot B \cdot C \cdot D + \overline{A} \cdot B \cdot C \cdot D + \overline{A} \cdot B \cdot \overline{C} \cdot D + \\ & + A \cdot \overline{B} \cdot C \cdot D + \overline{A} \cdot \overline{B} \cdot C \cdot D + \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot D + A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} + A \cdot B \cdot \overline{C} \cdot D + \\ & \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot B \cdot C \cdot \overline{D} + \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot D. \end{aligned}$$

Нижче зображено карту Карно, що відповідає аналізованій функції (див. рис. 6.2).

Мінтерми функції утворюють у карті чотири групи: $\overline{A}\cdot\overline{B}\cdot\overline{C}$, $A\cdot\overline{B}\cdot\overline{C}$, D, $(\overline{A}\cdot B\cdot C)$.

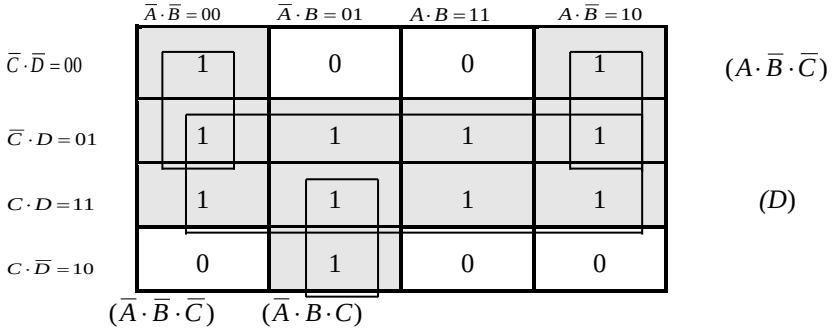


Рис. 6.2. Карта Карно

Таким чином, отримаємо

$$f=\overline{A}\cdot\overline{B}\cdot\overline{C}+A\cdot\overline{B}\cdot\overline{C}+\overline{A}\cdot B\cdot C+D=\overline{B}\cdot\overline{C}\cdot(A+\overline{A})+\overline{A}\cdot B\cdot C+D=$$
$$=\overline{B}\cdot\overline{C}+\overline{A}\cdot B\cdot C+D.$$

Разом з тим, треба мати на увазі, що в загальному випадку функція може мати кілька мінімальних форм.

Хоча зазвичай карти Карно для функцій трьох і чотирьох аргументів зображаються на площині, але, як було відзначено, з огляду формування прямокутних груп карту треба вважати тривимірною.

Для карт Карно функції трьох аргументів карту слід розглядати як циліндр зі склеєними правим і лівим краями. Оскільки прямокутні групи формуються на такому циліндрі, на плоскому рисунку та чи інша група може виявитися розірваною.

Розглянемо раніше наведений приклад, але тепер представимо карту у вигляді циліндра. Тоді в нас вийде три групи мінтермів замість чотирьох (див. рис. 6.3).

Мінтерми функції утворюють на карті три групи: $\overline{B}\cdot\overline{C}$ – мінтерм, що відповідає групі з чотирьох верхніх одиниць, другий мінтерм – D; третій – $\overline{A}\cdot B\cdot C$. Отже отримаємо $f=\overline{B}\cdot\overline{C}+\overline{A}\cdot B\cdot C+D$.

При отриманні мінімальної форми для досконалій мінімальній кон'юнктивній форми функцію задано термами, які приймають нульове значення на відповідних наборах, тобто макстермами. Тому в клітках

мінімізуючих карт пишуть нулі, які групують за описаною раніше методикою.

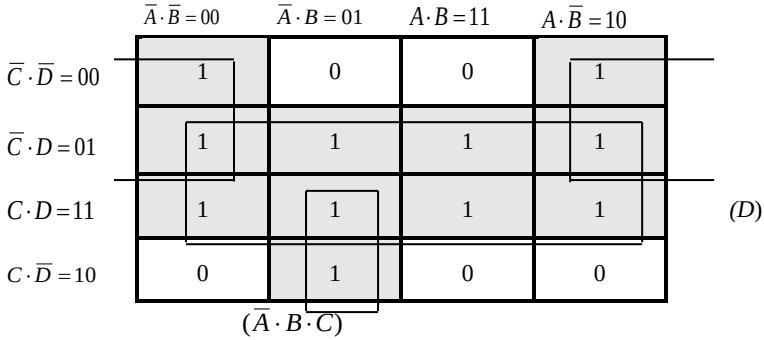


Рис. 6.3. Карта Карно

За наявності сформованої мінімальної нормальній диз'юнктивній формі функції мінімальну мінімальну кон'юнктиву форму можна також знайти так. Беруть заперечення від знайденої мінімальної нормальної диз'юнктивної формі і здійснюють перетворення за формулою де Моргана. Таким чином, виходить мінімальна кон'юнктива форма функції. Наприклад, якщо мінімальні нормальні диз'юнктивні форми функції дорівнює $f(A,B,C)=\overline{\overline{A}\cdot B}+\overline{A\cdot\overline{C}}$, то, провівши заперечення й застосувавши теорему де Моргана, отримаємо мінімальну кон'юнктиву форму – $f(A,B,C)=\overline{\overline{A}\cdot B}+\overline{A\cdot\overline{C}}=(A+\overline{B})(\overline{A}+C)$.

Розглянуті приклади дозволяють алгоритмізувати процес мінімізації логічних функцій з використанням карт Карно. Таблицю з алгоритмом представлено нижче (див. табл. 6.11).

Таблиця 6.11

Послідовність дій при мінімізації логічної функції за допомогою карт Карно

№ кроку	Дія
1	Намалювати таблицю для *n* аргументів. Розмітити її сторони.
2	Гнізда таблиці, що відповідають наборам аргументів, які обертають функцію в одиницю, заповнити одиницями (1), інші гнізда – нулями (0).
3	Вибрати найкраще покриття таблиці правильними прямокутниками. Найкращим вважається покриття, утворене мінімальною кількістю прямокутників. Якщо таких варіантів кілька, то з них обирають той, який дає максимальну сумарну площу прямокутників.

Якість мінімізації оцінюється *коефіцієнтом покриття* [14,15,33]:

$$k = m / s,$$

де m – загальна кількість прямокутників;
s – сумарна площа прямокутників.

Чим менше значення k, тим краще покриття.

Наприклад, для покриттів, зображених на рис. 6.1 (а, б), коефіцієнти покриття відповідно дорівнюють: а) - $k = 2/5$; б) $k = 2/6 = 1/3$.

6.4. Неповністю визначені логічні функції

Розглядаючи двійково-десяткові коди, відмічалось, що десяткові цифри представляються в них чотирма двійковими розрядами. З усіх шістнадцяти кодових комбінацій використовують лише десять, а інші комбінації заборонені й виникати не можуть. Якщо кожному розряду поставити у відповідність двійковий аргумент то для двійково-десяткових кодів отримаємо шість заборонених комбінацій аргументів, які представлено в табл. 6.12.

Таблиця 6.12

Набори аргументів при двійково-десятковому кодуванні

Цифра	Аргументи				Набір
	A	*B*	*C*	*D*	
0	0	0	0	0	Дозволений
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	
-	1	0	1	0	Заборонений
-	1	0	1	1	
-	1	1	0	0	
-	1	1	0	1	
-	1	1	1	0	
-	1	1	1	1	

Якщо яка-небудь функція має заборонені набори аргументів, то її значення на вказаних наборах не визначені й у таблиці істинності позначаються зірочкою – *. Наприклад, у табл. 6.13 представлено функцію, яка має три заборонених набори аргументів. Двійкові функції, значення яких визначено не для всіх наборів вхідних аргументів, називаються *неповністю визначеними*.

На карті Карно гнізда, що відповідають забороненим наборам аргументів, також позначено знаком *, як показано на рис. 6.4 а (для розглянутих функцій).

Таблиця 6.13

Неповністю визначена функція

Аргументи			**Функція**
A	*B*	*C*	
0	**0**	**0**	*****
0	**0**	**1**	**1**
0	**1**	**0**	*****
0	**1**	**1**	**1**
1	**0**	**0**	**0**
1	**0**	**1**	*****
1	**1**	**0**	**0**
1	**1**	**1**	**1**

При мінімізації неповністю визначеної функції її слід довизначити, тобто невизначені значення гнізд карти Карно довільно замінити одиницями й нулями. На рис. 6.4 б, показано функцію f_1, все невизначені значення якої доповнено одиницями.

Мінімізована функція f_1 має формулу $f(A,B,C) = \overline{A} \vee C$, яка показує, що довизначена функція f_1 взагалі не залежить від значення аргументу *B*. Якщо ж крайні гнізда верхнього рядка карти Карно доповнити не одиницями, а нулями, як це показано на рис. 6.4 в, то отримаємо функцію f_2, відмінну від f_1: $f_2(A,B,C) = C$, тобто мінімальна форма f_2 простіша, ніж f_1.

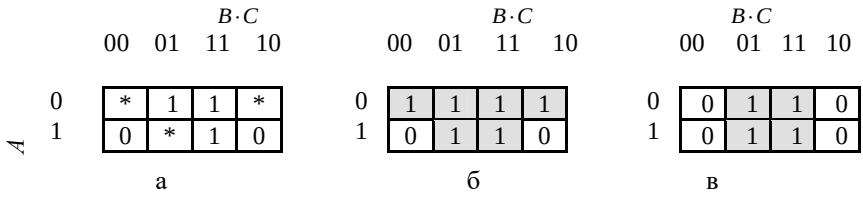


Рис. 6.4 Карти Карно для неповністю визначених функцій

Розглянуті вище приклади ілюструють можливість спрощення формули неповністю визначеної функції при її відповідному довизначенні. Якщо функція має h заборонених наборів аргументів, то може бути 2^h варіантів розв'язання задачі довизначення.

На завершення відмітимо, що на сьогодні розроблено багато програм (Electronics Workbench, Multisim, Maple та ін.), які дозволяють здійснювати процедуру мінімізації логічних функцій різними методами (див. п. 7.5). Однак зі зростанням кількості аргументів логічних функцій експоненційно зростає і складність мінімізації. На практиці складні вирази логічних функцій, які погано оптимізуються, трапляються нечасто, а основні закони тотожних перетворень дають можливість скоротити складність більшості виразів без особливих зусиль.

Запитання для самоперевірки

1. Які Ви знаєте методи мінімізації логічних функцій?
2. Мінімізувати за допомогою карт Карно такі логічні функції:

$$f_1(A,B) = A \cdot \overline{B} \vee A \cdot B;$$
$$f_2(A,B,C) = \overline{A} \cdot \overline{B} \cdot \overline{C} \vee A \cdot \overline{B} \cdot \overline{C} \vee \overline{A} \cdot B \cdot \overline{C} \vee A \cdot B \cdot C.$$

3. Скільки гнізд містить карта Карно для логічної функції п'яти аргументів?
4. Яким критерієм оцінюється якість мінімізації логічної функції?
5. Що таке карта Карно?
6. Сформулюйте теорему Квайна.
7. Які закони алгебри логіки використовують для мінімізації логічних функцій?

Література для самостійної підготовки за темою:

7, 8, 9, 14, 15,22-25, 27, 30, 31.

Розділ 7. МЕТОДИ АНАЛІЗУ Й СИНТЕЗУ ЛОГІЧНИХ ЕЛЕКТРОННИХ СХЕМ

7.1. Логічні оператори електронних схем або ланцюгів

Логічною схемою називається сукупність логічних електронних елементів, з'єднаних між собою так, щоб виконувався заданий закон функціонування схеми, тобто виконувалася задана логічна функція.

За залежністю вихідного сигналу від вхідного всі електронні логічні схеми можна умовно розподілити на:

- *схеми першого роду*, тобто *комбінаційні схеми*, вихідний сигнал яких залежить лише від стану вхідних сигналів у кожний момент часу (безінерційні схеми);
- *схеми другого роду* або накопичувальні схеми, які містять елементи з пам'яттю, вихідний сигнал яких залежить і від вхідних сигналів, і від стану схеми в попередні моменти часу (схеми з пам'яттю).

За кількістю входів і виходів схеми бувають:

- з одним входом і одним виходом;
- з кількома входами й одним виходом;
- з одним входом й кількома виходами;
- з кількома входами й виходами.

За способом здійснення синхронізації схеми бувають із *зовнішньою синхронізацією (синхронні автомати)* та *з внутрішньою синхронізацією* (асинхронні автомати є їх окремим випадком).

Практично будь-який цифровий автомат складається з комбінації схем першого й другого родів різної складності. Таким чином, основою будь-якого цифрового автомата, що обробляє цифрову інформацію, є електронні елементи двох типів: *логічні* або *комбінаційні* й *запам'ятовуючі*. Логічні елементи виконують найпростіші логічні операції над цифровою інформацією, а запам'ятовуючі слугують для її зберігання.

7.1.1. Найпростіші цифрові елементи

Технічним аналогом булевої функції в обчислювальній техніці є так звана *комбінаційна схема.* На вхід комбінаційної схеми надходять, а з

виходу знімаються електричні імпульсні сигнали у вигляді одного з рівнів напруги, що відповідає значенням логічного 0 і логічної 1.

Наприклад, розглянемо схему, яка має m входів і n виходів (рис. 7.1). На її входи можуть бути подані набори значень вхідних сигналів X_i $\{0,1\}$, $i = \overline{1,m}$, а на виходах формуються вихідні сигнали $Y_j \in \{0,1\}$, $j = \overline{1,n}$.

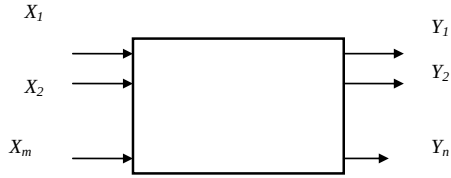


Рис. 7.1. Комбінаційна схема

Схема називається *комбінаційною*, якщо кожну з n функцій її виходів $Y_1, Y_2, ..., Y_n$ можна представити як булеву функцію вхідних сигналів $X_1, X_2, ..., X_m$.

Комбінаційна схема описується за допомогою системи рівнянь [14, 24]:

$$Y_1 = F_1(X_1, X_2, ..., X_m);$$
$$Y_2 = F_2(X_1, X_2, ..., X_m);$$
$$.....$$
$$Y_n = F_n(X_1, X_2, ..., X_m),$$

де F_i – булева функція.

Як випливає з визначення комбінаційної схеми, значення вихідних сигналів Y_j у довільний момент часу однозначно визначаються значеннями вхідних сигналів X_i.

Структурно комбінаційна схема може бути представлена як сукупність елементарних логічних схем – логічних елементів (ЛЕ). Логічні елементи виконують над вхідними аргументами (сигналами) елементарні логічні операції типу **„І-НЕ”, „І”, „АБО”, „АБО-НЕ”** і т. ін.

Кількість входів логічного елемента відповідає кількості аргументів відтворюваної ним булевої функції. Графічне зображення комбінаційної схеми, при якому показано зв’язки між різними елементами, а самі елементи представлено умовними позначеннями, називається *функціональною схемою.*

Під час розробки комбінаційних схем доводиться розв’язувати задачі аналізу й синтезу.

Задача аналізу полягає у визначенні статичних та динамічних властивостей комбінаційної схеми. *У статиці* визначаються булеві функції,

реалізовувані комбінаційною схемою за відомою їй структурою. *У динаміці* розглядається здатність надійного функціонування схеми в перехідних процесах при зміні значень сигналів на входах схеми, тобто визначається наявність на виходах схеми можливих завадних імпульсних сигналів, які не випливають безпосередньо з виразів для булевих функцій, реалізовуваних схемою.

Задача синтезу полягає в побудові із заданого набору логічних елементів комбінаційної схеми, яка реалізує задану систему булевих функцій.

Розв'язання задачі синтезу не є однозначним, можна припустити різні варіанти комбінаційних схем, які реалізують одну й ту саму систему булевих функцій, але відрізняються за тими або іншими параметрами. Розробник комбінаційних схем з цієї множини варіантів вибирає один, виходячи з додаткових критеріїв: мінімальної кількості логічних елементів, необхідних для реалізації схеми, максимальної швидкодії й т. ін. Існують різні методи синтезу комбінаційних схем, серед яких найбільше розроблено канонічний метод.

Елементи, які реалізують найпростіші логічні функції, схематично представляють у вигляді прямокутників, на полі яких зображують символ, що позначає функцію, виконувану цим елементом. Вхідні сигнали прийнято зображувати зліва, а вихідні – справа. Вважається, що передача інформації відбувається зліва направо.

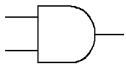
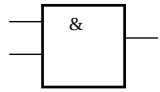
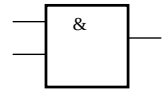
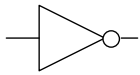
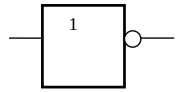
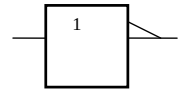
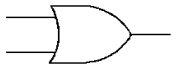
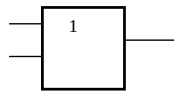
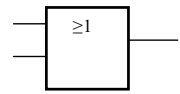
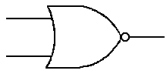
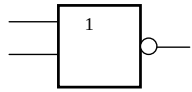
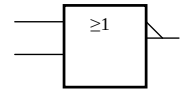
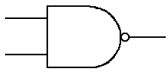
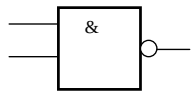
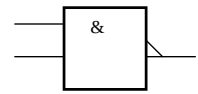
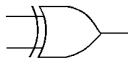
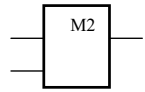
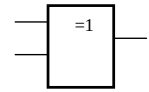
Якщо виходи одних елементів з'єднати з входами інших, то отримаємо схему, яка реалізує більш складну функцію. Сукупність різних типів елементів, достатніх для відтворення будь-якої логічної функції, назвемо логічним базисом. Елементи **„І"** й **„НЕ"** формують такий логічний базис. Елемент типу **„АБО"** може бути отримано з'єднанням елементів **„І"** та **„НЕ".**

Логічний базис може складатися лише з одного типу елементів, наприклад елемента типу **„І-НЕ"**, схему якого показано в табл. 7.1. Універсальність елемента **„І-НЕ"** забезпечила йому широке застосування при створенні засобів обчислювальної техніки. Існують й інші елементи, які реалізують найпростіші логічні функції. До них, наприклад, належать елементи додавання за модулем два, які реалізують функцію нерівнозначності двох сигналів.

Зазначимо, що кожен реальний логічний елемент має деякий час затримки зміни вихідного сигналу по відношенню до вхідного. Однак у комбінаційних схемах при їх формальному описі час затримки логічних елементів не враховується.

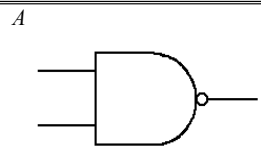
Таблиця 7.1

Приклади графічного зображення елементів логічних схем

Традиційне	Традиційне	IEEE/ANSI
Схема, яка реалізує логічну функцію **„І”**		
 $y = x_1 \wedge x_2$		
Схема, яка реалізує елементарну логічну функцію **„НЕ”**		
 $y = \bar{x}_1$		
Схема, яка реалізує елементарну логічну функцію **„АБО”**		
 $y = x_1 \vee x_2$		
Схема, яка реалізує елементарну логічну функцію **„АБО-НЕ”** (стрілка Пірса)		
 $y = \overline{x_1 \vee x_2}$		
Схема, яка реалізує логічну функцію **„І-НЕ”**(штрих Шеффера)		
 $y = \overline{x_1 \wedge x_2}$		
Схема, яка реалізує логічну функцію **„АБО, що виключає”** (тобто суму за модулем 2)		
 $y = x_1 \otimes x_2 = \bar{x}_2 x_1 + \bar{x}_1 x_2$		

Продовження табл. 7.1

ІНТЕРПРЕТАЦІЯ ПОЗНАЧЕННЯ ЛОГІЧНИХ ЕЛЕМЕНТІВ
Інтерпретація двох варіантів позначень логічного елемента „І-НЕ”

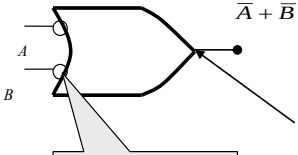


Вихідний сигнал буде низькими, лише якщо на якомусь з входів сигнал буде високими

$\overline{A \cdot B}$

Низький рівень сигналу

Збуджуваний високим рівнем сигналу



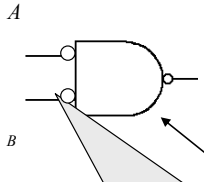
$\overline{A} + \overline{B} = \overline{AB}$

Вихідний сигнал буде високим, лише якщо на якомусь з входів сигнал буде низьким

Збуджуваний низьким рівнем сигналу

Високий рівень сигналу

ІНТЕРПРЕТАЦІЯ ПОЗНАЧЕННЯ ЛОГІЧНИХ ЕЛЕМЕНТІВ
Інтерпретація двох варіантів позначень логічного елемента „АБО”

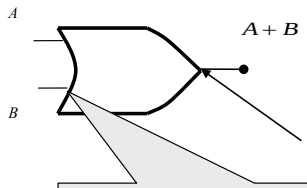


$\overline{\overline{A} \cdot \overline{B}} = A + B$

Вихідний сигнал буде низьким, лише якщо на якомусь із входів буде низький сигнал

Низький рівень сигналу

Збуджуваний низьким рівнем сигналу



$A + B$

Вихідний сигнал буде високим, якщо всі вхідні сигнали будуть високими

Високий рівень сигналу

Збуджуваний високим рівнем сигналу

Як уже було відзначено, прості (елементарні) логічні функції апаратно реалізуються за допомогою відповідних комбінаційних електронних логічних елементів – *вентилів*. Можна вважати, що ці елементарні логічні функції є *логічними операторами* згаданих електронних елементів, тобто схем. Кожну таку схему позначають певним графічним символом, який може бути орієнтованим або ж неорієнтованим. Наведемо приклади графічного позначення цих схем (див. табл. 7.1).

Аналізуючи або синтезуючи логічні ланцюги, слід ураховувати таку обставину.

Здійснити логічні функції на практиці дозволяють різноманітні так звані логічні (цифрові) напівпровідникові схеми – вентилі, вихідні сигнали яких однозначно визначено комбінаціями рівнів сигналів на входах цих схем. Причому, і вхідні, і вихідні сигнали цих вентилів можуть бути імпульсними або потенційними й мають два фіксовані значення: високий (H) або низький (L) рівень. Коли логічній „1” відповідає високий рівень, тоді логічному „0” – низький.

Якщо логічній „1” відповідає наявність сигналу (високого рівня), то в такому випадку говорять, що логічні схеми працюють у *позитивній логіці*. Якщо ж логічній „1” відповідає відсутність сигналу (низький рівень), то вважається, що схеми працюють у *негативній логіці*. Існує також і *змішана логіка*, тобто коли в електронному вузлі, що розглядається, одні вентилі працюють у позитивній логіці, а інші – у негативній.

Таким чином, відповідно до властивості подвійності логічних функцій, той самий вентиль, який у позитивній логіці реалізує функцію, наприклад, **„І-НЕ”** (чи **„АБО-НЕ”**), у негативній логіці виконуватиме логічну функцію **„АБО-НЕ”** (чи **„І-НЕ”**). Наприклад, вентиль, який реалізує функцію **„І-НЕ”** в позитивній логіці, графічно зображений на рис. 7.2.

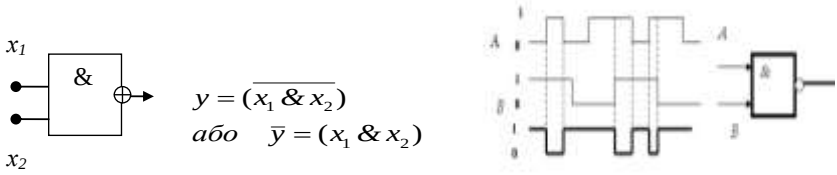


Рис. 7.2. Графічне зображення і часова діаграма роботи вентиля, який реалізує функцію „І-НЕ” в позитивній логіці

Графічне зображення цього ж вентиля при негативній логіці представлено на рис. 7.3.

Як випливає з наведених схем, там, де логічній „1” відповідає відсутність сигналу, а логічному „0” – наявність сигналу, на схемі прийнято креслити коло. Інакше кажучи, можна вважати, що коло означає, що на відповідний вхід логічного елемента подається інверсія логічного сигналу, а з виходу – виводиться інверсне значення сигналу або логічної функції.

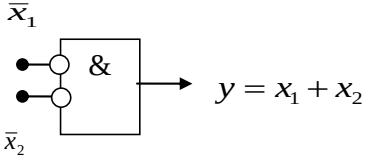


Рис. 7.3. Графічне зображення вентиля, який реалізує функцію „І-НЕ” в негативній логіці

Як показано рис. 7.3, випадок, коли логічній „1” відповідає відсутність сигналу, а логічному „0” – наявність сигналу, на схемі прийнято зображати колом. Наявність кола означає, що на відповідний вхід логічного елемента подається інверсія вхідного сигналу, а з виходу – виводиться інверсне значення сигналу, як відгука логічної функції.

Це дає можливість абстрагуватися від фізичних характеристик сигналів і вважати, що на входах і виходах вентилів (та інших компонентів) логічної схеми з’являються безпосередньо логічні значення сигналів. Отже, різні вентилі або їхні комбінації на логічному рівні можуть бути описані за допомогою логічних операторів. Такий операторний опис вентилів дозволяє абстрагуватися від фізичної природи конкретних вентилів та інших електронних елементів і здійснювати аналіз відповідних електронних схем.

Розглянемо, приклад реалізації логічного елементу **„АБО”** в реальних схемах.

У багатьох промислових системах контролю необхідно активізувати вихідну логічну функцію кожного разу, коли активізується який-небудь з входів, наприклад, у хімічних процесах це може бути потрібно при включенні сигналізації в момент перевищення максимально можливого значення температури або тиску. На рис. 7.4 показано блок-схему такої системи контролю. Ланцюг температурного датчика створює вихідну напругу, пропорційну температурі процесу. Ця напруга V_T порівнюється з еталонною напругою V_{TR} в ланцюгу схеми порівняння (компаратора) напруг. Зазвичай на виході компаратора $T_н$ низька напруга (логічний 0), але він переключається на напругу високого рівня (логічну 1) у момент, коли V_T перевищує V_{TR}, указуючи на те, що температура процесу надто висока.

Схожий механізм реалізується під час вимірювання тиску. Вихідний сигнал компаратора *Рн* переключається з низького на високий, коли тиск процесу стає високим.

Оскільки треба отримати сигнал тривоги, коли високими є температура або тиск, очевидно, що два вихідних сигнали компаратора подаються на двовхідний логічний елемент **„АБО”**. Вихідний сигнал логічного елемента **„АБО”**, таким чином, стане *високим* (тобто логічною 1) для будь-якої умови тривоги й активізує сигналізацію. Такий же підхід застосовано й у системах з великою кількістю оброблюваних сигналів.

Розглянемо ще кілька прикладів. Визначимо вихідний сигнал логічного елемента **„АБО”**, зображеного на рис. 7.5 а, б.

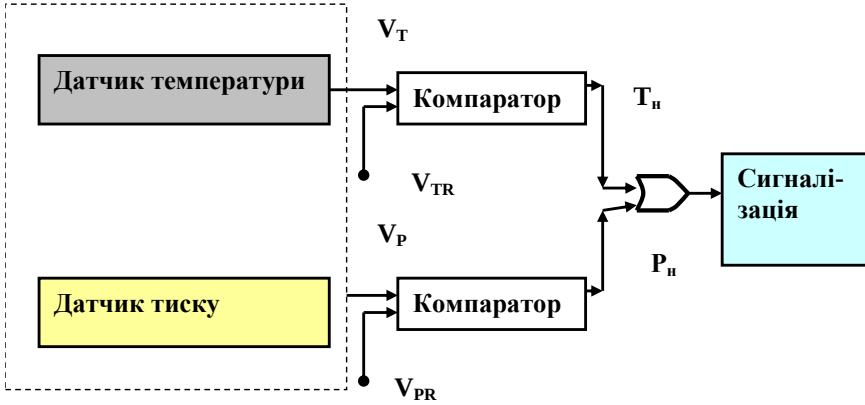


Рис. 7.4. Застосування логічного елемента «АБО»

Ланцюг температурного датчика створює вихідну напругу, пропорційну температурі процесу. Ця напруга V_T порівнюється з еталонною напругою V_{TR} в ланцюгу схеми порівняння (компаратора) напруг. Зазвичай на виході компаратора $T_н$ низька напруга (логічний 0), але він переключається на напругу високого рівня (логічну 1) у момент, коли V_T перевищує V_{TR}, фіксуючи той факт, що температура процесу перевищує контрольовану. Такого роду механізм реалізується і під час вимірювання тиску. Вихідний сигнал компаратора *Рн* переключається з низького на високий, коли тиск процесу перевищує контрольований.

Оскільки необхідно отримати сигнал тривоги, коли надто високими є температура або тиск, очевидно, що два вихідних сигнали компаратора подаються на двовхідний логічний елемент **„АБО”**. Вихідний сигнал логічного елемента **„АБО”** стає *високим* (тобто логічною 1) і активізує

сигналізацію. Аналогічний спосіб в інших промислових системах використовується для контролю значної кількості параметрів.

Розглянемо ще приклад роботи логічного елемента **„АБО”**.

На рис. 7.5 наведені часові діаграми вхідних і вихідних сигналів логічного елемента **«АБО»**.

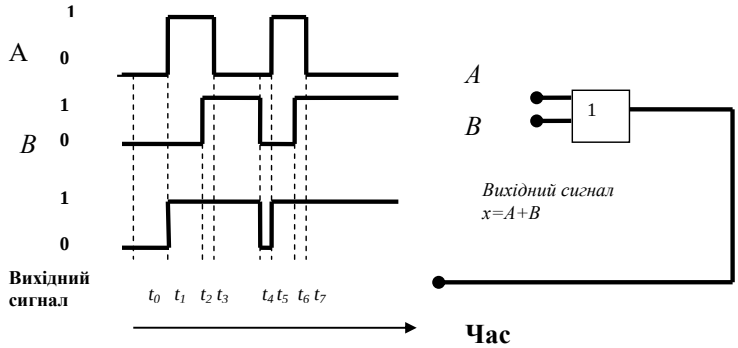


Рис. 7.5. Часові діаграми роботи логічного елементу **«АБО»**

Вхідні сигнали *A* і *B* змінюються відповідно до часової діаграми. Наприклад, сигнал *A* низький починаючи з моменту t_0; він стає високим у момент t_1 і знову низьким у момент t_3 і т. д.

Вихідний сигнал логічного елемента **«АБО»** буде високим, якщо *будь-який* з вхідних сигналів буде високим. Між моментами t_0 і t_1 обидва вхідні сигнали низькі, тому вихідний сигнал = 0. У момент t_1 вхідний сигнал *A* стає високим, тоді як *B* залишається низьким. Це призводить до того, що вихідний сигнал стає високим у момент t_1 і залишається таким до моменту t_4, поки один чи обидва вхідних сигнали високі.

У момент t_4 стан *B* змінюється з „1” на „0”, і тепер обидва вхідні сигнали низькі, тому вихідний сигнал стає низьким. У момент t_5 вхід *A* стає високим, перетворюючи вихідний сигнал знов у високий, і залишається таким до кінця показаного часового проміжку.

7.1.2. Задачі аналізу й синтезу електронних схем

Для аналізу електронних схем за допомогою апарату алгебри логіки треба знайти логічну функцію, що описує роботу заданої схеми. При цьому виходять з того, що кожному функціональному елементу електронної схеми можна поставити у відповідність логічний оператор. Цим самим

установлюють однозначну відповідність між елементами схеми та її математичним описом.

Аналіз електронної схеми проводять у три етапи [6, 8, 14, 22, 38]:

1) з принципової схеми вилучають усі неіснуючі допоміжні елементи, які не впливають на логіку роботи схеми;

2) через логічні оператори виражають усі електронні елементи, отримуючи логічне рівняння, яке є моделлю функції, виконуваною заданою схемою;

3) відомим способом отримують отримання мінімальні нормальні кон'юнктивні форми і отримання мінімальні нормальні диз'юнктивні форми цього рівняння й тим самим виявляють зайві частини аналізованої схеми.

Задачу синтезу електронних схем можна сформулювати так: при заданих вхідних сигналах і відомій вихідній функції спроектувати логічний пристрій, який реалізує цю функцію. Отже, у результаті розв'язання задачі синтезу виникає логічна схема, яка відтворює задану функцію.

Зазвичай, розв'язуючи задачі аналізу й синтезу, використовують повні базиси логічних функцій. При цьому кожну логічну функцію, що входить у базис, зіставляють з деяким фізичним електронним елементом, що дозволяє логічну схему замінити структурною схемою, яка складається з електронних елементів.

Таким чином, удається з'єднати математичну задачу синтезу логічної схеми з інженерною задачею проектування електронної схеми. Розробляючи електронну схему, за основні критерії приймають мінімум апаратури, мінімум типів застосовуваних елементів, максимум швидкодії й надійності.

З погляду математичної логіки задача синтезу розв'язується при забезпеченні мінімальної кількості логічних операторів, мінімальної кількості типів логічних операторів. Можна сформулювати послідовно розв'язувані задачі при синтезі електронної схеми [6,7, 14, 24, 27]:

- складання математичного опису (системи логічних рівнянь), який адекватно відображає процеси, що відбуваються в схемі;
- аналіз логічних рівнянь й отримання мінімальної форми для кожного з них у заданому базисі;
- перехід від логічних рівнянь до логічної (структурної) схеми за допомогою застосування логічних операторів.

Таким чином, синтез електронних схем розпочинається із завдання вхідних і вихідних сигналів. Очевидно, що все це може бути задано або відповідною таблицею істинності, або ж аналітично. У будь-якому випадку наступним етапом синтезу є процедура отримання будь-яким методом мінімальних нормальних кон'юнктивних форми і мінімальних нормальних диз'юнктивних форм відповідного логічного рівняння або рівнянь.

Далі на підставі отриманих мінімальних нормальних кон'юнктивних і диз'юнктивних форм синтезують структурні або функціональні схеми кількох варіантів синтезованої електронної схеми з використанням різних базисів. Після цього обирають як остаточний варіант ту структурну схему, яка має мінімальну кількість логічних електронних елементів, мінімальну кількість типів цих елементів і мінімальну „глибину" синтезованої схеми.

Під терміном „глибина" в цьому випадку розуміємо кількість логічних елементів на шляху сигналу від входу в схему до виходу. Мінімальна глибина забезпечує мінімальну затримку реакції схеми на зміну сигналів на її входах при вибраному типі (серії) електронних елементів синтезованої схеми.

Задача синтезу зазвичай має різні рішення залежно від вибраної системи логічних елементів. Однак для будь-якої заданої функції алгебри логіки майже завжди можна синтезувати схему, що відповідає цій функції. Отримання схеми з мінімальною кількістю логічних зв'язок вимагає знаходження мінімальної форми для логічної функції.

7.2. Електронні технології логічних елементів електронних обчислювальних машин

Електронні технології й елементи, на основі яких було створено електронну обчислювальну машину, багато разів змінювалися. Перше покоління електронних обчислювальних машин було побудоване на електронних лампах, друге – на дискретних напівпровідникових приладах (діодах і тріодах – транзисторах), наступні покоління – на інтегральних напівпровідникових схемах.

Змінювались електронні напівпровідникові елементи за видом використовуваних елементів, типом зв'язків між транзисторами. Зокрема, використовували такі системи елементів:

- резисторно-діодові;
- резисторно-транзисторні;
- ферито-транзисторні;
- діодово-транзисторні;
- транзисторно-транзисторні.

Найбільшого поширення в сучасних інтегральних схемах набули транзисторно-транзисторні системи елементів (*ТТЛ* – транзисторно-транзисторна логіка), у яких роль резисторів і діодів виконують транзистори з фіксованими напругами на своїх електродах. У цій системі забезпечено повну однорідність структури мікросхеми – вони містять лише транзистори, що полегшує технологію їх виготовлення [13,22, 29, 24 та ін.].

Архітектура використовуваних в електронних обчислювальних машинах транзисторів також зазнавала змін: у машинах другого покоління застосовували біполярні германієві й кременеві *pnp*- і *npn*- транзистори; в інтегральних схемах – уніполярні польові *МОП*-транзистори (*МОП* – метал-оксид-напівпровідник, або *MOS* – Metal-Oxide-Semiconductor).

Польові транзистори мають три електроди (див. рис. 7.6).

- затвор (аналог бази біполярних транзисторів);
- виток (аналог емітера);
- стік (аналог колектора).

Затвор електрично ізольований від інших електродів плівкою оксиду кременя, керує протіканням струму між витоком і стоком не шляхом дифузії електронів (як у *npn*-транзисторах) або дірок (як у *pnp*-транзисторах), а створюваним ним електростатичним полем. Тому метал-оксид-напівпровідник -транзистори й називають польовими.

Уніполярні транзистори мають більшу швидкодію, ніж біполярні, оскільки механізм їхньої роботи не пов'язаний з повільними дифузійними процесами. Елементи транзистора розміщено на пласкій кремнієвій підкладці (див. рис. 7.6).

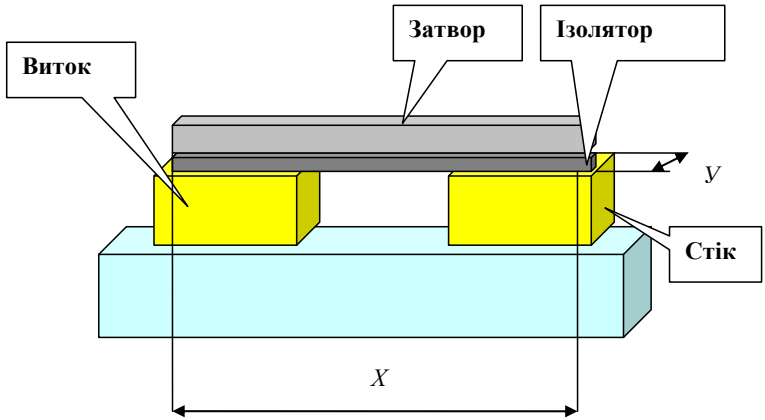


Рис. 7.6. Структура польового транзистора

Зазнавала змін й архітектура систем логічних елементів. Польові транзистори мають кілька різновидів:

- *n* метал-оксид-напівпровідник;
- *p* метал-оксид-напівпровідник;

• метал-оксид-напівпровідник з додатковою симетрією (*КМОП*-транзистори – комплементарна структура метал-оксид-напівпровідник, *CMOS* – Complementary Metal Oxide Semiconductor).

Первісно польові транзистори називалися *МДП*-транзисторами (метал-діелектрик-провідник), та оскільки як діелектрик стали використовувати оксид кременя, їх перейменували в метал-оксид-напівпровідник транзистори. Але, імовірно, найближчим часом доведеться повернутися до їхньої первісної назви, бо як ізолятор починають використовувати інший більш ефективний діелектрик, що має меншу, ніж оксид, діелектричну проникність і тим самим створює менші величини паразитних ємностей між електродами.

Транзистори *n* метал-оксид-напівпровідник і *p* метал-оксид-напівпровідник по відношенню до джерела живлення називаються послідовно ввімкненими, а по відношенню до вихідного сигналу – паралельно ввімкненими. Оскільки затвори *n* метал-оксид-напівпровідник або *p* метал-оксид-напівпровідник транзисторів ввімкнено паралельно, завжди один з цих транзисторів виявляється ввімкненим, а інший – вимкненим, і енергоспоживання й вихідний опір КМОП-схеми буде малим (невеликий струм протікатиме лише в перехідних режимах транзисторів). Затвор транзистора електрично ізольований від витоку й стоку, управління здійснює електростатичне поле, тому вхідний опір у польових транзисторів дуже великий.

Ця обставина створює зручність з'єднань *КМОП*-схем між собою й забезпечує стійкість їх роботи. *КМОП*-схеми мають менше енергоспоживання, ніж біполярні транзистори й інші типи польових транзисторів, можуть більш щільно упаковуватися; створені на їхній основі інтегральні схеми можуть бути використані в більш мініатюрному масштабі мікротехнологій.

Зараз *КМОП*-транзистори застосовують і в системах оперативної пам'яті й флеш-пам'яті. У модулях оперативної пам'яті для збереження одного біта інформації використовують конденсатор. Величина заряду цієї ємності визначає біт, що зберігається: наявність заряду – „0", відсутність заряду – „1".

У *КМОП*-транзисторах флеш-пам'яті для забезпечення енергонезалежності під затвором вміщено ще один, так званий *плаваючий затвор* (див. рис. 7.7). Плаваючий затвор має металізацію (плівку з арсеніду галію, хрому, нікелю, вольфраму та ін.) для створення на межі розділу між металом і напівпровідником потенційного бар'єра Шоткі, що дозволяє зберігати заряд конденсатора тривалий час.

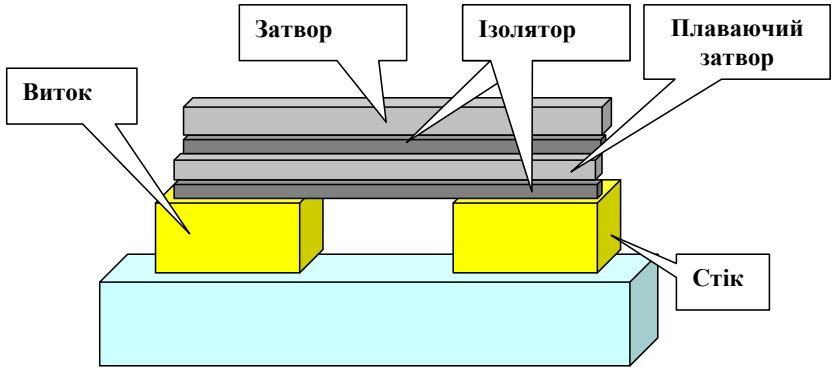


Рис. 7.7. Структура елемента флеш-пам'яті

Виготовляють інтегральні схеми з *МОП*-транзисторами за *планарною технологією*. На поверхні пластини з напівпровідника (кременю) наносять захисний шар діелектрика (зазвичай – діоксид кременя), у якому методами фотолітографії розкривають мікровікна. Поверх шару діелектрика наносять металеву плівку, яка має у вікнах контакт з поверхнею напівпровідника. Через вікна для створення електронно-діркових переходів потрібної (*n*- чи *p*-) полярності проводять дифузію матеріалів-донорів або акцепторів-електронів. Оскільки кремінь – чотиривалентний хімічний елемент, то для утворення *p*-областей використовують тривалентні матеріали (бор, галій, алюміній), а для створення *n*-областей – п'ятивалентні матеріали (сурма, миш'як, фосфор) [5,13].

Перспективною є розроблена в університеті Буффало технологія використання хімічних речовин, що *„самоорганізовуються"* – матеріалів з мікроскопічними структурами (*„квантовими точками"*) при виготовленні напівпровідникових приладів [5]. За даними досліджень [5], у названих речовинах навіть при кімнатній температурі самовільно відбувається реакція, яка призводить до створення регулярних мікроскопічних структур з гніздами діаметром 0,04 мкм.

Параметри транзисторів залежать від масштабу технологічного процесу їх виготовлення (масштабу технології), який безперервно вдосконалюють. Зараз використовують технології 0,09 – 0,065 мкм.

Зменшення розмірів транзисторів підвищує щільність їхнього розміщення, зменшує паразитні індуктивності та ємності електродів і дозволяє підвищити робочу частоту мікросхеми. Але при цьому мініатюризація транзисторів (в окремих випадках товщина ізолюючих

шарів у транзисторі зіставлювана з розмірами атомів) призводить до росту паразитних струмів витоку, що, у свою чергу, підвищує енергоспоживання й знижує стійкість роботи схеми. Зниження напруги живлення схеми зменшує розігрів схем лише частково, а потужність струмів витоку може досягати сотень ватт.

Зменшення струмів витоку досягають такими способами:

- використанням мідних провідників (замість алюмінієвих, що мають більший питомий електричний опір);
- застосуванням технології напруженого кременя (при збільшенні відстані між атомами кристалічної решітки зменшується питомий електричний опір).

Логічні операції **„І”, „НЕ”, „АБО”** доволі просто технічно виконуються на будь-яких системах елементів. Найпростіші принципові електричні схеми **„АБО”, „І”** на резисторно-діодних елементах і схему **„НЕ”** на біполярних транзисторах показано на рис. 7.8, а, б, в відповідно.

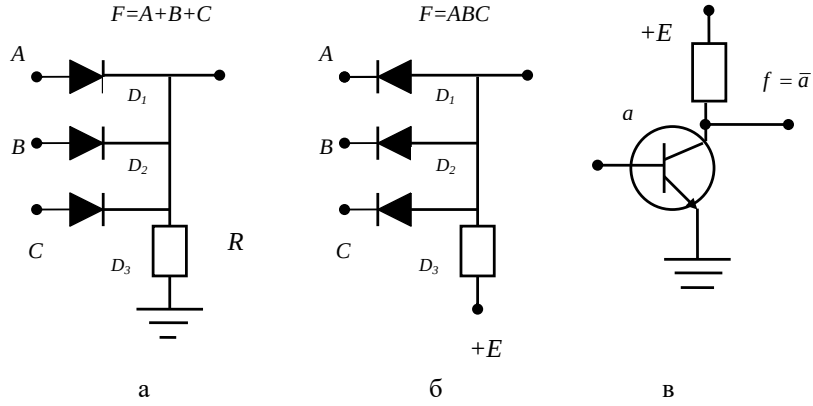


Рис. 7.8. Найпростіші принципові електричні схеми „АБО”, „І”, „НЕ”

Пояснення до схеми **„АБО”**: додатний імпульс на виході виникає при появі додатного імпульсу на будь-якому (*A, B, C*) вході, оскільки внутрішній опір діода в прямому напрямку малий (набагато менший за *R*).

Пояснення до схеми **„І”**: додатний імпульс на виході виникає лише при одночасній наявності додатних імпульсів на всіх трьох (*A, B, C*) виходах. За відсутності хоча б одного вхідного імпульсу відповідний йому діод буде відкритий і замкне напругу живлення *+E* через внутрішні опори діода й джерела вхідного сигналу (вони набагато менші за *R*) на „землю”.

Пояснення до схеми **„НЕ”**: при подачі на вхід (базу) *npn*-транзистора додатного імпульсу тріод відкриється й на виході (колекторі) напруга з високої знизиться практично до нуля.

Реалізацію **„АБО”, „І”, „НЕ”** на основі унікальних операторів використовують у логічному синтезі обчислювальних схем, оскільки для базових операторів процедури формалізованого логічного синтезу її розроблено найбільш детально й конструктивно.

Серед багатьох елементарних схем у цифрових автоматах найбільшого поширення набула схема тригера – статичного запам’ятовуючого й логічного елемента.

На тригерах побудовано системи статичної пам’яті, регістри, лічильники, дільники частоти й ще багато інших комп’ютерних схем.

Тригер – елемент, який може перебувати в одному з двох стійких станів, які умовно іменують станами „0” і „1”. Тригер має два виходи:

- вихід „0” (іноді іменований $\bar{q}$ -виходом);
- вихід „1” (іменований іноді q -виходом).

Якщо тригер перебуває в стані „0”, то в нього на виході q „висока” напруга (порядку кількох вольт або менше), на виході $\bar{q}$ „низька” (нульова) напруга, якщо тригер перебуває в стані „1”, то напруги розподілені навпаки.

Тригери можуть мати роздільні входи:

- *R (Reset)* – вхід установки „0”;
- *S (Set)* – вхід установки „1”.

Кожен вхід установлює тригер у відповідний стан, такі тригери називають *R-S*-тригерами. Тригери можуть мати лічильний вхід *T* (релаксатор), черговий імпульс „1” на вході змінить стан тригера. Такі тригери називають *T*-тригерами.

Тригер, установлений у будь-який стан, зберігає його до тих пір, поки імпульс, поданий на один з входів, не змінить його стану.

Логічні схеми *R-S* і *T*-тригера показано на рис. 7.9.

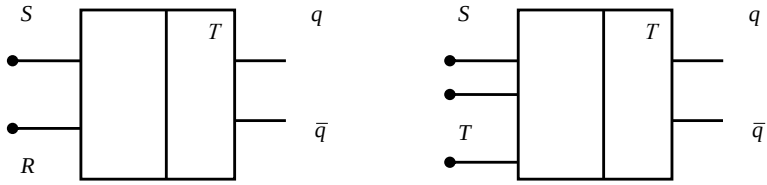


Рис. 7.9. Логічні схеми *R-S* (а) і *T* - тригера (б)

Принципову електричну *КМОП*-схему *R-S* тригера, виконаного за транзисторно-транзисторною технологією, показано на рис. 7.10. Для установки тригера в стан „1” необхідно подати імпульс або високу напругу на точку *(S,q)*.

Цей сигнал піде на затвори транзисторів T_2 і T_4. Транзистор T_2 з каналом *n* типу відкриється, а транзистор T_4 з каналом *p* типу закриється. Напруга на стоку транзистора T_2 (точка $R,\ \bar{q}$) стане низькою. Низька напруга пройде на затвори транзисторів T_1 і T_3. Транзистор T_1 з каналом *n* типу закриється, а транзистор T_3 з каналом *p* типу відкриється. На стоку транзистора T_1 з’явиться висока напруга, яка пройде на затвори транзисторів T_1 і T_4 і підтримуватиме стан „1” тригера до тих пір, поки на вхід ($R,\ \bar{q}$) не надійде імпульс чи висока напруга.

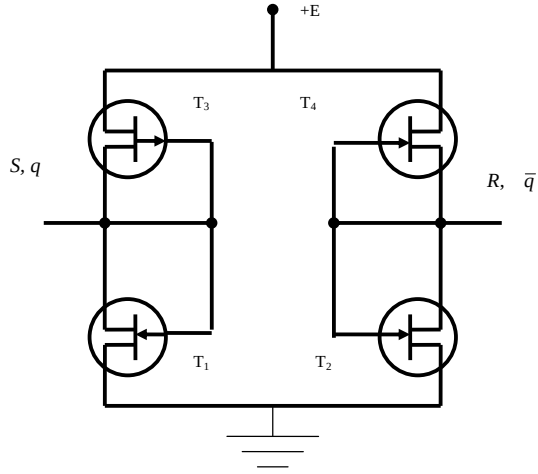


Рис. 7.10. Електрична *КМОП*-схема R-S тригера, виконаного за транзисторно-транзисторною технологією

Інформація з тригера зчитується з його виходів шляхом опитування їх через схеми **AND** (**„І”**). Якщо на виході, який підключено до одного входу схеми **AND**, висока напруга, то сигнал опитування пройде й нестиме інформацію про стан тригера.

Тригери використовують для організації запам’ятовуючих регістрів і лічильників. При цьому в регістрах здебільшого використовують тригери з роздільними входами, а в лічильниках – з лічильними.

Логічну схему 3-розрядного регістра з вентилями (схемами **AND**) для введення й зчитування інформації показано на рис. 7.11.

У кожному *i*-му розряді регістр містить *RS* тригер T_i і підключену до нього для зчитування інформації схему **AND**. Зчитування інформації з регістра відбувається так. При подачі імпульсу зчитування, який опитує схеми **AND** усіх тригерів, на розрядні виходи a_i надійде „1” через вентилі, тригери яких були в стані „1”. Запис інформації в регістр може виконуватися у двох режимах – одно- й двотактному. В однотактному режимі на відповідний вхід кожного тригера подається „1”. У двотактному режимі всі входи R тригерів підключено до одного проводу установки „0”, за яким спочатку всі тригери встановлюються на нульову позначку, а потім на входи S тих тригерів, які треба встановити в „1”, подається відповідний імпульс.

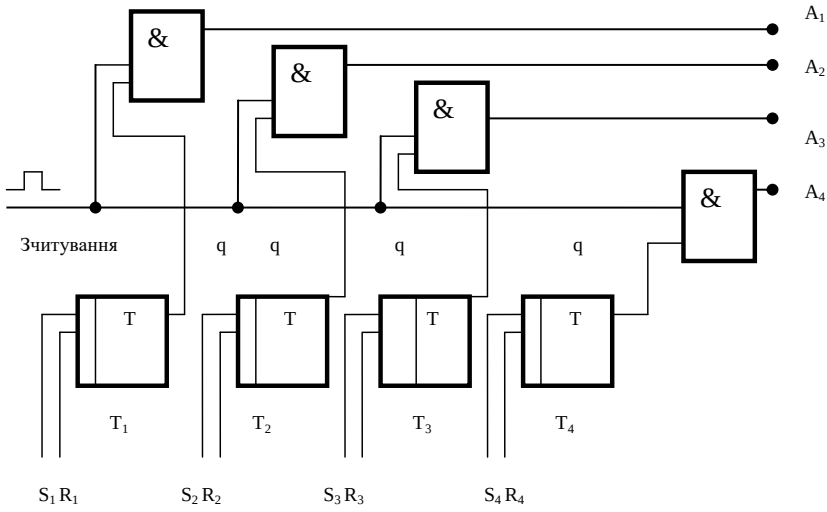


Рис. 7.11. Логічна схема регістра

7.3. Синтез комбінаційних схем

Як було відзначено, комбінаційна схема може мати кілька виходів. При канонічному методі синтезу передбачено, що кожна вихідна функція

реалізується власною схемою. Сукупність цих схем і дає потрібну комбінаційна схема. Таким чином, синтез складної комбінаційної схеми з n виходами заміняється синтезом n схем з одним виходом.

Відповідно до канонічного методу синтез комбінаційної схеми містить кілька етапів [7, 14, 22-25, 27, 31 та ін.]:

1. Булеву функцію, що підлягає реалізації, представлено у вигляді досконалої диз'юнктивної нормальної форми.

2. З використанням методів мінімізації (див. розділ 6) визначають мінімальну диз'юнктивну нормальну форму або мінімальну кон'юнктивну нормальну форму. З отриманих мінімальних форм вибирають більш просту.

3. Булеву функцію в мінімальній формі представляють у заданому (або вибраному розробником) базисі (див. табл. 7.1).

4. За представленням функції в заданому базисі будують комбінаційну схему.

Слід відзначити, що булева функція, яка підлягає реалізації $f(x_1,x_2,...,x_m)$, може бути задана не на всіх можливих наборах аргументів x_1, x_2, ..., x_m. На тих наборах, де функція невизначена, її довизначають так, щоб у результаті мінімізації отримати більш просту мінімальну диз'юнктивну нормальну форму або мінімальну кон'юнктивну нормальну форму. При цьому спроститься й сама комбінаційної схеми. Часто з метою отримання більш простого представлення функції мінімальної диз'юнктивної нормальної форми вони представляються в так званій формі дужок, тобто за дужки виносять спільні частини імплікант мінімальної диз'юнктивної нормальної форми.

Розглянемо канонічний метод синтезу на прикладі побудови комбінаційної схеми перетворювача двійково-десяткового коду у спеціальний семи розрядний код, який керує роботою семиелементного цифрового індикатора, схема включення якого показана на рис. 7.12. Сигнали поступають на цифровий індикатор з виходів перетворювача кодів. На входи перетворювача подаються логічні сигнали чотирьох розрядних двійково-десяткових кодів. Задача полягає в синтезі такої логічної схеми перетворювача, яка сформує на індикаторі зображення цифр, відповідних вхідним двійково-десятковим кодам.

Вихідні логічні функції $f_1...,f_7$ є функціями вхідних аргументів x_1 ..., x_4. Таблиця істинності функцій $f_1...,f_7$ наведена далі, див. табл. 7.2. Значення функцій $f_1...,f_7$ відповідають елементам, що підсвічуються при зображенні цифри. Наприклад, десяткова цифра 0 задається двійково-десятковим кодом 0000. При цьому підсвічуються всі елементи, за винятком f_7. Цифра 7 задається кодом 0111, а підсвічуються елементи f_2, f_3 і f_4. Логічні функції в таблиці визначені не повністю: останні шість кодових комбінацій не відповідають ніяким десятковим цифрам і є забороненими.

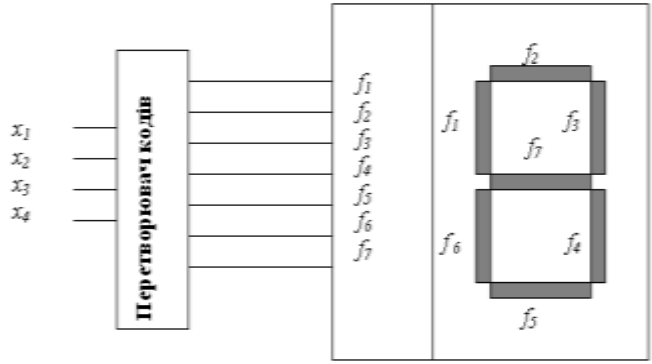


Рис. 7.12. Схема включення перетворювача кодів

Таблиця 7.2

Таблиця істинності логічних функцій перетворювача кодів

Цифра	x_1	x_2	x_3	x_4	f_1	f_2	f_3	f_4	f_5	f_6	f_7
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	0	1	1	0	0	0
2	0	0	1	0	0	1	1	0	1	1	1
3	0	0	1	1	0	1	1	1	1	0	1
4	0	1	0	0	1	0	1	1	0	0	1
5	0	1	0	1	1	1	0	1	1	0	1
6	0	1	1	0	1	1	0	1	1	1	1
7	0	1	1	1	0	1	1	1	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	1	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	*	*	*	*	*	*	*

Зобразимо карту Карно для логічної функції f_1, див. рис. 7.13. На карті Карно гнізда, що відповідають забороненим наборам аргументів, позначено знаком *.

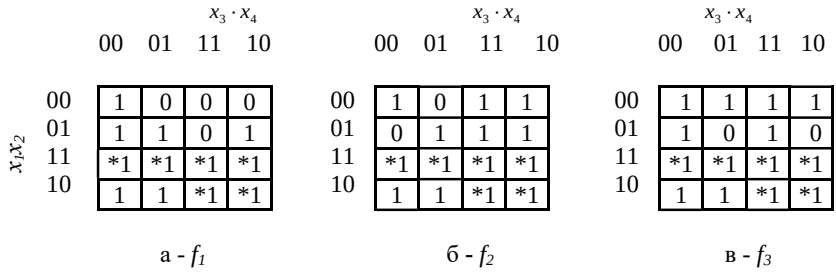


Рис. 7.13. Карти Карно функцій f_1, f_2, f_3

Аналогічно, до визначивши одиницями всі невизначені значення функцій f_2 і f_3, отримаємо для них якнайкращі покриття і мінімальні диз'юнктивні форми див. табл.7.3. та рис. 7.14.

Таблиця 7.3

Таблиця логічних функцій f_1, f_2, f_3 перетворювача кодів

f_1	$f_1 = x_1 \vee x_2 \cdot \bar{x}_4 \vee x_2 \cdot \bar{x}_3 \vee \bar{x}_3 \cdot \bar{x}_4$
f_2	$f_2 = x_1 \vee x_3 \vee x_2 \cdot x_4 \vee \bar{x}_2 \cdot \bar{x}_4$
f_3	$f_3 = x_1 \vee x_2 \vee x_3 \cdot x_4 \vee \bar{x}_3 \cdot \bar{x}_4$

Таким чином, задача синтезу має зазвичай кілька рішень. Для порівняння різних варіантів комбінаційних схем використовують їхні основні характеристики – складність і швидкодію.

Складність схеми оцінюють кількістю обладнання, яке складає схему. У розробці схем на основі елементної бази кількість обладнання зазвичай вимірюють кількістю корпусів (модулів) інтегральних мікросхем, використовуваних у схемі. У теоретичних розробках орієнтуються на довільну елементну базу, і тому для оцінки затрат обладнання використовують оцінку складності схем за Квайном [7, 14, 22-25, 27, 31].

Складність (ціна) за Квайном визначається сумарною кількістю входів логічних елементів у складі схеми.

При такій оцінці одиниця складності – один вхід логічного елемента. Ціна інверсного входу зазвичай приймається рівною двом. Цей підхід до оцінки складності виправданий з таких причин:

- складність схеми легко обчислюється за булевими функціями, на основі яких побудовано схему. Для нормальної диз'юнктивної форми складність схеми дорівнює сумі кількості букв (букві зі знаком від'ємності відповідає ціна 2) і кількості знаків диз'юнкції, збільшеній на 1 для кожного диз'юнктивного виразу;
- усі класичні методи мінімізації булевих функцій забезпечують мінімальність схеми саме в розумінні ціни за Квайном.

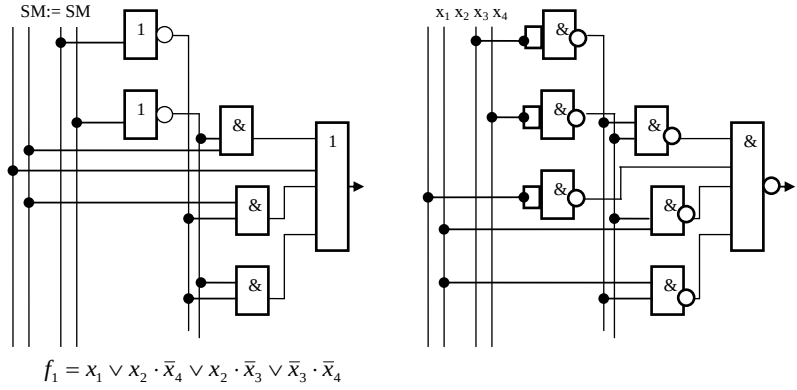


Рис. 7.14. Реалізація функції f_1

Практика показує, що схема з мінімальною ціною за Квайном зазвичай реалізується найменшою кількістю конструктивних елементів – корпусів інтегральних мікросхем [5, 30].

Швидкодію схеми оцінюють максимальною затримкою сигналу при проходженні його від входу схеми до виходу, тобто визначають проміжком часу від моменту надходження вхідних сигналів до моменту встановлення відповідних значень вихідних сигналів. Затримка сигналу кратна кількості елементів, через які проходить сигнал від входу до виходу схеми. Тому швидкодія схеми характеризується значенням $r \cdot \tau$, де τ – затримка сигналу на одному елементі, r – кількість рівнів комбінаційної схеми. Параметр r

розраховують так. Входам КС відповідає нульовий рівень (див. рис. 7.15). Логічні елементи, зв'язані лише з входами схеми, належать до першого рівня. Елемент належить до рівня *k*, якщо він зв'язаний за входами з елементами рівнів *k-1*, *k-2* і т. д. Максимальний рівень елементів *r* визначає кількість рівнів КС, що називається *рангом схеми* (див. рис. 7.15).

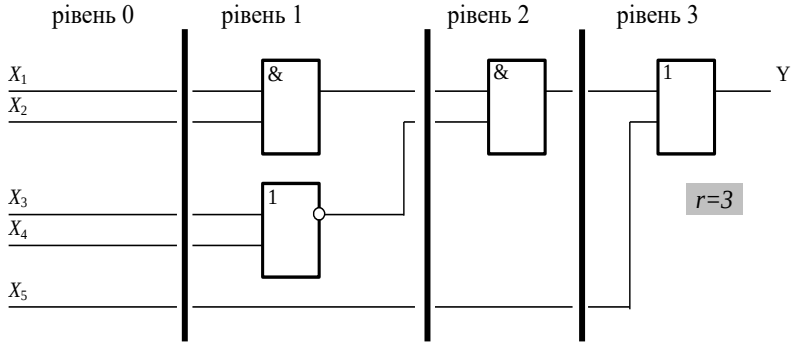


Рис. 7.15. Визначення рангу схеми

Раніше ми встановили, що будь-яка булева функція може бути представлена в *НДФ*, якій відповідає дворівнева комбінаційна схема. Отже, швидкодію будь-якої КС в принципі можна довести до 2τ.

Мінімізація булевої функції з метою зменшення складності схем зазвичай призводить до необхідності представлення функцій у формі дужок, якій відповідають схеми з $r>2$. Тобто, зменшення затрат обладнання в загальному випадку призводить до зниження швидкодії схем.

У побудові КС пристроїв обчислювальної техніки використовують різні логічні елементи, які мають бути узгоджені за вхідними й вихідними сигналами, напругою живлення й т. ін. З цією метою логічні елементи об'єднують у серії.

Серією (системою, комплексом) логічних елементів цифрового автомата називається призначений для побудови цифрових пристроїв функціонально повний набір логічних елементів, об'єднаний спільними електричними, конструктивними й технологічними параметрами, який використовує однаковий спосіб представлення інформації, однаковий тип міжелементних зв'язків. Система елементів найчастіше надмірна за її функціональним складом, що дозволяє будувати схеми більш економічні за кількістю використаних елементів.

До складу серії входять елементи для виконання логічних операцій, запам'ятовуючі елементи, елементи, які реалізують функції вузлів ЦА, а

також спеціальні елементи для посилення, відновлення й формування сигналів стандартної форми.

Конструктивно логічні елементи становлять собою інтегральні електронні схеми (мікросхеми), сформовані в кристалі кременю за допомогою спеціальних технологічних процесів [5, 30].

Логічні елементи у вигляді інтегральних схем реалізують сукупність простих логічних операцій.

Основними параметрами серії логічних елементів є:

- живляча напруга й сигнали для представлення логічного „0" й логічної „1";
- коефіцієнти об'єднання за входом;
- навантажувальна здатність (коефіцієнт розгалуження за виходом);
- стійкість до перешкод;
- потужність, що розсіюється;
- швидкодія.

Серія елементів характеризується кількістю використовуваних *живлячих напруг* і їхніми номінальними значеннями. Зазвичай логічному „0" відповідає низький рівень напруги, а логічній „1" – високий. Для найбільш часто використовуваних серій напруга живлення становить +5 В, рівень логічної одиниці 2,4 – 5 В, рівень логічного нуля – 0 – 0,4 В.

Коефіцієнт об'єднання за входом *($K_{об}$)* визначає максимально можливу кількість входів логічного елемента, інакше кажучи, функцію скількох сигналів (аргументів) може реалізувати цей елемент. Зазвичай $K_{об}$ набуває значення від 2 до 4, рідше $K_{об}$=8. Збільшення кількості входів пов'язане з ускладненням схеми елементів і призводить до погіршення інших параметрів – стійкості до перешкод, швидкодії й т. ін.

Коефіцієнт розгалуження за виходом *($K_{роз}$)* показує, на скільки логічних входів може бути одночасно навантажений вихід цього логічного елемента. Зазвичай $K_{роз}$ для найбільш часто використовуваних серій дорівнює 10. Іноді замість $K_{роз}$ задається гранично допустиме значення вихідного струму логічного елемента в стані „0" або „1".

Стійкість до перешкод – це здатність елемента правильно функціонувати за наявності перешкод. Вона визначається максимально допустимою напругою перешкоди, при якому не відбувається збою в його роботі. Зазвичай це напруга порядку 0,6 – 0,9 В.

Швидкодія логічних елементів є одним з найважливіших параметрів і характеризується часом затримки поширення сигналу. Цей параметр суттєво залежить від технології виготовлення мікросхем і лежить у діапазоні від одиниць до сотень наносекунд.

При синтезі КС на реальних логічних елементах необхідно обов'язково враховувати обмеження на $K_{об}$ і $K_{роз}$.

7.3.1. Синтез логічних схем з одним виходом

Перш ніж перейти до прикладів синтезу комбінаційних логічних схем, розглянемо способи використання універсальності вентилів **„АБО-НЕ”** та **„І-НЕ”** (див. рис. 7.16, 7.17).

Властивість універсальності вентиля **„АБО-НЕ”**:

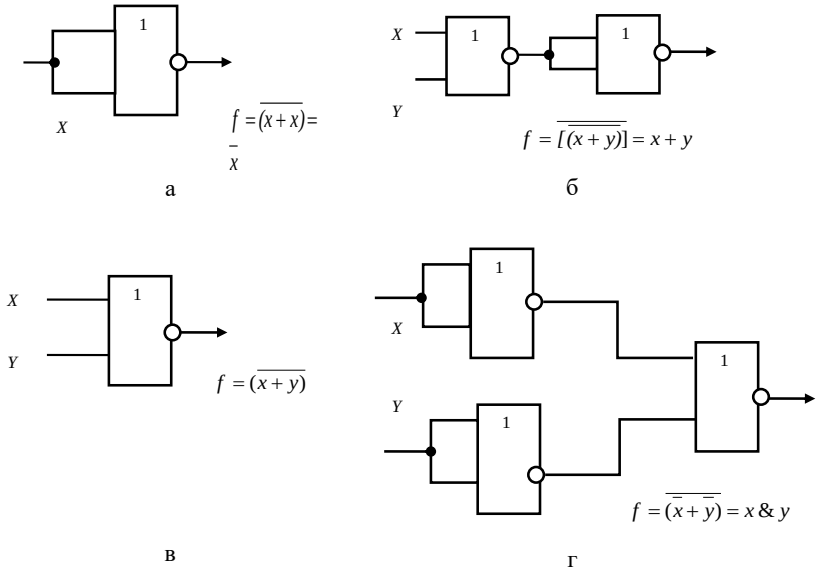


Рис. 7.16. Способи використання універсальності вентилів „**АБО-НЕ**”

Коли необхідний вихідний рівень логічної схеми задано для будь-яких вхідних умов, результат зазвичай можна відобразити в таблиці істинності. З таблиці істинності також отримують булевий вираз для потрібної схеми. Наприклад, розглянемо рис. 7.18 а, на якому зображено таблицю істинності для ланцюга з двома входами, *A* і *B*, і виходом *x*. У таблиці показано, що вихідний сигнал *x* дорівнюватиме 1 лише тоді, коли *A=0* і *B=1*. Тепер залишається визначити, яка логічна схема відповідає здійсненій операції. На рис. 7.18 б показано одне з можливих рішень – тут елемент **„І”** використано разом з вхідними сигналами $\overline{A}$ і *B*, так що $x = \overline{A} \cdot B$. Зрозуміло, що *x* дорівнюватиме 1, лише якщо обидва вхідних сигнали, які приходять на елемент **„І”**, дорівнюють 1, тобто $\overline{A}$ *=1* і *B=1*. Для всіх інших значень *A* і *B* вихідний сигнал *x* дорівнюватиме 0.

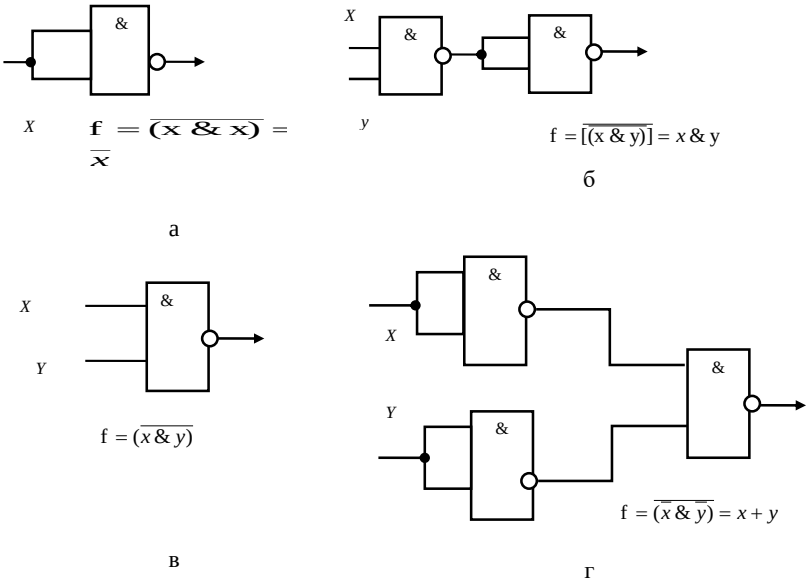
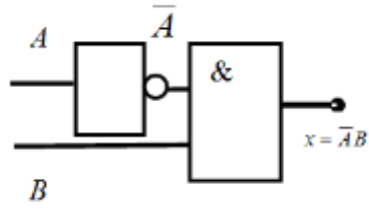


Рис. 7.17. Способи використання універсальності вентилів „І-НЕ”

Таблиця істинності

A	*B*	*x*
0	0	0
0	1	1
1	0	0
1	1	0

а



б

Рис. 7.18. Схема вихідного сигналу, що дорівнює *1*, якщо *A=0* і *B=1*

Такий підхід можна використовувати й для інших вхідних умов. Наприклад, якщо б *x* повинен був мати високий рівень лише за умови, що *A=1* і *B=0*, то в схемі містився б елемент **„І”** для вхідних сигналів *A* і $\overline{B}$. Інакше кажучи, для кількох з чотирьох можливих умов на вході можна отримати високий рівень сигналу *x,* подаючи на елемент **„І”** такі вхідні сигнали, які дозволяють отримати потрібний добуток.

Схеми з одним виходом і кількома входами належать до найпростіших. Основна складність у синтезі цих схем полягає в тому, щоб знайти вираз для вихідної функції в заданому базисі.

Розглянемо деякі прості приклади переходу від логічних рівнянь до логічних ланцюгів, тобто приклади синтезу простих логічних ланцюгів. Зокрема, розглянемо перехід від представлення функції в нормальну диз'юнктивну форму до її реалізації на елементах **„І-НЕ"** та **„АБО-НЕ"**.

Нехай задано функцію чотирьох аргументів у нормальну диз'юнктивну форму:

$$F = \overline{C} \cdot D + \overline{A} \cdot \overline{B} \cdot \overline{D} + A \cdot B \cdot \overline{D} + \overline{A} \cdot B \cdot \overline{C} + A \cdot \overline{C} \cdot \overline{D}.$$

Проведемо її мінімізацію за допомогою карти Карно (див. рис. 7.19).

		AB			
		00	01	11	10
CD	00	1	1	1	1
	01	1	1	1	1
	11	0	0	0	0
	10	1	0	1	0

Рис. 7.19. Карта Карно

Мінімальна нормальна диз'юнктива форма має вигляд: $F = \overline{A} \cdot \overline{B} \cdot \overline{D} + A \cdot B \cdot \overline{D} + \overline{C}$.

Розглянемо реалізацію цього рівняння за допомогою елементів **„І-НЕ"**. У загальному випадку на елементах **„І-НЕ"** нормальній диз'юнктивій формі функцію реалізовано за допомогою двох сходинок логіки. На першій сходинці виходять інверсні значення логічних добутків й однобуквених членів. На другій сходинці виконуютється операція **„НЕ-АБО"**, над отриманими інверсіями.

За допомогою застосування подвійного заперечення можна привести задану функцію й відповідну схему до вигляду (див. рис. 7.20).

У наведеній схемі для елементів першої й другої сходинок застосовано різні, але еквівалентні умовні позначки. Під час реалізації *НДФ* функції за допомогою елементів **„І-НЕ"** такий прийом дозволяє проектувати схеми, користуючись операціями **„І", „АБО"** й **„НЕ"**.

Розглянемо наступний приклад. На елементах **„АБО-НЕ"** нормальної диз'юнктивої форми функції $F = \overline{A} \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot \overline{C} + \overline{D} \cdot \overline{C}$ реалізовно за допомогою трьох сходинок (див. рис. 7.21). На першій сходинці за допомогою операції **„АБО-НЕ"** над значеннями аргументів, що входять у нормальну диз'юнктивну форму, утворено логічні суми. На

другій сходинці виконано операцію **„І-НЕ”** над логічними сумами й однобуквеними членами (якщо вони є). На третій сходинці виконано інверсію й отримано шукану функцію.

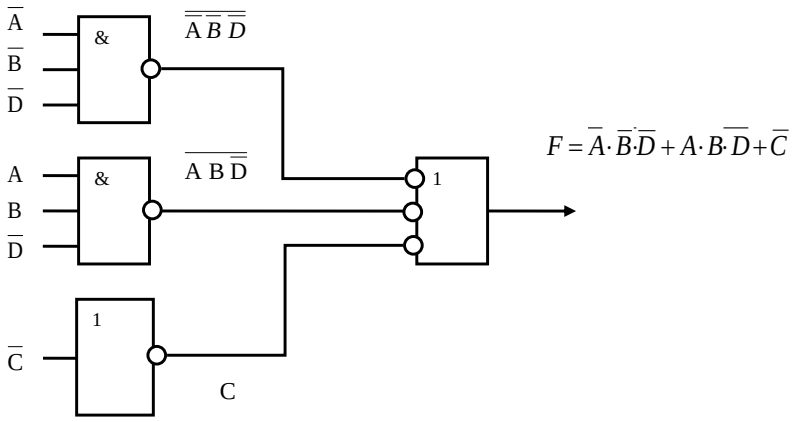


Рис. 7.20. Комбінаційна схема

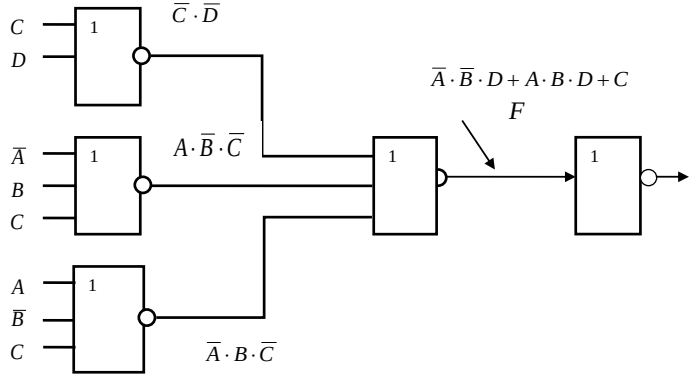


Рис. 7.21. Комбінаційна схема

При мінімізації логічних функцій для логічних схем, які передбачають будувати на базі елементів **„І-НЕ”** чи **„АБО-НЕ”**, необхідно, крім власне мінімізації, прагнути також до того, щоб структурна формула була представлена у вигляді комбінації з елементів **„І-НЕ”** чи **„АБО-НЕ”**. Тоді перехід від структурної формули до

функціональної схеми не буде складним. У будь-якому випадку, будуючи логічну схему в базисі **„І-НЕ”** на основі логічної функції, представленої в мінімальній нормальній диз’юнктивій формі, необхідно скрізь замість елементів **„І”** та **„АБО”** ставити елемент **„І-НЕ”**. Будуючи логічну схему в базисі **„АБО-НЕ”** на основі логічної функції, представленої в мінімальну нормальну кон’юнктивну форму, необхідно скрізь замість елементів **„І”** та **„АБО”** ставити елемент **„АБО-НЕ”**. Однак слід урахувати, що є думка, відповідно до якої найзручнішим для вирішення синтезу схем цифрових автоматів є базис **„І”, „АБО”, „НЕ”.**

Розглянемо, наприклад, вираз $F = D + \overline{B}\,\&\,C + \overline{D}\,\&\,(A + B)$. Застосувавши закон де Моргана, отримаємо таку схему та наступний вираз $F = D + \overline{B}\,\&\,C + \overline{D}\,\&\,(A + B) = A + B + C + D$, (див. рис. 7.22).

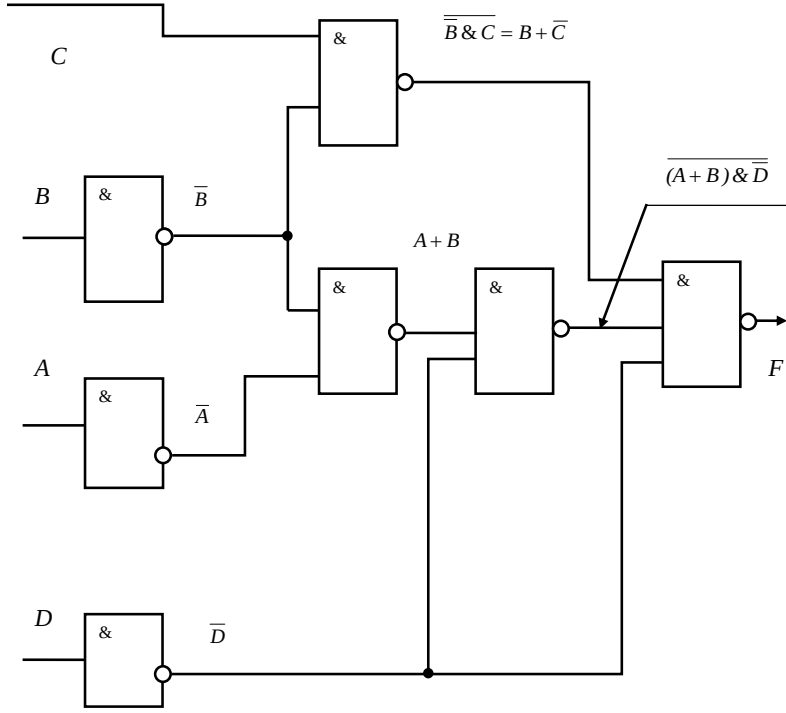


Рис. 7.22. Комбінаційна схема

Функцію реалізовано за допомогою вентилів типу **„І-НЕ”**. Ця сама функція, реалізована в іншому, матиме вигляд (див. рис. 7.23).

Тепер розглянемо способи формування схеми, яка реалізує функцію додавання за модулем 2 **(mod2)**, у різних базисах. Логічна функція mod2, як відомо, в аналітичному вигляді представляється: $f = A \cdot \bar{B} + \bar{A} \cdot B$ і має таку таблицю істинності (див. табл. 7.4).

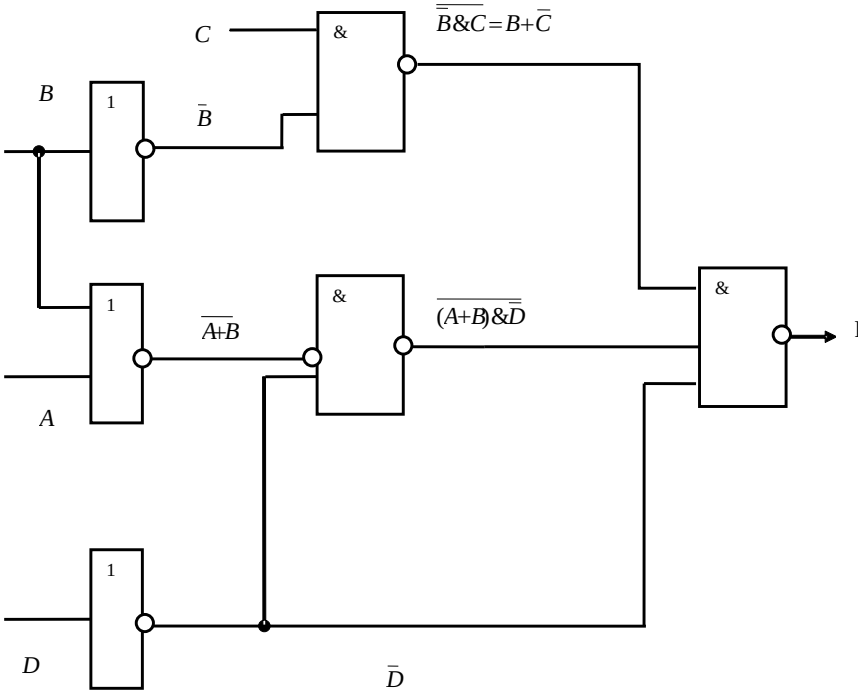


Рис. 7.23. Комбінаційна схема

Т а б л и ц я 7 . 4

Таблиця істинності

A	***B***	***f***
0	0	0
0	1	1
1	0	1
1	1	0

У базисі **„І”, „АБО”, „НЕ”** схема, яка реалізує функцію **mod2**, має такий вигляд (див. рис. 7.24).

Більш складні схеми, які мають кілька виходів, можуть бути, зокрема, представлені як набір схем з одним виходом. Тоді синтез здійснюють шляхом декомпозиції для кожної схеми. Розглянемо приклад синтезу однорозрядного двійкового суматора методом декомпозиції. Суматор задано таблицею (див. табл. 7.5). Також є карта Карно (див. рис. 7.25).

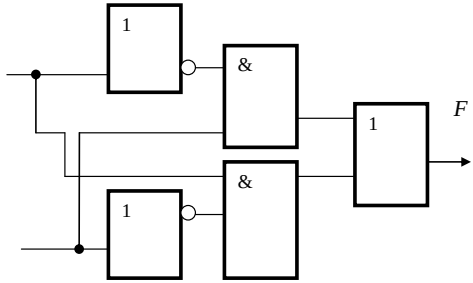


Рис. 7.24. Схема, яка реалізує функцію mod 2

Таблиця 7.5

Таблиця істинності

A	B	C_i	S	C_0
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

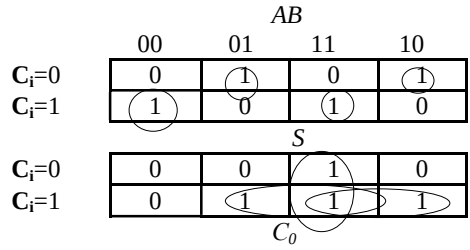


Рис. 7.25. Карта Карно

На виході S маємо результат додавання двох однорозрядних чисел – A і B – і переносу C_i (перенос, який при цьому виникає, подається на вихід C_0). Синтезовану схему можна розглядати як схему, що складається з двох частин:

1) схеми для отримання порозрядної суми S (півсуматор);

2) схеми для отримання переносу C_0.

Для складання схеми суматора можуть бути використані частково мінімізовані функції, і після різноманітних алгебраїчних перетворень можна отримати такі диз'юнктивні нормальні форми для функцій S і C_0:

$$S = \overline{A}\cdot\overline{B}\cdot C_i + A\cdot\overline{B}\cdot\overline{C_i} + \overline{A}\cdot B\cdot\overline{C_i} + A\cdot B\cdot C_i =$$
$$(A+B+C_i)\cdot\overline{C_0} + A\cdot B\cdot C_i,$$

де $C_0 = A\cdot B + C_i\cdot A + C_i\cdot B = A\cdot B + C_i\cdot(A+B)$.

На підставі цих рівнянь можна побудувати схему повного суматора на елементах **„І", „АБО", „НЕ"**.

Введемо такі позначки: E = A ⊕ B; D = A·B; T = A + B.

Тоді остаточно отримаємо:

$$S = E \oplus C_i; \qquad C_0 = D + E\cdot C_i; \qquad \overline{C_0} = \overline{T} + E\cdot\overline{C_i}.$$

Таким чином, схему повного двійкового суматора, побудованого на елементах **„І-НЕ"**, відповідно до останніх рівнянь можна реалізувати так (див. рис. 7.26).

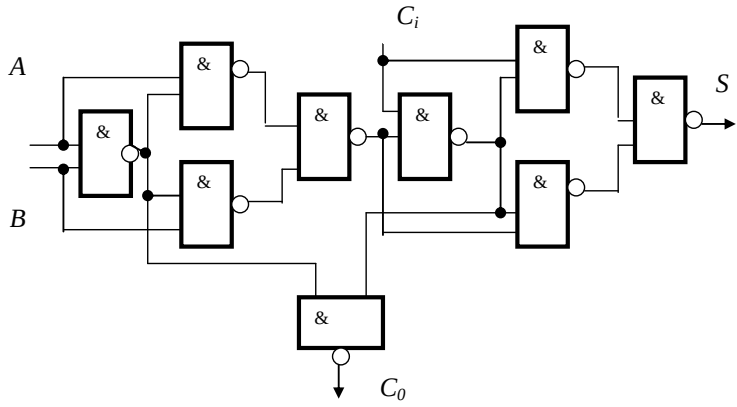


Рис. 7.26. Схема повного двійкового суматора,
побудованого на елементах „І-НЕ"

На рис. 7.27 показано логічну схему 4-розрядного двійкового лічильника. Цей лічильник лічить від 0 до 15, а потім 16-м імпульсом скидається в 0. На лічильний вхід кожного наступного тригера через вентилі пройде імпульс з входу лічильника лише в тому випадку, якщо всі попередні тригери перебували в стані „1”.

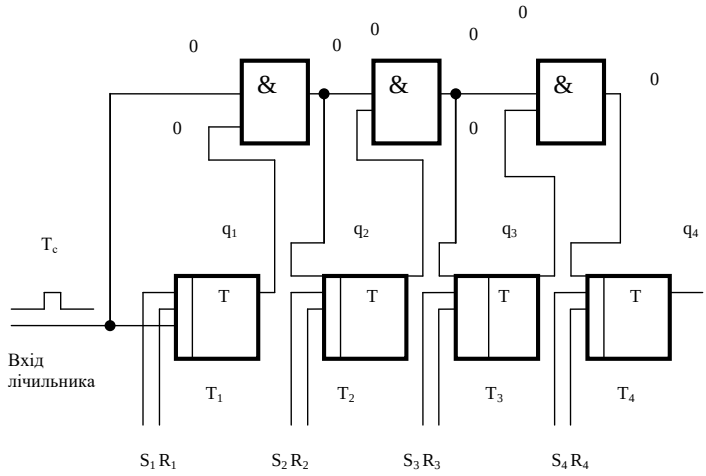


Рис. 7.27. Логічна схема двійкового лічильника

Але на практиці зазвичай використовують готові лічильники та півсуматори, реалізовані у вигляді інтегральних схем.

7.3.2. Електронні схеми з кількома виходами

Задача синтезу схеми з *n* входами й *k* виходами відрізняється від задачі синтезу *k* схем з *n* входами й одним виходом тим, що, розв’язуючи її, необхідно виключити дублювання в *k* схемах синтезованих функцій.

Прикладом схем з кількома входами й виходами може бути схема дешифратора. Принцип роботи дешифратора простий: при заданому наборі вхідних сигналів на виході збуджується один або кілька виходів відповідно до заданої залежності.

Нижче показано схему дешифратора 3-розрядного двійкового коду. Дешифратор за двійковим кодом, що надходить на вхід, вибирає один вихід, на який формує сигнал „1”. На інших виходах формується сигнал 0. У дешифратора n-розрядного двійкового коду може бути 2^n виходів. У

наведеній схемі n=3, отже 2^3=8 виходів. Наприклад, сигнал „1” на шостому виході f_6 *буде сформовано, якщо на вхід* надійшов двійковий код 010, оскільки $f = \bar{x}_1 \cdot x_2 \cdot \bar{x}_3$.

Таблицю істинності дешифратора та його логічну схему наведено нижче (див. табл. 7.6 і рис. 7.28).

Т а б л и ц я 7 . 6

Таблиця істинності дешифратора

x_1	x_2	x_3	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	1	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	1	0	0	0
1	1	1	1	0	0	0	0	0	0	0

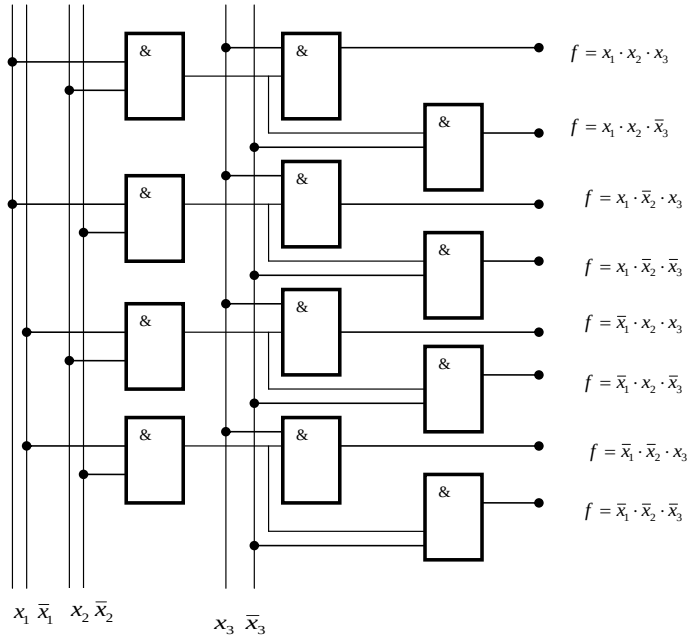


Рис. 7.28. Логічна схема дешифратора

Синтез такої схеми може бути здійснено, якщо розглядати окремо кожну вихідну функцію.

Реалізація цих виразів дає можливість створити логічну схему дешифратора.

На рис. 7.29 показано логічну схему десяткової тетради двійково-десяткового лічильника. Імпульс установлення лічильника на нульову позначку формується від кожного 10-го імпульсу, що надходить на вхід лічильника. Запис інформації в лічильник здійснюється не через лічильний вхід, і зчитування показань виконується так само, як у двійковому регістрі.

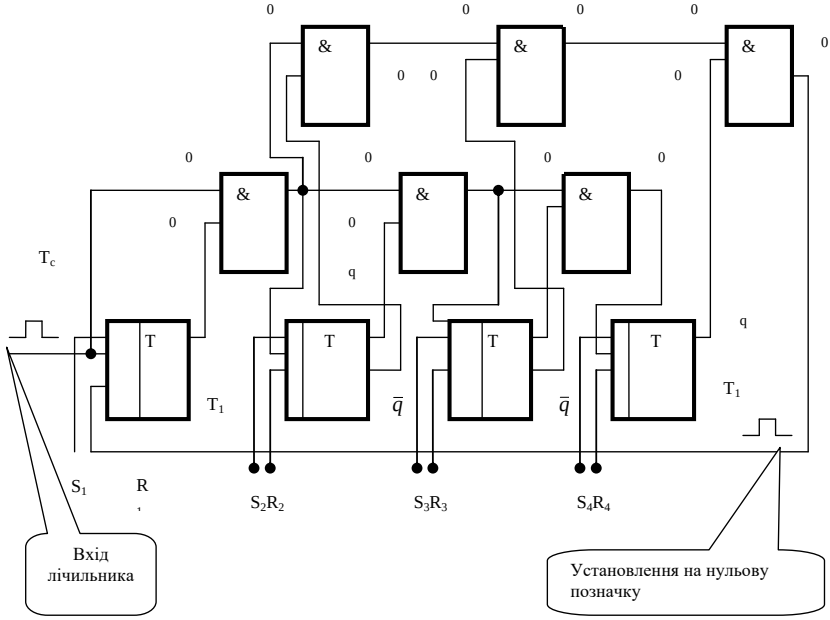


Рис. 7.29. Логічна схема тетради двійково-десяткового лічильника

7.4. Часові булеві функції та послідовні автомати

Вище ми розглянули способи аналізу й синтезу схем першого роду (комбінаційних), які неможливо використати в описі схем другого роду (схем з пам'яттю). Основна особливість схем з пам'яттю полягає в тому,

що алгоритм їхньої роботи залежить від часу *(t)*. Але час не є двійковим аргументом. Тому використовують поняття *автоматного часу*, який приймає дискретні цілочисельні значення 0, 1, 2 і т. д. Це означає, що роботу схеми з пам'яттю розбито на інтервали, протягом яких автоматний час умовно приймає постійне значення. Ці інтервали часу формуються деякими сигналами – *тактами.*

Часова булева функція (ЧБФ) – це логічна функція $y = \varphi(x_1, x_2, ..., x_n, t)$, яка приймає значення (0,1) за умови, що

$$0 \le t \le s-1,$$

де s – кількість інтервалів автоматного часу.

Кількість різноманітних часових булевих функцій дорівнює 2^{s2^n}. Дійсно, якщо функція часу приймає значення t = 0, 1, 2, ..., s-1 і кожному інтервалу часу відповідає 2^n різних двійкових наборів, то завжди буде $s2^n$ різних наборів. Отже, загальна кількість *ЧБФ* дорівнює 2^{s2^n}.

Будь-яка часова булева функція може бути представлена у вигляді

$$y = \varphi(x_1, x_2, ..., x_n, t) = \varphi_0 \tau_0 \vee \varphi_1 \tau_1 \vee \ldots \vee \varphi_{s-1} \tau_{s-1}, \qquad (7.1)$$

де φ_i – кон'юнктивний або диз'юнктивний терм від аргументів $(x_1, x_2, ..., x_n)$;

τ_i – допоміжна функція, яка приймає значення $\tau_i = (0,1)$ в момент часу t_i.

Наведена форма представлення часових логічних функцій дозволяє застосувати до функції y всі методи спрощення й мінімізації, розглянуті раніше.

Наприклад, перетворимо функцію, задану таблицею 7.7, у вигляд (7.1).

Таблиця 7.7

Таблиця істинності

x_1	x_2	t	$\varphi(x_1, x_2, t)$	x_1	x_2	t	$\varphi(x_1, x_2, t)$
0	0	0	0	1	0	1	1
0	1	0	0	1	1	1	0
1	0	0	1	0	0	2	0
1	1	0	0	0	1	2	0
0	0	1	0	1	0	2	1
0	1	1	1	1	1	2	1

Функцію $y=\varphi(x_1, x_2, t)$ представимо як сукупність трьох логічних функцій $\varphi_0(x_1, x_2)$; $\varphi_1(x_1, x_2)$; $\varphi_2(x_1, x_2)$, які для таблиці 7.7 мають вигляд:

$$\varphi_0(x_1, x_2) = x_1\bar{x}_2; \quad \varphi_1(x_1, x_2) = \bar{x}_1 x_2 \vee x_1\bar{x}_2; \quad \varphi_2(x_1, x_2) = x_1\bar{x}_2 \vee x_1 x_2 = x_1.$$

На підставі рівняння (7.1) записуємо остаточний вигляд часової логічної функції:

$$y = x_1\bar{x}_2\tau_0 \vee (\bar{x}_1 x_2 \vee x_1\bar{x}_2)\tau_1 \vee x_1\tau_2.$$

Слід відзначити, що вираз (7.1) можна застосувати лише до періодичних часових функцій. Перехід до схеми від логічного виразу (7.1) можна здійснити так.

Припустимо, що на виходах деякої схеми (дешифратора) в моменти часу t з'являються сигнали:

якщо $t_1 = 0$, то на виході 1 сигнал $\tau_0 = 1$, при $\tau_1 = 0$, $\tau_2 = 0$;
якщо $t_2 = 1$, то на виході 2 сигнал $\tau_1 = 1$, при $\tau_0 = 0$, $\tau_2 = 0$;
якщо $t_3 = 2$, то на виході 3 сигнал $\tau_2 = 1$, при $\tau_0 = 0$, $\tau_1 = 0$.

Для кожної функції φ_i необхідно побудувати відповідну логічну схему, яка не залежить від аргументу t. Після цього всі схеми з'єднуємо між собою відповідно до (7.1).

Рекурентна булева функція (РБФ) – логічна функція, яка залежить і від поточних значень x_{t_i} вхідних аргументів, і від попередніх значень самої функції $y_{(t-1)}$. Повний аналітичний запис такої функції має такий вигляд:

$$y_t = \varphi(x_{t_1}, x_{t_2}, ..., x_{t_n}, y_{t-1}, y_{t-2}, ..., y_{t-k}),$$

$$y_t = \{0, 1\} \text{ при } t > 0,$$

де x_{t_i} – поточні значення вхідних аргументів;

y_j – значення вихідних функцій у момент часу $j = t\text{-}1; t\text{-}2$ і т. д.

Введемо поняття елемента затримки *(D)*, для якого справедливою є рівність $y_{t+1} = x_i$ тобто значення вихідного сигналу в момент часу $t+1$ дорівнює значенню вхідного сигналу в момент часу t. Отже, $D(t)$ є його логічним оператором [7, 14, 22-25, 27, 31,33 та ін.].

Тепер розглянемо логічну схему, яка має ланцюг зворотного зв'язку з включеною в неї схемою затримки (див. рис. 7.30):

Припустимо, що як схему з функцією *f(x,y)* взято логічну схему **„АБО”**. Тоді наша схема працює так:

$$f(x,y) = x_{t+1} \vee y_t.$$

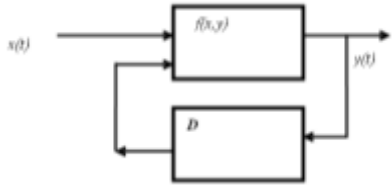


Рис. 7.30. Логічна схема

У цій схемі вихідний сигнал залежить і від вхідного сигналу в певний момент часу, і від вихідного сигналу в попередній момент часу.

Таким чином, будь-яка рекурентна булева функція може бути реалізована за допомогою набору логічних операторів функціональних елементів, які представляють звичайні функції алгебри логіки, і операторів схем затримки.

Замість схеми затримки у зворотному зв’язку може бути включено запам’ятовуючий елемент, наприклад, тригер чи групу тригерів.

Тому справедливим є таке твердження: будь-яку схему з пам’яттю можна представити у вигляді сукупності схем одного з розглянутих раніше базисів і тригерів.

7.5. Типова методика проектування комбінаційних логічних схем у прикладах

Такий підхід можна використовувати й для інших вхідних умов. Наприклад якщо б *x* повинен був мати високий рівень лише за умови, що *A=1* і *B=0, то в схемі містився б елемент* **„І”** для вхідних сигналів A і $\overline{B}$. Тобто для кількох з чотирьох можливих умов на вході можна отримати високий рівень сигналу *x,* подаючи на елемент **„І”** такі вхідні сигнали, які дозволяють отримати потрібний добуток. Усі чотири різні ситуації показано на рисунку. Кожен з елементів **„І”** виробляє вихідний сигнал, який дорівнює „1” лише при одному вхідному положенні й дорівнює „0” при всіх інших положеннях (див. рис. 7.31).

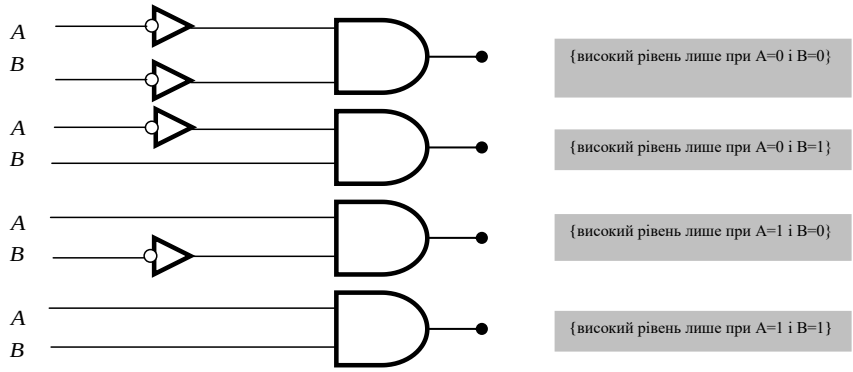
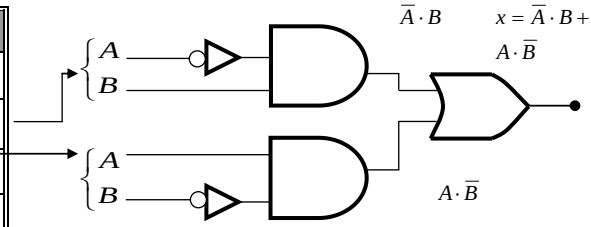


Рис. 7.31. Логічна схема з використанням міжнародних позначень ЛЕ

Розглянемо другий приклад. На рис. 7.32, б наведено логічну схему. Таблиця істинності (див. рис. 7.32, а) показує, що вихідний сигнал *x* дорівнює „1” у двох випадках: *A=0, B=1* і *A=1, B=0*. Як можна реалізувати таку схему? Відомо, що добуток $\overline{A} \cdot B$ дасть 1, лише якщо *A=0, B=1*, а добуток $A \cdot \overline{B}$ дорівнює 1 лише за умови, що *A=1, B=0*. Потрібний вихідний сигнал *x* повинен мати *високий* рівень у *будь-якому* випадку, тому всі члени виразу треба скласти за допомогою елемента **„АБО”**, щоб отримати *x*. Це показано на рис. 7.32, б, де кінцевий вираз для вихідного сигналу дорівнює $x = \overline{A} \cdot B + A \cdot \overline{B}$.

A	*B*	*x*
0	0	0
0	1	1
1	0	1
1	1	0



а б

Рис. 7.32. Таблиця істинності й логічна схема з використанням міжнародних позначень ЛЕ

У цьому прикладі кожний потрібний добуток отримуємо лише за таких наведених у таблиці початкових умов, за яких x=1. Потім вихідні сигнали елемента **„І”** складаються, щоб отримати вихідний сигнал *x,* який дорівнює „1”, лише якщо кожний вхідний сигнал елемента **„І”** теж відповідає „1”. Та сама процедура може бути розширена й на приклади з великою кількістю входів. Розглянемо таблицю істинності для схеми з трьома входами (табл. 7.8).

Таблиця 7.8

Таблиця істинності

A	*B*	*C*	*x*	
0	0	0	0	
0	0	1	0	
0	1	0	1	$\rightarrow \overline{A}B\overline{C}$
0	1	1	1	$\rightarrow \overline{A}BC$
1	0	0	0	
1	0	1	0	
1	1	0	0	
1	1	1	1	ABC

Є три випадки, коли вихідний сигнал x = 1. У таблиці показано потрібні добутки **„І”** для кожного з цих випадків. Зазначимо, що в кожному випадку, коли сигнал дорівнює 0, вона інвертується перед входом на елемент **„І”**. Диз’юнктивну форму для x отримано додаванням за допомогою елемента **„АБО”** всіх трьох добутків:

$$x = \overline{A} \cdot B \cdot \overline{C} + \overline{A} \cdot B \cdot C + A \cdot B \cdot C.$$

Як тільки з таблиці істинності визначено вираз для вихідного сигналу в диз’юнктивній формі, можна легко реалізувати логічну схему, використовуючи елементи **„І”** та **„АБО”**, а також інвертори. Однак звичайний вираз можна спростити, що дасть можливість побудувати більш ефективну схему. Наступний приклад демонструє повну методику проектування.

Спроектуємо логічну схему з трьома входами *A, B* і *C,* виходи якої матимуть *високий* рівень, лише якщо на більшій частині входів також наявний *високий* рівень.

Етап 1 – складання таблиці істинності.

Відповідно до умови цієї задачі вихідний сигнал x має дорівнювати „1”, якщо „1” дорівнюють два чи більше вхідних сигналів; для всіх інших випадків вихідний сигнал має дорівнювати „0” (табл. 7.9).

Етап 2. Записати добуток для кожного випадку, коли вихідний сигнал дорівнює „1”.

Усього можливо чотири такі випадки. Добутки для кожного з них показано поруч з таблицею істинності (табл. 7.9). Кожен добуток, представлений у табл. 7.9, містить усі вхідні сигнали або в прямій, або в інверсійній формі.

Таблиця 7.9

Таблиця істинності

A	B	C	x	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\rightarrow \overline{A} \cdot B \cdot C$
1	0	0	0	
1	0	1	1	$\rightarrow A \cdot \overline{B} \cdot C$
1	1	0	1	$\rightarrow A \cdot B \cdot \overline{C}$
1	1	1	1	$\rightarrow A \cdot B \cdot C$

Етап 3. Записати вираз для вихідного сигналу в диз'юнктивній формі.

$$x = \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C.$$

Етап 4. Спростити вихідний вираз.

Цей вираз можна спростити кількома способами. Ось найшвидший з них: оскільки член $A \cdot B \cdot C$ містить два спільні аргументи з будь-яким з інших членів виразу, можна застосувати $A \cdot B \cdot C$, згрупувавши його з кожним іншим членом. Вираз, переписаний після використання члена $A \cdot B \cdot C$ тричі, має такий вигляд:

$$x = \overline{A} \cdot B \cdot C + A \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C.$$

Винесемо за дужки попарно спільні члени й отримаємо:

$$x = B \cdot C \cdot \left(\overline{A} + A\right) + A \cdot C \cdot \left(\overline{B} + B\right) + A \cdot B \cdot \left(\overline{C} + C\right).$$

Усі члени в дужках у цьому виразі дорівнюють „1", а значить маємо

$$x = B \cdot C + A \cdot C + A \cdot B.$$

Етап 5. Побудувати схему, яка реалізує остаточний вираз. Таку схему реалізовано на рис. 7.33. Вираз для неї записано в диз'юнктивній формі, тому схема містить кілька елементів **„І"**, сигнали з яких надходять на елемент **„АБО"**.

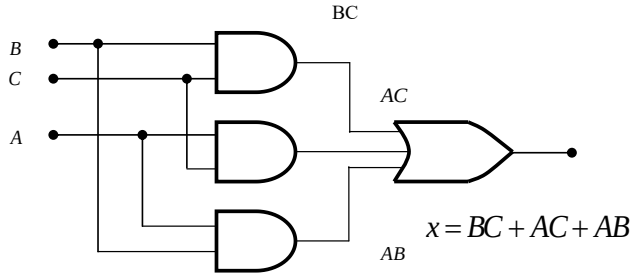


Рис. 7.33. Логічна схема з використанням міжнародних позначень ЛЕ

Розглянемо ще один приклад [30]. На рис. 7.34 а, зображено аналого-цифровий перетворювач, який відстежує постійну напругу акумуляторної батареї на 12 В. Вихідний сигнал перетворювача – двійкове число $A{\cdot}B{\cdot}C{\cdot}D$ із чотирьох біт, що відповідає напрузі батареї з кроком 1 В, де A – старший значущий біт. Двійкові вихідні сигнали перетворювача потрапляють на логічну схему, яка виробляє *високий* рівень сигналу на виході, якщо двійкове значення, яке надійшло, більше, ніж $0110_2 = 6_{10}$; тобто напруга батареї більша, ніж 6 В. Треба побудувати логічну схему.

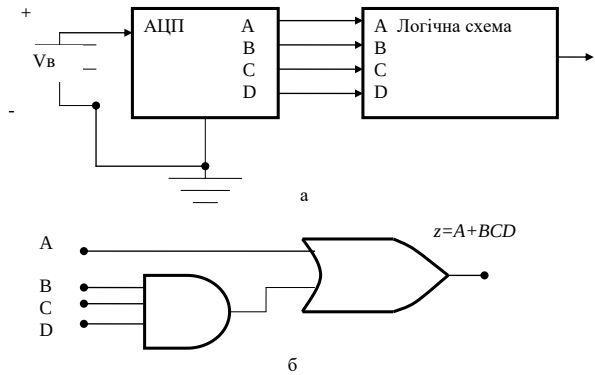


Рис. 7. 34. Схема аналого-цифрового перетворювача
та його комбінаційна схема з використанням міжнародних позначень ЛЕ

У таблиці істинності (див. табл. 7.10) для кожного набору вхідних аргументів показано десятковий еквівалент двійкового числа, яке представлено комбінацією $A \cdot B \cdot C \cdot D$.

Таблиця 7.10

Таблиця істинності

	A	B	C	D	z	
0	0	0	0	0	0	
1	0	0	0	1	0	
2	0	0	1	0	0	
3	0	0	1	1	0	
4	0	1	0	0	0	
5	0	1	0	1	0	
6	0	1	1	0	0	
7	0	1	1	1	1	$\rightarrow \overline{A} \cdot B \cdot C \cdot D$
8	1	0	0	0	1	$\rightarrow A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}$
9	1	0	0	1	1	$\rightarrow A \cdot \overline{B} \cdot \overline{C} \cdot D$
10	1	0	1	0	1	$\rightarrow A \cdot \overline{B} \cdot C \cdot \overline{D}$
11	1	0	1	1	1	$\rightarrow A \cdot \overline{B} \cdot C \cdot D$
12	1	1	0	0	1	$\rightarrow A \cdot B \cdot \overline{C} \cdot \overline{D}$
13	1	1	0	1	1	$\rightarrow A \cdot B \cdot \overline{C} \cdot D$
14	1	1	1	0	1	$\rightarrow A \cdot B \cdot C \cdot \overline{D}$
15	1	1	1	1	1	$\rightarrow A \cdot B \cdot C \cdot D$

Вихідний сигнал z дорівнює „1” для тих випадків, коли двійкове число більше 0110. В усіх інших випадках сигнал z дорівнює „0”. Таблиця істинності представляє в диз’юнктивній формі такий вираз:

$$z = \overline{A} \cdot B \cdot C \cdot D + A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} + A \cdot \overline{B} \cdot \overline{C} \cdot D + A \cdot \overline{B} \cdot C \cdot \overline{D} \cdot + A \cdot \overline{B} \cdot C \cdot D +$$
$$A \cdot B \cdot \overline{C} \cdot \overline{D} + A \cdot B \cdot \overline{C} \cdot D + A \cdot B \cdot C \cdot \overline{D} + A \cdot B \cdot C \cdot D.$$

Спростити цей вираз видається неймовірною задачею, однак крок за кроком процес втягуватиме в себе все нові й нові спільні множники, і члени типу $A + \overline{A}$ скорочуватимуться:

$$z = \overline{A} \cdot B \cdot C \cdot D + A \cdot \overline{B} \cdot \overline{C} \cdot \left(\overline{D} + D\right) + A \cdot \overline{B} \cdot C \cdot \left(\overline{D} + D\right) + A \cdot B \cdot \overline{C} \cdot \left(\overline{D} + D\right) + A \cdot B \cdot C \cdot \left(\overline{D} + D\right) =$$
$$= \overline{A} \cdot B \cdot C \cdot D + A \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C =$$
$$= \overline{A} \cdot B \cdot C \cdot D + A \cdot \overline{B} \cdot \left(\overline{C} + C\right) + A \cdot B \cdot \left(\overline{C} + C\right) =$$

$= \overline{A} \cdot B \cdot C \cdot D + A \cdot \overline{B} + A \cdot B =$

$= \overline{A} \cdot B \cdot C \cdot D + A \cdot \left(\overline{B} + B\right) =$

$= \overline{A} \cdot B \cdot C \cdot D + A.$

Останній вираз можна ще спростити (Теорема $A + \overline{A}B = A + B$).

$$z = \overline{A} \cdot B \cdot C \cdot D + A = B \cdot C \cdot D + A.$$

Остаточний вираз реалізовано в схемі на рис. 7.34 б.

Наведений вище приклад показує відносну громіздкість методу алгебраїчних спрощень, особливо коли початковий вираз містить велику кількість членів. Це обмеження не стосується, однак, методу карт Карно.

Розглянемо ще один приклад. Раніше ми розглянули схему додавання цілих багаторозрядних чисел (див. рис. 3.1). Ця схема скоріше ілюструє принцип роботи суматора, але не дозволяє перейти до стадії його розробки на логічному рівні.

Розглянемо приклад, у якому спробуємо спроектувати логічну схему п'ятибітового суматора, який використовує окремі повні суматори. Сума з'являється на виходах S_4, S_3, S_2, S_1, S_0.

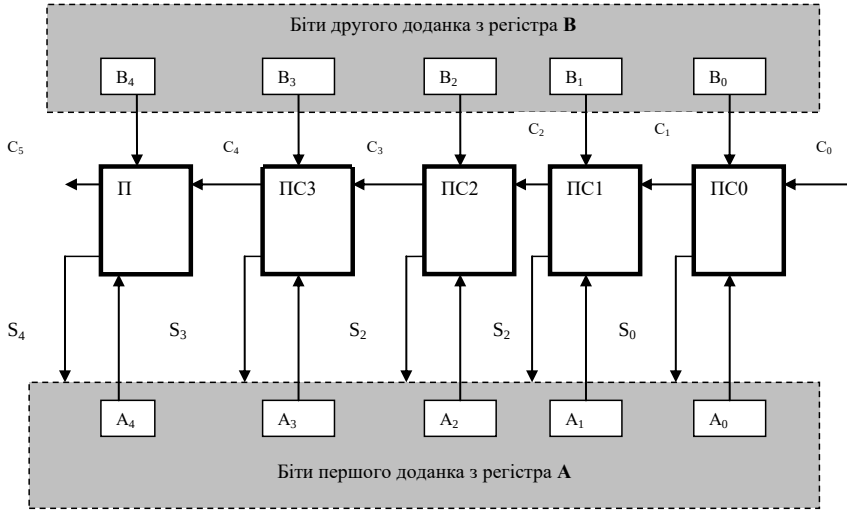


Рис. 7.35. Блок-схема паралельного п'ятибітового суматора, який використовує окремі повні суматори

Пристрій називається паралельним суматором, тому що всі біти чисел, які додаються, подаються на входи схеми **одночасно.** Це означає, що в кожному розряді додавання виконується за один такт. Це відрізняється від звичного додавання на папері, коли ми складаємо біти (цифри) по черзі, отже, паралельне додавання виконується швидше.

Нижче наведено таблицю істинності пристрою, який має три входи: *A, B,* C_{in} , а також два виходи: *S* і C_{out}. Усього для трьох входів можливі вісім наборів станів, при цьому для кожного конкретного випадку в таблиці істинності й на рис. 7.36 указано можливі вихідні сигнали [14,15,30].

Таблиця 7.11

Таблиці істинності повного суматора

Вхід бітів першого доданка	**Вхід бітів другого доданка**	**Вхід бітів переносу**	**Вихід бітів суми**	**Вихід бітів переносу**
A	**B**	**$C_{вх}$**	**S**	**$C_{вих}$**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

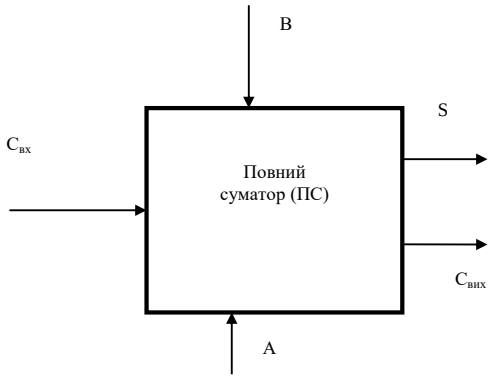


Рис. 7.36. Вихідні сигнали

Розглянемо, наприклад, випадок, коли $A=0$, $B=0$, $C_{вх}=1$. Повний суматор має скласти ці біти й отримати суму S, що дорівнює 0, і перенос $C_{вих}$, що дорівнює 1.

Схема має два виходи, тому можна проектувати логіку для кожного виходу окремо. Почнемо з виходу S. Таблиця істинності показує, що можливі чотири ситуації, коли $S=1$. Можна записати вираз для S, використовуючи диз'юнктивну форму:

$$S = \overline{A} \cdot \overline{B} \cdot C_{вх} + \overline{A} \cdot B \cdot \overline{C}_{вх} + A \cdot \overline{B} \cdot \overline{C}_{вх} + A \cdot B \cdot C_{вх}.$$

Спробуємо спростити цей вираз, тобто винесемо за дужки спільні множники. Оскільки жодна пара доданків не має двох спільних аргументів, винесемо за дужки з перших двох членів множник $\overline{A}$, а з двох останніх – A. У результаті отримаємо:

$$S = \overline{A} \cdot (\overline{B} \cdot C_{вх} + B \cdot \overline{C}_{вх}) + A \cdot (\overline{B} \cdot \overline{C}_{вх} + B \cdot C_{вх}).$$

Перший член, записаний у дужках, є вираженням **„АБО**, **яке виключає**", для аргументів B *и* $C_{вх}$. Його можна записати у вигляді $B \otimes C_{вх}$. Другий член у дужках – це операція **„АБО-НЕ**, **що виключає**", для тих самих сигналів B *и* $C_{вх}$. Його можна записати $\overline{B \otimes C_{вх}}$. Тоді вираз для S набуде такого вигляду:

$$S = \overline{A} \cdot (B \otimes C_{вх}) + A \cdot (\overline{B \otimes C_{вх}}).$$

Замінимо $B \otimes C_{вх}$ на X. Тоді $S = \overline{A} \cdot X + A \cdot \overline{X} = A \otimes X$, а це буде просто операція **„АБО**, **що виключає**". Проведемо зворотну заміну $S = A \otimes [B \otimes C_{вх}]$. Тепер розглянемо сигнали на виході $C_{вих}$, наведені в таблиці істинності. Запишемо вираз для сигналу $C_{вих}$ в диз'юнктивній формі:

$$C_{вих} = \overline{A} \cdot B \cdot C_{вх} + A \cdot \overline{B} \cdot C_{вх} + A \cdot B \cdot \overline{C}_{вх} + A \cdot B \cdot C_{вх}.$$

Спростимо цей вираз, виносячи множники за дужки

$$C_{вих} = B \cdot C_{вх} \cdot (\overline{A} + A) + A \cdot C_{вх} \cdot (\overline{B} + B) + A \cdot B \cdot (\overline{C}_{вх} + C_{вх}) = B \cdot C_{вх} + A \cdot C_{вх} + A \cdot B.$$

Подальша мінімізація отриманого виразу неможлива. Схема, отримана в результаті, матиме такий вигляд (див. рис. 7.37).

Вираз для сигналів на виходах S і $C_{вих}$ можна мінімізувати, використовуючи карти Карно (див. рис. 7.38).

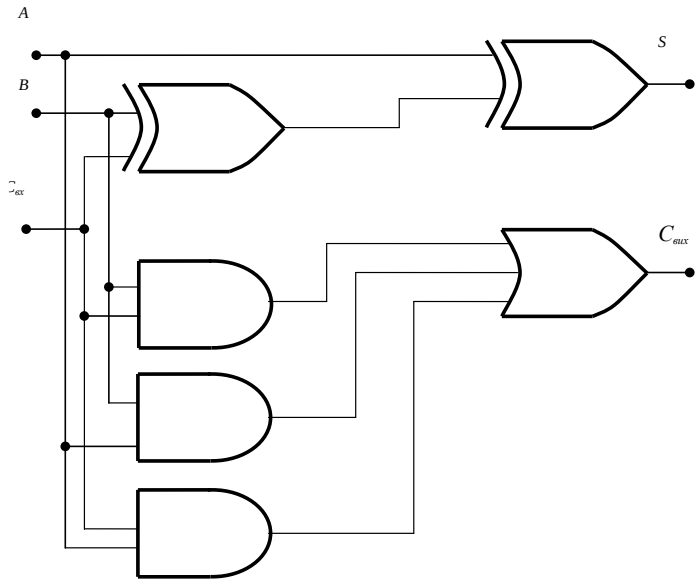
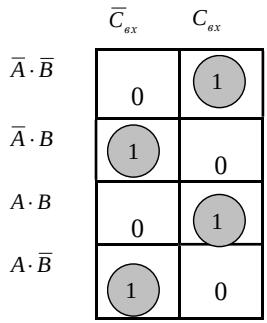


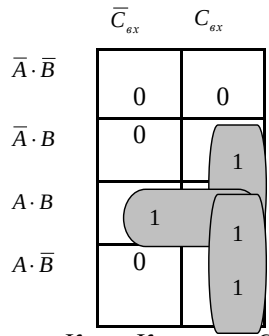
Рис. 7.37. Логічна схема повного суматора з використанням міжнародних позначень ЛЕ



Карта Карно для S

$$S = \overline{A}\cdot\overline{B}\cdot C_{вх} + \overline{A}\cdot B\cdot\overline{C}_{вх} + A\cdot B\cdot C_{вх} + A\cdot\overline{B}\cdot\overline{C}_{вх}.$$

а



Карта Карно для $C_{вих}$

$$C_{вих} = B\cdot C_{вх} + A\cdot C_{вх} + A\cdot B.$$

б

Рис. 7.38. Карти Карно для вихідних сигналів повного суматора

На рис. 7.38 а показано карту Карно для виходу *S*. Ця карта не має гнізд з одиницями, які прилягають одне до одного, тому їх неможливо згрупувати, щоб отримати квартети або навіть пари, тобто вираз для виходу *S* не може бути мінімізований. Карту Карно для виходу $C_{вих}$ показано на рис 7.38 б. Три пари, які можна згрупувати, дозволяють отримати вираз, аналогічний алгебраїчно отриманому виразу.

Розглянемо найпростіші приклади з використанням тригерів.

На рис. 7.39 б показано спрощене позначення засувки. Тут виходи S (SET) і C (CLEAR) відповідають входам установки й скидання, а кола на входах показують, що ці входи управляються *низьким* рівнем. У подальшому таке позначення буде застосовано для засувки на елементах **„І-НЕ”.**

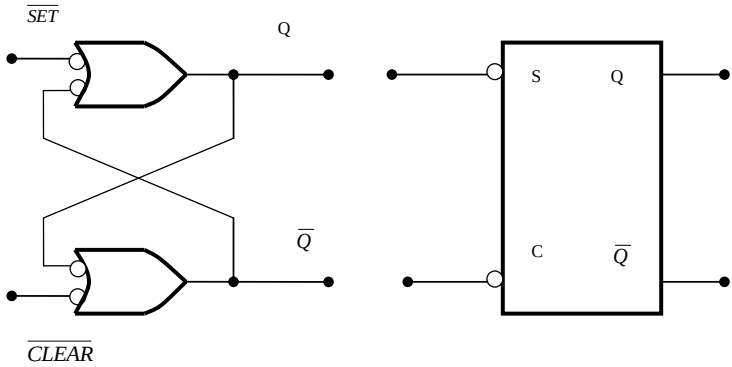


Рис. 7.39. Схема тригера
а - еквівалентне позначення засувки на елементах **„І-НЕ”**;
б - спрощене позначення

Скидання тригера або засувки іноді також називається *очисткою*. Фактично, вхід скидання можна також називати *входом очистки*.

Приклад. Сигнали, наведені на рис 7.40 а, подаються на входи засувки, зображені на рисунку, наведеному вище. Нехай у початковий момент часу *Q=0*. Визначте форму вихідного сигналу.

Рішення. У початковий момент часу сигнали $\overline{SET} = \overline{CLEAR} = 1$, Так що Q залишається в нульовому стані. Імпульс з *низьким* рівнем, який надходить на вхід скидання $\overline{CLEAR}$ у момент часу T_1, не справляє на вхід ніякого впливу, оскільки Q уже перебуває в скинутому (нульовому) стані.

Єдиний спосіб, яким можна перевести Q в одиничний стан, – це подати імпульс з низьким рівнем на вхід установки $\overline{SET}$ (установка із інверсією). Такий імпульс надходить у момент часу T_2, і вхід $\overline{SET}$ переходить у нульовий стан. Коли він повертається в одиничний стан у момент T_3, Q все одно залишиться в новому одиничному стані. У момент T_4, коли вхідний сигнал $\overline{SET}$ знову переходить у нульовий стан, це ніяк не змінює стан Q, оскільки Q уже встановлено в одиничний стан.

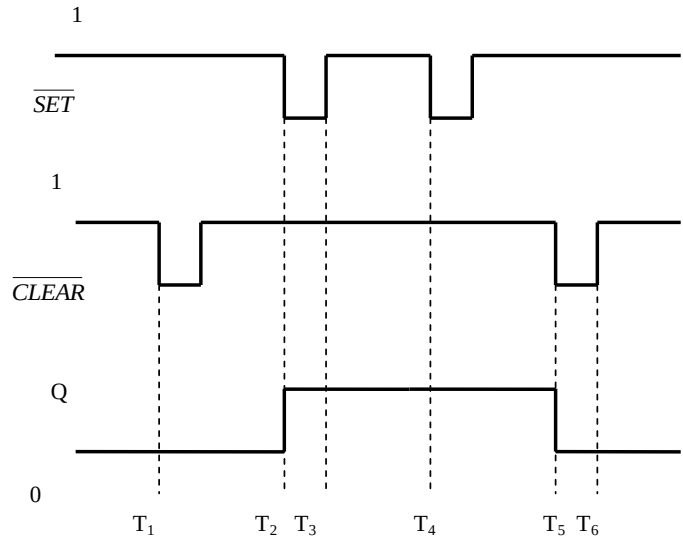


Рис. 7.40. Форма вихідного сигналу тригера

Єдиний спосіб перевести Q назад у нульовий стан – подати імпульс з низьким рівнем на вхід $\overline{CLEAR}$ (скидання із інверсією). Такий імпульс надходить у момент часу T_5. коли вхідний сигнал $\overline{SET}$ повертається в одиничний стан у момент T_6, Q все одно залишається в новому нульовому стані.

Цей приклад показує, що вихід засувки „запам'ятовує" попередній стан на вході й не змінює його, поки на вхід не надійде протилежний стан.

Приклад. Використовуючи механічний комутатор (ключ), практично неможливо отримати „чистий" перехід напруги від одного рівня до іншого. Цьому заважають випадкові зміни напруги. На рис. 7.41 показано, що при переключенні комутатора з положення 1 в положення 2 такі зміни напруги (контакт замикається й розривається в положенні 2

кілька разів) виникають на виході перед тим, як ключ остаточно перейде в положення 2.

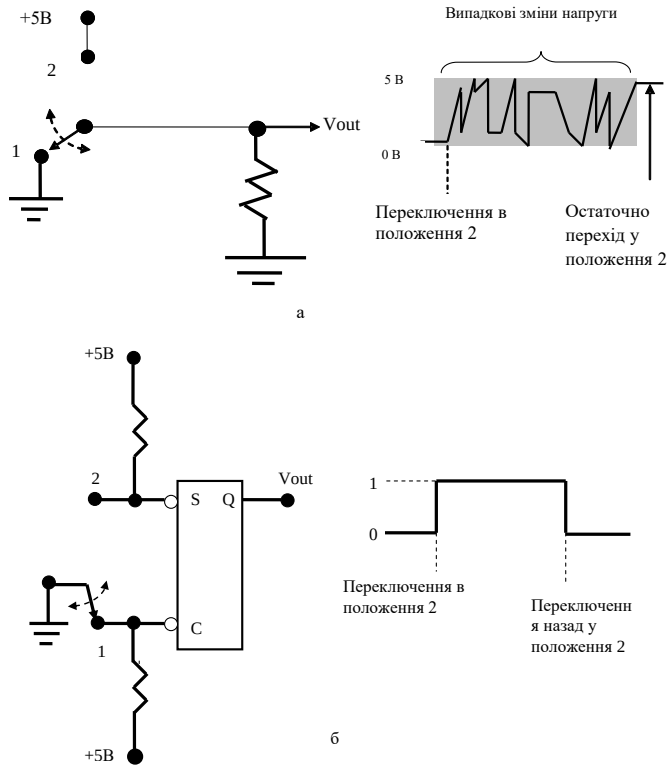


Рис. 7.41. Приклад цифрового перемикача

На рис. 7.41 а, видно, що випадкові зміни напруги механічного переключення контакту дає велику кількість стрибків рівнів напруги. Засувку на елементах **„І-НЕ”** використовують для встановлення брязкоту під час переключення механічного ключа [30].

Множинні стрибки напруги на виході зазвичай тривають не більше кількох мілісекунд, але вони можуть виявитися неприйнятними в багатьох практичних застосуваннях. Для попередження брязкоту на виході схеми використовують засувку на елементах **„І-НЕ” [**14, 20, 22-25 та ін.**]**.

Рішення. Припустимо, що ключ знаходиться в положенні 1, так що на вході скидання із інверсією ($\overline{CLEAR}$) є *низький* рівень напруги й Q = 0.

Коли ключ перейде в положення 2, з'явиться перший контакт і на вході $\overline{CLEAR}$ буде *високий* рівень напруги, а на вході установки із інверсією ($\overline{SET}$) – *низький.* Це установить вихід Q = 1 протягом кількох наносекунд (часу відгуку елемента **„І-НЕ"**). Якщо тепер ключ розімкне контакт 2 через брязкіт, на входах $\overline{SET}$ і $\overline{CLEAR}$ одночасно з'явиться *високий* рівень, що ніяк не вплине на вихід Q – він залишиться в тому самому одиничному стані. Стан виходу Q не зміниться, навіть якщо ключ брязкотітиме на контакті 2 перед остаточною установкою в новому положенні.

Точно так, якщо пересунути ключ з положення 2 назад у положення 1, на вході $\overline{CLEAR}$ при першому контакті установиться *низький* рівень. Це скине вихід Q в нульовий стан, у якому вихід залишиться, навіть якщо ключ брязкотітиме на контакті ще кілька разів.

Таким чином, на виході Q при кожному переключенні ключа з одного положення в інше спостерігається лише один перехід напруги.

У підрозділі 6.4 ішлося про програмні продукти, які дозволяють здійснювати процедуру мінімізації на комп'ютерах різноманітними методами. У деяких випадках для цього можна використовувати спеціальні математичні програми, наприклад, *Maple, MatLAB та ін.*

Наведемо простий приклад мінімізації логічної функції в системі *Maple.* Універсальна математична система *Maple* є однією з найкращих систем для науково-технічних обчислень. У середовищі *Maple* доступні понад 3000 вбудованих функцій, призначених для чисельного й символьного вирішення технічних проблем різної складності. Вона має потужні засоби для реалізації чисельних методів розрахунку, можливість виконання багатьох операцій символьної математики, зокрема із функціями алгебри логіки.

Приклад. Нехай необхідно скласти таблицю істинності функції $x \otimes (y \vee \bar{z})$.

Рішення. Загальний вигляд вирішення задачі представлено на рис. 7.42. Нижче показано рішення з коментарями.

Введення заданої функції.

```
> f:=proc(x,y,z)mo(x+diz(y,no(z)))
> end;
```

Таблиця істинності функції алгебри логіки трьох аргументів містить вісім рядків і чотири стовпці. Перші три стовпці мають стандартний вигляд, що відповідає запису чисел від 0 до 7 у двійковій системі числення.

Тому вводимо:

```
> a:=[0,0,0,0,1,1,1,1];
```

$$a := [0, 0, 0, 0, 1, 1, 1, 1]$$

```
> b:=[0,0,1,1,0,0,1,1];
```

$$b := [0, 0, 1, 1, 0, 0, 1, 1]$$

```
> c:=[0,1,0,1,0,1,0,1];
```

$$c := [0, 1, 0, 1, 0, 1, 0, 1]$$

Виведення елементів четвертого стовпця – значень заданої функції:

```
> p:=[seq(f(a[i],b[i],c[i]),i=1..8)];
```

$$p := [f(0, 0, 0), f(0, 0, 1), f(0, 1, 0), f(0, 1, 1), f(1, 0, 0), f(1, 0, 1), f(1, 1, 0), f(1, 1, 1)]$$

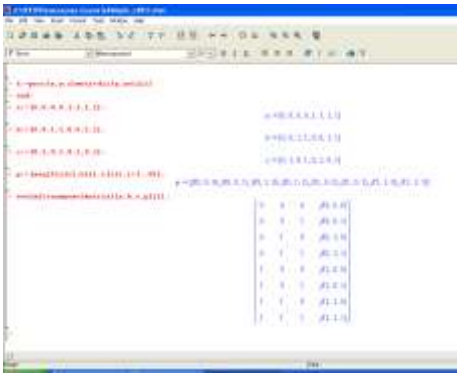


Рис. 7.42. Загальний вигляд середовища Maple

Виведення таблиці істинності:

```
> evalm(transpose(matrix([a,b,c,p])));
```

$$\begin{bmatrix} 0 & 0 & 0 & f(0, 0, 0) \\ 0 & 0 & 1 & f(0, 0, 1) \\ 0 & 1 & 0 & f(0, 1, 0) \\ 0 & 1 & 1 & f(0, 1, 1) \\ 1 & 0 & 0 & f(1, 0, 0) \\ 1 & 0 & 1 & f(1, 0, 1) \\ 1 & 1 & 0 & f(1, 1, 0) \\ 1 & 1 & 1 & f(1, 1, 1) \end{bmatrix}$$

>

Більш детально з особливостями роботи в середовищі *Maple* можна ознайомитися в спеціальній літературі.

Коротко розглянемо спеціалізовані програмні продукти, призначені не лише для мінімізації логічних функцій, але й для моделювання процесів у комбінаційних і послідовних схемах.

На рис. 7.43 показано загальний вигляд програмного комплексу *ПРОГМОЛС.* Цей комплекс дозволяє створювати й редагувати логічні схеми, здійснювати моделювання в синхронному (без урахування затримок сигналів в елементах схеми) і в асинхронному (з урахуванням затримок) режимах, а також зберігати отримані моделі у вигляді файлів на дисках. На рис. 7.43 показано модель логічної схеми, побудованої із застосуванням програмного комплексу *ПРОГМОЛС 2.0.*

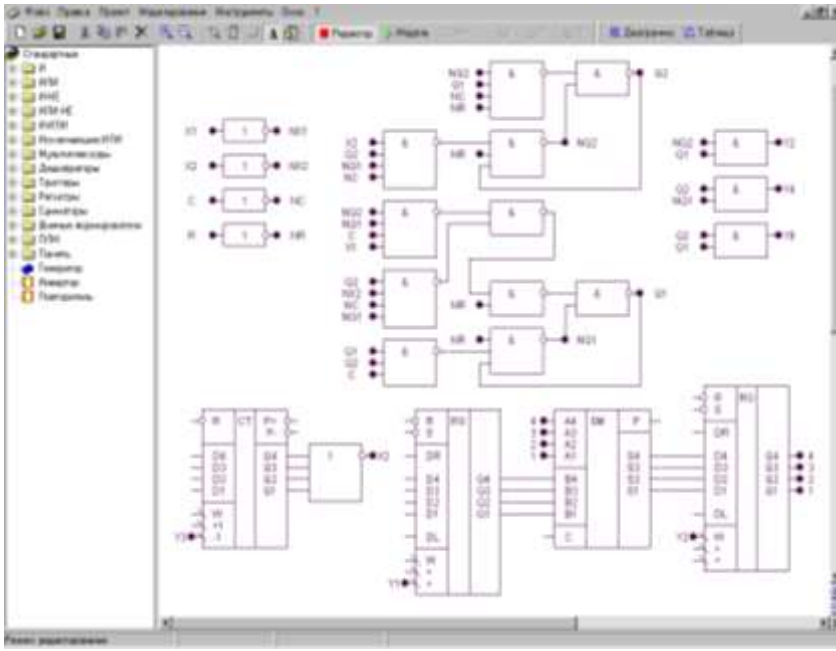


Рис. 7.43. Загальний вигляд комплексу ПРОГМОЛС 2.0

Комплекс *ПРОГМОЛС 2.0* містить систему підказок, які полегшують роботу в різних режимах моделювання.

Ще одним популярним програмним продуктом для синтезу й аналізу логічних схем, який, зокрема, дозволяє мінімізувати логічні схеми, є канадський редактор – *Electronics Workbench* (схематичний редактор).

Повний набір функцій аналізу логічних схем підвищує продуктивність проектування цифрової техніки. *Electronics Workbench* містить різноманітні зразкові бібліотечні засоби, що дозволяє скоротити час роботи над проектом. На рис. 7.44 показано інтерфейс програми *Electronics Workbench* і приклад синтезу логічної схеми.

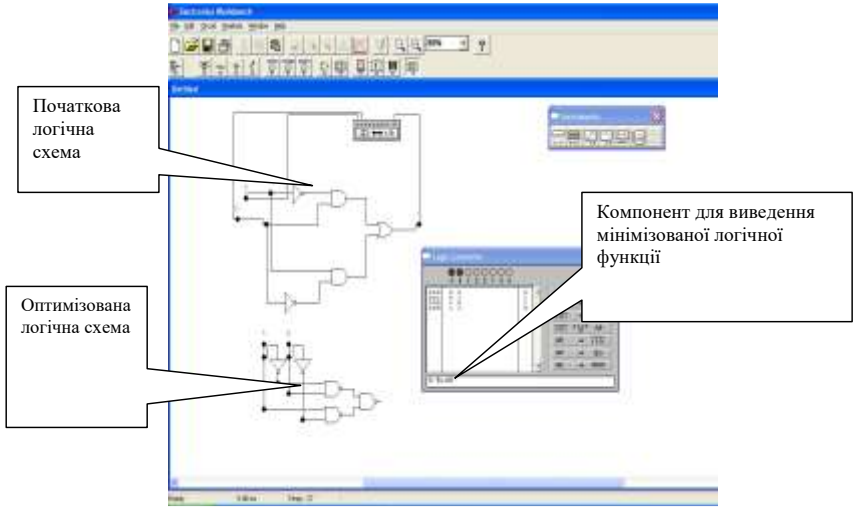


Рис. 7.44. Загальний вигляд комплексу Electronics Workbench

Запитання для самоперевірки

1. Спроектуйте логічну схему відповідно до таблиці істинності, наведеної нижче.

A	*B*	*C*	*X*
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

2. Спроектуйте логічну схему, на виході якої сигнал матиме високий рівень лише тоді, коли на більшій частині входів (*A, B, C*) буде низький рівень.

3. Спроектуйте логічну схему у базисі «**АБО-НЕ**», на виході якої сигнал матиме високий рівень лише тоді, коли на більшій частині входів (*A, B, C,D, E*) буде низький рівень.

Література для самостійної підготовки за темою:
1, 2, 6, 8, 14, 24, 27.

Розділ 8. СТРУКТУРНИЙ СИНТЕЗ ЦИФРОВИХ АВТОМАТІВ. АЛГОРИТМИ РЕАЛІЗАЦІЇ АРИФМЕТИЧНИХ ДІЙ У ЦИФРОВИХ АВТОМАТАХ

8.1. Автомати Мілі й Мура

На основі визначення цифрового автомата (див. розділ 1) наведемо наступне.

Автомат називається скінченним, якщо множина його внутрішніх станів, а також множина значень вхідних і вихідних сигналів скінченні.

Цифрові автомати можуть бути з *„жорсткою”*, або *схемною* логікою й з логікою, що зберігається в пам’яті. Розрізняють два класи автоматів – *асинхронні* й *синхронні*.

Для синхронного автомата характерним є те, що він функціонує під управлінням тактових (або синхронізуючих) сигналів з постійною тривалістю $t_{\text{ТС}}$ і постійною частотою $f_{\text{ТС}}$, якщо квантування часу рівномірне. Такт (квант) часу t_i суміщається з фронтом i-го сигналу тактових (або синхронізуючих) сигналів. Вхідні сигнали можуть впливати на автомат лише за наявності тактового сигналу й не змінюються протягом $t_{\text{ТС}}$. Період надходження тактових сигналів має бути більшим або дорівнювати часу, який необхідний реальному автомату для переходу з одного стану в інший. Коли розглядають абстрактний автомат, то вважають, що зміна внутрішніх станів автомата відбувається в інтервали часу між суміжними тактових (або синхронізуючих) сигналів, а вихідні сигнали формуються за фронтом чергового тактового сигналу.

В асинхронних автоматах інтервал часу, протягом якого залишається незмінним стан вхідних сигналів, є величиною перемінною й визначений часом, який необхідний автомату для установки відповідних вихідних сигналів і завершення переходу в новий стан. Таким чином, асинхронний автомат повинен формувати яким-небудь прийнятним способом сигнал про завершення чергового такту, за яким поточні вхідні сигнали можуть бути зняті, після чого може розпочатися наступний такт, тобто можливе надходження нових вхідних сигналів.

Для завдання скінченного автомата фіксуються три скінченних множини (алфавіти):

- множина можливих вхідних сигналів: $X = \{x_1, x_2, ..., x_m\}$;

- множина можливих вихідних сигналів: $Y = \{y_1, y_2, ..., y_k\}$;

- множина можливих внутрішніх станів автомата: $A = \{a_0, a_1, ..., a_n\}$.

На цих множинах задають два логічних оператори:

- *функцію переходів f*, яка визначає стан автомата *a(t+1)* в момент дискретного часу *t+1* залежно від стану автомата *a(t)* і значення вхідного сигналу *x(t)* в момент часу *t*:

$$a(t+1) = f[a(t), x(t)];$$

- *функцію виходів* φ, яка визначає залежність вихідного сигналу автомата *y(t)* від стану автомата *a(t)* і вхідного сигналу *x(t)* у момент часу *t*:

$$y(t) = \varphi\ [a(t), x(t)].$$

Крім того, на множині станів автомата фіксують один із внутрішніх станів a_0 як *початковий стан*.

Термін *стан автомата* використовують для опису систем, вихідні сигнали яких залежать не лише від вхідних сигналів у певний момент часу, але й від деякої передісторії, тобто сигналів, які надходили на входи системи раніше. Отже, цифрові автомати належать до послідовних схем, які мають пам'ять. Поняття стан автомата відповідає деякій пам'яті про минуле, тому введення цього поняття дозволяє усунути час як явну перемінну (тобто аргумент логічної функції) й виразити вихідні сигнали як функцію станів і вхідних сигналів.

Роботу абстрактного автомата слід розглядати щодо конкретних інтервалів часу, оскільки кожному інтервалу дискретності *t* відповідатиме свій вихідний сигнал *y(t)*. Отже, функціонування автомата розглядають через дискретні інтервали часу скінченої тривалості. В абстрактній теорії цифрових автоматів вважається, що вхідні сигнали впливають на синхронний автомат у момент початку кожного *i*-го інтервалу (кванта) часу, виділеного відповідним синхроімпульсом (тактом), а зміна внутрішніх станів автомата відбувається в інтервали часу між суміжними синхроімпульсами, коли немає впливу вхідних сигналів.

Оператори, які описують роботу автомата, зазвичай задають таблицею переходів і таблицею виходів.

У таблиці переходів показують, у який стан потрапляє автомат від того чи іншого вхідного сигналу. У таблиці виходів показують, який вихідний сигнал генерує автомат залежно від типу вхідного сигналу й поточного стану автомата.

У табл. 8.1 наведено приклад таблиці переходів і виходів деякого автомата. У клітинку таблиці переходів, що знаходиться на перетині стовпця з буквою a_i і рядка з буквою x_j, записують стан автомата, у який він переходить із стану a_i при подачі на вхід сигналу x_j. В аналогічну клітинку таблиці виходів записують вихідний сигнал y_i, формований автоматом при такому переході.

Оператори переходів і виходів можуть бути задані одною таблицею, за якою однозначно визначають переходи й виходи автомата (див. табл. 8.2).

Таблиця 8.1

Таблиця переходів і виходів автомата

Вхідний сигнал	Стан			
	a_0	a_1	a_2	a_3
x_1	a_1	a_2	a_3	a_3
x_2	a_0	a_0	a_0	a_0

Продовження табл. 8.1

Таблиця виходів автомата

Вхідний сигнал	Стан			
	a_0	a_1	a_2	a_3
x_1	y_2	y_2	y_1	y_2
x_2	y_2	y_2	y_2	y_3

Таблиця 8.2

Таблиця переходів і виходів автомата

Вихідний сигнал	Стан			
	a_0	a_1	A_2	a_3
x_1	a_1 / y_2	a_2 / y_2	a_3 / y_1	a_3 / y_2
x_2	a_0 / y_2	a_0 / y_2	a_0 / y_2	a_0 / y_3

Більшу наочність забезпечує завдання скінченних автоматів за допомогою графів або діаграм станів [7-9, 14, 22-25, 27, 30, 31].

Граф автомата складається з *вузлів*, з'єднаних *гілками*. Вузли (кола на схемі графа) ототожнюють внутрішні стани автомата. Кожну гілку графа, тобто орієнтовну лінію, стрілка якої вказує наступний стан автомата, відзначено вхідним сигналом, який викликає в автоматі відповідний цій гілці перехід, і вихідним сигналом, який виникає при цьому переході. Вхідний і відповідний йому вихідний сигнали розділяються на кресленні комою або скісною рискою. Якщо деякий вхідний сигнал не змінює стану автомата, то відповідна гілка замикається на колі (вузлі), з якого вона виходить.

Оскільки таблиця станів і граф (діаграма) станів несуть ту саму інформацію, їх можна перетворити одна в одну. Кожен стан представляють колом, а кожен елемент таблиці перетворюють у відрізок орієнтованої лінії, яка з'єднує відповідні кола. Процедура зворотного перетворення очевидна.

У наш час у класі синхронних автоматів розглядають в основному два типи автоматів: *автомат Мілі* й *автомат Мура*.

Граф автомата Мілі, заданого таблицями 8.1 – 8.2, показано на рис. 8.1. Закон функціонування автоматів Мілі може бути заданий так:

$$a(t+1) = f[a(t), x(t)];$$
$$y(t) = \varphi[a(t), x(t)],$$

де $t = 1, 2, \ldots$.

Особливість автоматів Мілі полягає в тому, що їхні вихідні сигнали в певний момент часу не залежать ні від стану автомата, ні від значення вхідного сигналу в цей же момент часу.

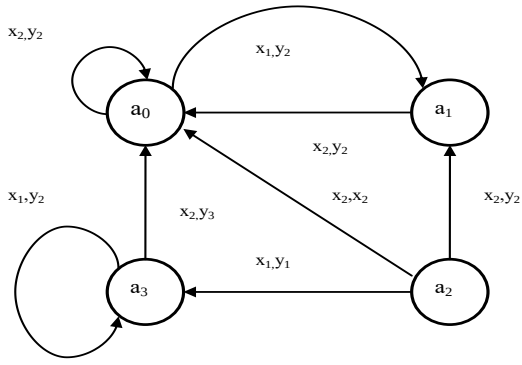


Рис. 8.1. Граф автомата Мілі

У автоматів Мура вихідні сигнали в момент часу t однозначно визначені станом автомата в цей же момент часу й у явному вигляді не залежать від значення вхідних сигналів $x_i(t)$.

Функції переходів і виходів автомата Мура, заданого на множині вхідних сигналів X, множині внутрішніх станів $A=\{a_0, a_1,\ldots,a_n\}$ і множині вихідних сигналів Y, можна записати у вигляді:

$$a(t+1) = f[a(t), x(t)];$$

$$y(t) = \varphi[a(t)].$$

Ці таблиці побудовано так само, як і таблиці переходів автомата Мілі, але над символами кожного внутрішнього стану записують вихідні сигнали, які видає автомат в цьому стані (див. табл. 8.3).

Таблиця 8.3

Таблиця переходів автомата Мура

	Вихідний сигнал					
	y_2	y_2	y_2	y_1	y_2	y_3
Вхідний сигнал	Стан					
	a_0	a_1	a_2	a_3	a_4	a_5
x_1	a_1	a_2	a_3	a_4	a_4	a_1
x_2	a_0	a_0	a_0	a_5	a_5	a_0

Граф автомата Мура, заданого цією таблицею, наведено на рис. 8.2.

На цьому рисунку стани автомата позначено символами b_i.

На графах автомата Мура значення вихідних сигналів записуються поряд з вузлами.

Між автоматами Мілі й Мура існує відповідність, яка дозволяє перетворювати закон функціонування одного з них в інший чи навпаки. Автомат Мура можна розглядати як окремий випадок автомата Мілі. При цьому необхідно пам’ятати, що послідовність станів виходів автомата Мілі випереджає на один такт послідовність станів виходів автомата Мура, тобто відмінність між автоматами Мілі й Мура полягає в тому, що в автоматах Мілі стан виходу виникає одночасно зі станом входу, який викликає його, а в автоматах Мура – із затримкою на один такт, оскільки в автоматах Мура вхідні сигнали змінюють лише стан автомата.

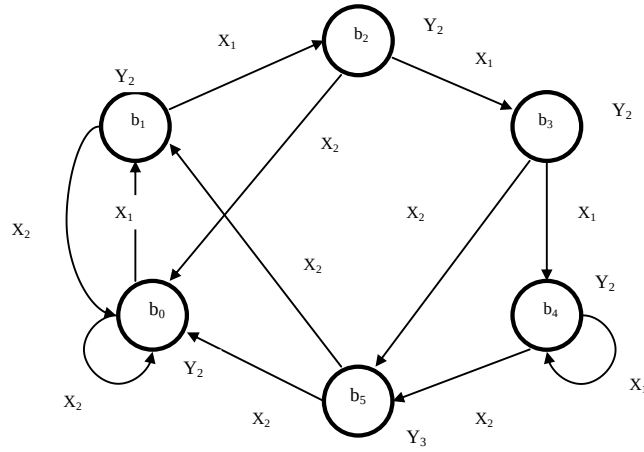


Рис. 8.2. Автомат Мура

Окрім автоматів Мілі й Мура, виділяють ще так званий *сумісний автомат – С-автомат* (див. рис. 8.3). Абстрактний *С - автомат* можна визначити як пристрій з одним входом, на який надходять вхідні сигнали *X*, і двома виходами, на яких з'являються вихідні сигнали *Y* і *U* [3, 8,18].

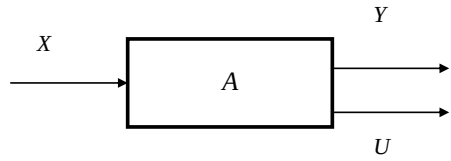


Рис. 8.3. Схема сумісного автомата

На рис 8.3. *X* – вхідний алфавіт; *A* – множина станів; *Y* – вихідний алфавіт першого типу; $U = \{u_1(t) \dots u_m(t)\}$ – вихідний алфавіт другого типу.

Відмінність *С-автомата* від автоматів Мілі й Мура полягає в тому, що він одночасно реалізує дві функції виходів φ_1 і φ_2, кожна з яких характерна для цих автоматів окремо. Цей автомат можна описати такою системою рівнянь:

$$a(t+1) = f\,[a(t), x(t)];$$
$$y(t) = \varphi_1[a(t), x(t)];$$
$$u(t) = \varphi_2[a(t)].$$

Вихідний сигнал $u=\varphi_2(a_s)$ виділяється весь час, поки автомат перебуває в стані a_s. Вихідний сигнал $y_k=\varphi_1(a_s, x_n)$ видається під час дії вхідного сигналу x_n під час перебування автомата в стані a_s. Від *С*-автомата легко перейти до автоматів Мілі або Мура (з урахуванням можливих зсувів у часі на один такт), так само як можлива трансформація автомата Мілі в автомат Мура й навпаки.

8.2. Методи структурного синтезу й мови опису цифрових автоматів

Залежно від способу задавання функцій переходів і виходів (f і φ) у наш час виділяють два класи мов опису цифрових автоматів – *початкові мови й стандартні, або автоматні мови*. У *початкових мовах* автомат описують виходячи із ситуації, коли функції переходів і виходів в явному вигляді не задано. Серед початкових мов слід виділити мову регулярних виразів алгебри подій, мову логічних схем алгоритмів, мову граф-схем алгоритмів.

Серед *автоматних мов* найбільш поширеними є таблиці переходів і виходів, а також графи, які ми розглядали раніше.

Використовують також так звані кодовані таблиці переходів і виходів, у яких визначено залежність значень у двійковому коді внутрішніх станів і вихідних сигналів від значень у цьому ж коді вхідних сигналів і внутрішніх станів елементарних автоматів у попередній момент часу.

Задача синтезу скінченних автоматів полягає в побудові складного автомата з більш простих, які називаються елементарними автоматами. На практиці в більшості випадків застосовують елементарні автомати з двома внутрішніми стійкими станами, з'єднані між собою за допомогою логічних елементів так, щоб забезпечити функціонування схеми відповідно до заданих кодованих таблиць переходів і виходів.

Для синтезу скінченних автоматів необхідно, перш за все, вибрати систему елементів, з яких повинні будуватися задані автомати. У більшості схем цифрових автоматів як елементи пам'яті застосовують елементарні автомати, які мають такі особливості:

1. Елементарні автомати, наприклад *тригери*, є автоматами Мура й мають два внутрішніх стійких стани.

2. Двом часовим станам елементарного автомата відповідають два різних вихідних сигнали, які по суті й дозволяють фізично розрізняти стани елементарних автоматів.

3. У загальному випадку елементарні автомати можуть мати кілька фізичних входів.

Для того щоб можна було побудувати схему будь-якого скінченного автомата, набір елементів, за допомогою яких проводять синтез автомата, повинен бути *функціонально повним* і містити як елементарні автомати тригери, тобто необхідно й достатньо, щоб він містив:

- хоча б один елементарний автомат з двома різними станами, для яких дотримано умови повноти системи переходів і виходів;
- логічні елементи, які утворюють функціонально повну систему для синтезу логічних комбінаційних схем.

Нижче наведено приклад загальної структурної моделі послідовної схеми, яка містить тригери (див. рис. 8.4).

Розв'язуючи задачі аналізу й синтезу схем, зазвичай розділяють послідовну схему на елементи пам'яті й комбінаційну частину.

У загальному вигляді алгоритм структурного синтезу послідовного цифрового автомата можна представити так [3, 8, 10, 22-25, 29, 31].

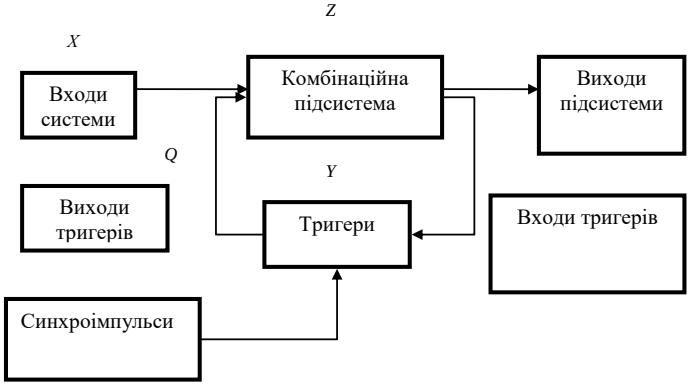


Рис. 8.4. Загальна структурна модель послідовної схеми цифрового автомату

Спочатку в чіткій словесній формі формулюють початкове завдання проекту. Залежно від типу й складності проектованого цифрового автомата наступний етап синтезу може бути різним.

Можливий варіант, коли можна одразу розпочинати формування в аналітичній формі системи рекурентних логічних функцій. Далі вибирають залежно від фізичного типу елементів функціонально повний набір логічних елементів (базис) і відповідну цьому набору систему логічних функцій. Потім рекурентні логічні функції представляються через логічні функції вибраної функціонально повної системи й мінімізують відповідно

до вибраних критеріїв. Цей етап синтезу зазвичай називають комбінаційним синтезом.

Задачі комбінаційного синтезу послідовних цифрових автоматів цілком збігаються із задачами синтезу комбінаційних логічних схем.

На цьому ж етапі розробок будується змістовна граф-схема алгоритму функціонування автомата.

Далі формують необхідні кодовані таблиці переходів і виходів і за необхідності будують граф автомата (*діаграму станів*).

Можливий варіант, коли замість формування системи рекурентних функцій спочатку складають попередній спрощений варіант граф-схеми (можливо змістовної) автомата чи граф автомата, тобто діаграми станів, і попередній варіант кодованої таблиці станів. І лише після цього формують систему рекурентних функцій і виконують усі згадані наступні процедури з поетапним уточненням граф-схеми автомата, діаграми його станів і кодованих таблиць станів (виходів, переходів і т. ін.).

Далі, орієнтуючись на сформовані таблиці й граф-схему, проектують логічну схему цифрового автомата, у якій використовують логічні елементи вибраного базису. Ця схема представляється як структурна або функціональна. Після її розробки можна переходити до створення принципової схеми розроблюваного послідовного цифрового автомата.

Слід відзначити, що в загальному випадку процедура синтезу цифрового автомата недостатньо формалізована й значною мірою залежить від реального досвіду розробника.

8.3. Елементарний автомат

Як було наведено в розділі 7, типовим прикладом елементарного автомата є тригер. Тригер має два вихідних сигнали Q і $\overline{Q}$. Сигнал Q вважається істинним, або прямим, а сигнал $\overline{Q}$ – додатковим, або інверсійним. Цим сигналам відповідає один з двох рівнів напруги – *L* чи *H*, які доповнюють один одного. Вихідні сигнали тригера постійні до тих пір, поки вони не будуть змінені під впливом вхідних сигналів, тобто тригер має два стійких стани (режими): $Q=H$ і $\overline{Q}=L$ або $Q=L$ і $\overline{Q}=H$. Перший з них, коли $Q=H$, називається станом установки, а другий – станом скидання. Припустимо, що для тригера використовують позитивну логіку. Тоді станам установки й скидання ставлять у відповідність логічні стани „1” і „0”.

Існують тригерні схеми різного типу, зокрема, типу: *D, T, RS, JK* [7, 9, 10, 14, 22, 25]. Для кожного з них є однозначна відповідність між

вхідними сигналами й відповідними переходами станів тригера. Цю відповідність задано таблицею станів. Наведемо, наприклад, умовне позначення *RS*-тригера та його таблицю станів (див. рис. 8.5 і табл. 8.4).

Тригер виявляється в стані установки, якщо на вхід *S* подано сигнал високого рівня, і в стані скидання, коли сигнал високого рівня подано на вхід *R*. Якщо на обидва входи тригера подано сигнали низького рівня, то його стан не змінюється, але якщо на обидва входи подано сигнали високого рівня, то стан тригера буде невизначеним.

Поведінку тригера, як і будь-якого іншого автомата, описують логічними таблицями станів і переходів. Наведемо такі таблиці для *RS*-тригера (див. табл. 8.5 і 8.6).

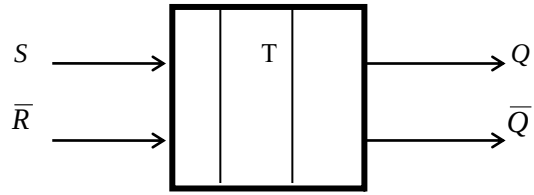


Рис. 8.5. Умовне позначення RS-тригера

Таблиця 8.4

Таблиця станів RS-тригера

	Входи			
Поточний стан	S = L R = L	S = L R = H	S = H R = L	S = H R = H
	Виходи			
Q = L $\overline{Q}$ = H	L H	L H	H L	Не визначено

Таблиця 8.5

Таблиця станів

	Входи *SR*			
Поточний стан	00	01	10	11
	Виходи			
Q = 0 Q = 1	0 1	0 0	1 1	Не визна-чено

Таблиця 8.6

Таблиця переходів

Поточний стан Q	Наступний стан $Q_н$	*S*	*R*
0	0	0	d(*)
0	1	1	0
1	0	0	1
1	1	d(*)	0

де d або * – недовизначений стан, тобто може дорівнювати або „0”, або „1”.

На рис. 8.6. наведено варіант реалізації *RS*-тригера на елементах **„І-НЕ”**.

У схемі з N тригерами (наприклад, регістр з N тригерами) стани характеризують N-розрядним двійковим словом, кожен розряд якого асоціюється з одним із тригерів. Оскільки існує 2^N різноманітних комбінацій N-розрядного слова, то в цій схемі є 2^N стійких станів. Очевидно, що стан вихідних сигналів такого регістра залежить не лише від того, на які його входи надійшли сигнали, але й від того, у якому стані були його тригери до надходження вхідних сигналів. Це і є відмінною рисою послідовних схем від схем першого роду – комбінаційних логічних схем.

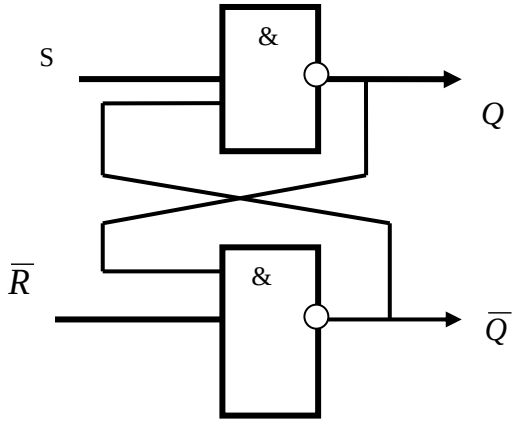


Рис. 8.6. Реалізація *RS*-тригера на елементах „І-НЕ”

Звичайний *Т*-тригер має один лічильний вхід *Т* і, як усі тригери, два виходи – *Q* і $\overline{Q}$. Він переключається лише під час зміни вхідного сигналу Т зі значення „0” на „1”.

Синхронізований *Т*-тригер має, крім лічильного входу *Т*, ще вхід СК для синхронізуючого сигналу (синхроімпульсу) СІ. Перекидання тригера відбувається в тому випадку, коли в момент надходження СІ $T = 1$, якщо $T=0$, то СІ не впливає на тригер.

8.4. Синтез цифрового автомата з пам’яттю

Розглянемо процес побудови простої послідовної схеми лічильника.

Як приклад розглянемо синтез десяткового лічильника за допомогою синхронізованих Т-тригерів. Ця схема містить чотири таких тригери, вихідні сигнали яких формують кодовий набір, інтерпретований як десяткова величина у двійково-десятковому коді. Кожен синхроімпульс збільшує значення коду, яке, досягнувши 9, переводиться наступним синхроімпульсом в 0. При переході від 9 до 0 схема видає вихідний сигнал, який указує на переповнення лічильника; цей сигнал часто називають сигналом переносу.

Крок 1 – визначення структурної моделі й використання її для ідентифікації вхідних і вихідних сигналів. На рис. 8.7 показано таку модель для десяткового лічильника. У цій схемі, крім синхросигналу, немає інших вхідних сигналів. Єдиним вихідним сигналом комбінаційної підсхеми є сигнал переносу *С*.

Якщо, крім сигналу *С*, вивести також сигнали Q_0 – Q_3 від усіх чотирьох Т-тригерів, то, використовуючи кілька таких лічильників, можна отримати зображення числа сигналів СІ, які надійшли ззовні, у двійково-десятковому коді (*Д*-коді).

У такому випадку зовнішній синхроімпульс повинен надходити на лічильник, який формує молодшу *тетраду* коду.

Для всіх інших лічильників синхроімпульсом буде сигнал *С* попереднього лічильника. Сигнал *С* лічильника, який формує старшу *тетраду*, є ознакою переповнення розрядної сітки, сформованої певною групою цих десяткових лічильників.

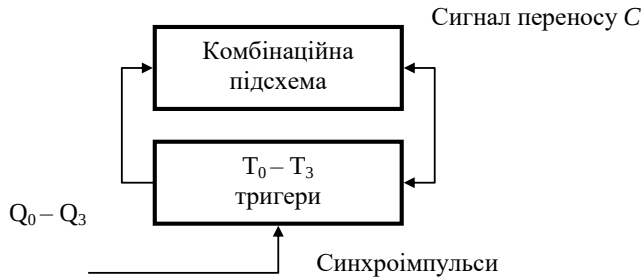


Рис. 8.7. Структурна модель десяткового лічильника

Крок 2 – побудова таблиці станів (або діаграми станів), що описує поведінку схеми. Діаграму станів для цього лічильника наведено на рис. 8.8. Згідно з діаграмою, схема має 10 допустимих станів, послідовність яких є 0000, 0001, 0010 ... ,1000, 1001, потім іде повторення 0000, 0001 і т. д. Оскільки схема не має входів, на орієнтованих лініях указано лише поточний вихідний сигнал, що відповідає кожному стану. Ці значення дорівнюють 0 для всіх допустимих станів, за винятком 1001. Таблиця 8.7 є відповідною таблицею станів. Оскільки схема не має входів, таблиця містить лише один вбудований стовпець. Відзначимо, що для шести невикористаних станів від 1010 до 1111 наступні стани не задано, їх позначено як невизначені стани. У даному випадку *d* означає *„байдуже”* (будь-який стан).

Таблиці 8.8 і 8.9 є відповідно таблицею істинності для виходу схеми *C* і розширеною таблицею істинності для входів тригерів T_3, T_2, T_1, T_0. Вихідні значення таблиці істинності й наступні стани розширеної таблиці істинності отримано послідовно за рядками з відповідних елементів таблиці станів чи відповідних орієнтованих ліній на діаграмі станів. Вихідні значення тригерів вибрано такими, щоб міг здійснюватися кожен заданий перехід станів. Для визначення цих станів було використано таблицю переходів Т-тригера, див. табл 8.8.

Крок 3 – отримання виразів для вихідного сигналу *C* і вхідних сигналів тригерів T_3, T_2, T_1 і T_0. На рис. 8.9 показано карти, використані для отримання цих мінімальних сум добутків. Логічну схему десяткового лічильника, реалізовану відповідно до цих виразів за допомогою вентилів **„І”, „АБО”, „НЕ”** і чотирьох синхронізованих Т-тригерів, наведено на рис. 8.10.

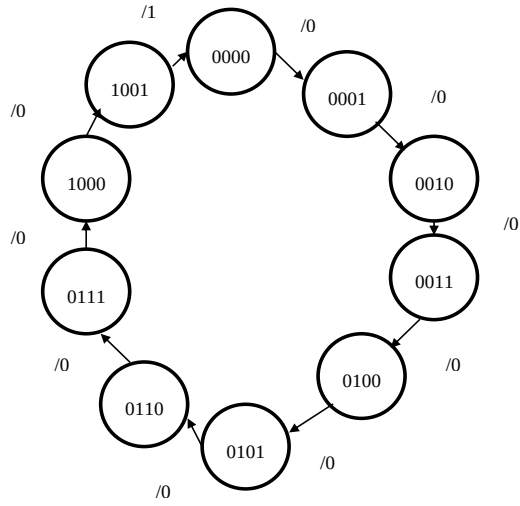


Рис. 8.8. Діаграма станів лічильника

Таблиця 8.7

Таблиця станів десяткового лічильника

Поточний стан				Наступний стан			
Q_3	**Q_2**	**Q_1**	**Q_0**	**Q_3**	**Q_2**	**Q_1**	**Q_0**
0	0	0	0	0	0	0	1/0
0	0	0	1	0	0	1	0/0
0	0	1	0	0	0	1	1/0
0	0	1	1	0	1	0	0/0
0	1	0	0	0	1	0	1/0
0	1	0	1	0	1	1	0/0
0	1	1	0	0	1	1	1/0
0	1	1	1	1	0	0	0/0
1	0	0	0	1	0	0	1/0
1	0	0	1	0	0	0	0/1
1	0	1	0	d	d	d	d/d
1	0	1	1	d	d	d	d/d
1	1	0	0	d	d	d	d/d
1	1	0	1	d	d	d	d/d
1	1	1	0	d	d	d	d/d
1	1	1	1	d	d	d	d/d

Таблиця 8.8

Таблиця переходів Т-тригера

Поточний стан Q	Наступний стан Q	N
0	0	0
0	1	1
1	0	1
1	1	0

Вивчення принципів синтезу таких вузлів цифрових автоматів, як суматори, множники, шифратори, дешифратори, лічильники, регістри та ін., є свого роду заключним етапом вивчення арифметичних і логічних основ обчислювальної техніки. Знання цих принципів, безумовно, необхідні для майбутніх фахівців з розробки й експлуатації обчислювальної техніки, а також різноманітних автоматизованих пристроїв.

На практиці, синтезуючи вузли цифрових автоматів, призначені для реалізації складних алгоритмів, треба чітко й добре знати номенклатуру, функції, які можуть бути реалізовані та робочі характеристики сучасних перспективних інтегральних схем середньої й великої інтеграції. У наш час існує достатньо велика кількість інтегральних схем середньої й великої інтеграції, які реалізують різноманітні функції вузлів інформаційної системи: тригери, лічильники, шифратори, дешифратори й т. ін. Тому немає необхідності розробляти такі вузли за допомогою схем малої інтеграції, які реалізують елементарні логічні функції: **„І", „АБО", „І-НЕ", „НЕ"** і т. ін.

Таблиця 8.9

Розширена таблиця істинності для входів тригерів

Поточний стан				Вихід	Поточний стан				Наступний стан				Вхід тригера			
Q_3	Q_2	Q_1	Q_0	C	Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	T_3	T_2	T_1	T_0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	0	0	1	0	0	1	0	0	0	1	1
0	0	1	0	0	0	0	1	0	0	0	1	1	0	0	0	1
0	0	1	1	0	0	0	1	1	0	1	0	0	0	1	1	1
0	1	0	0	0	0	1	0	0	0	1	0	1	0	0	0	1
0	1	0	1	0	0	1	0	1	0	1	1	0	0	0	1	1
0	1	1	0	0	0	1	1	0	0	1	1	1	0	0	0	1
0	1	1	1	0	0	1	1	1	1	0	0	0	1	1	1	1
1	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	1
1	0	0	1	1	1	0	0	1	0	0	0	0	1	0	0	1
1	0	1	0	d	1	0	1	0	d	d	d	d	d	d	d	d
1	0	1	1	d	1	0	1	1	d	d	d	d	d	d	d	d
1	1	0	0	d	1	1	0	0	d	d	d	d	d	d	d	d
1	1	0	1	d	1	1	0	1	d	d	d	d	d	d	d	d
1	1	1	0	d	1	1	1	0	d	d	d	d	d	d	d	d
1	1	1	1	d	1	1	1	1	d	d	d	d	d	d	d	d

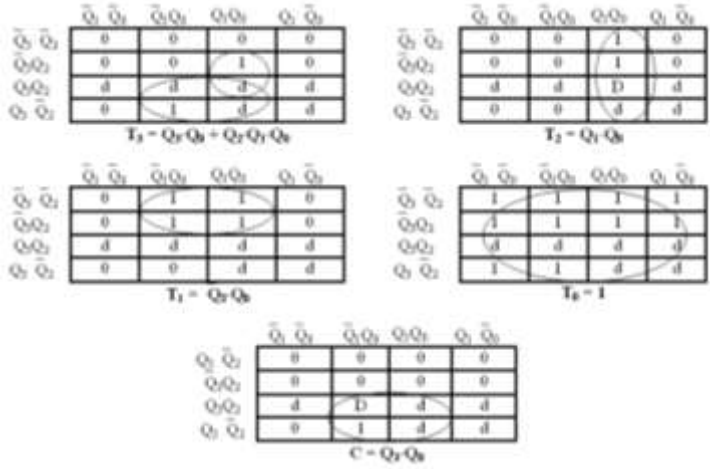


Рис. 8.9. Карти Карно

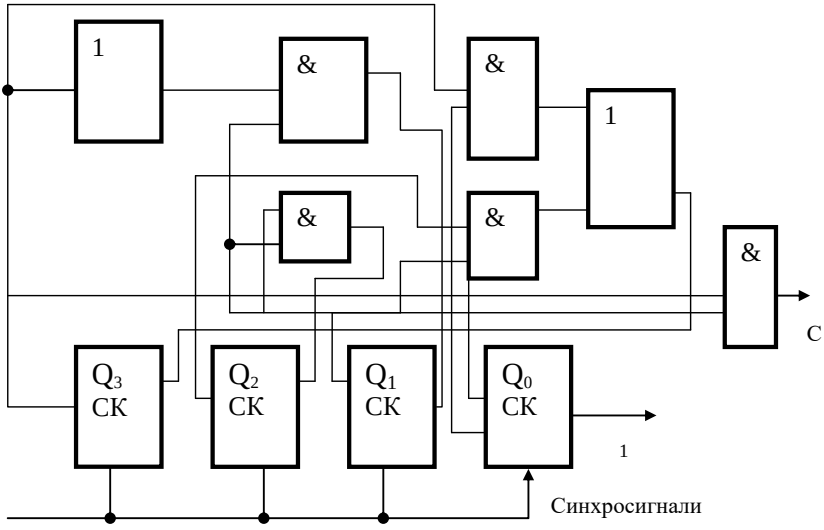


Рис. 8.10. Логічна схема десяткового лічильника

Ознакою послідовності логічної схеми є наявність петель. Під петлею розуміють шлях із виходу логічного елемента на його вхід безпосередньо або через інші елементи.

Крім розглянутого методу синтезу автомата у вигляді декомпозиції тригерів (елементарних автоматів), існує метод прямого синтезу послідовних логічних схем з використанням апарату синтезу часових функцій. Застосування цього методу дозволяє побудувати управляючий автомат у будь-якому функціонально повному елементному базисі.

Часові функції переключення, на відміну від звичайних функцій переключення, як аргумент можуть використовувати власні значення, що обумовлено наявністю петлі з виходу елемента на його вхід.

Розглянемо приклад. Нехай автомат (див. рис. 8.11) містить логічну схему (ЛС), яка складається з логічних елементів. Входами ЛС є зовнішні вхідні сигнали (логічні умови) $x_1...,x_k$, а також деякі виходи цієї ЛС, сигнали на яких розглядаються як часові функції $Q_1^t,...,Q_m^t$ (верхній індекс визначає момент автоматного часу). Логічна схема виробляє зовнішні управляючі сигнали $y_1...,y_p$ для операційного пристрою (на рис. 8.11 операційний пристрій не показано).

Стани автомата кодуються значеннями Q_i^t. У зв'язку з цим справедливе співвідношення $m \geq \lceil \log_2 S \rceil$. У загальному вигляді значення *i-й* часової функції можна записати як

$$Q_i^{t+1} = f(x_1,...,x_k, Q_1^t,...,Q_m^t).$$

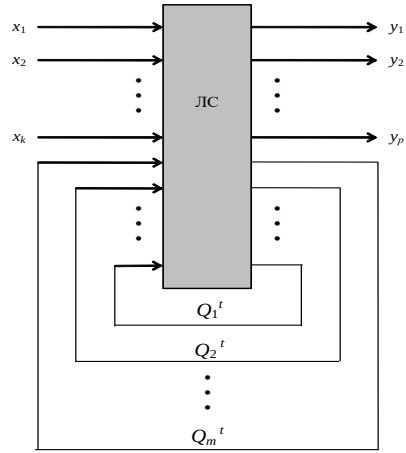


Рис. 8.11. Структура автомата

У процесі функціонування автомат переходить з одного стану в інший під дією зовнішніх управляючих сигналів. Кожному з множини станів $\{a_1...,a_i\}$ відповідає певний набір значень часових функцій Q_i^t.

Послідовність мікрооперацій, яка забезпечує задане перетворення інформації, називається *мікроалгоритмом.*

Для реалізації заданого мікроалгоритму перетворення інформації необхідний операційний пристрій, який виконуватиме необхідні мікрооперації. Управляючий автомат забезпечує формування управляючих сигналів для операційного пристрою.

Синтезу автомата передує побудова операційного пристрою й розробка мікроалгоритму заданого перетворення інформації. Вихідними даними для синтезу автомата є схема операційного пристрою й змістовний мікроалгоритм операції, яка містить описи мікрооперацій і логічних умов.

Синтез автомата містить такі етапи [6, 8, 14, 25, 27 та ін.]:

1) складання для операційного пристрою списку управляючих сигналів, які забезпечують виконання кожної мікрооперації;

2) визначення тривалості управляючих сигналів (тактів);

3) отримання закодованого мікроалгоритму;

4) оцінка станів автомата;

5) складання графа автомата;

6) кодування станів автомата значеннями Q_i^t;

7) визначення логічних виразів для часових функцій;

8) знаходження *МДНФ* управляючих сигналів;

9) представлення управляючих сигналів і часових функцій в операторній формі з урахуванням заданого елементного базису;

10) побудова й оптимізація схеми управляючого автомата.

Для аналізованого методу синтезу автоматів необхідною умовою є кодування його станів. Зв’язані дугами вершини графа, тобто стани автомату, повинні мати коди, які відрізняються значеннями лише одного розряду.

У загальному випадку для кодування станів автомата можна використовувати шаблони, показані на рис. 8.12. Граф накладають на шаблон відповідно до дуг, які з’єднують вершини. Якщо при накладанні графа на шаблон немає відповідних зв’язків, то вводять додаткові вершини. Це дозволяє перемістити розглянуту вершину на позицію, яка забезпечує необхідні зв’язки цієї вершини з іншими. Необхідність введення додаткових вершин може бути також обумовлена потребою забезпечення перепадів управляючих сигналів.

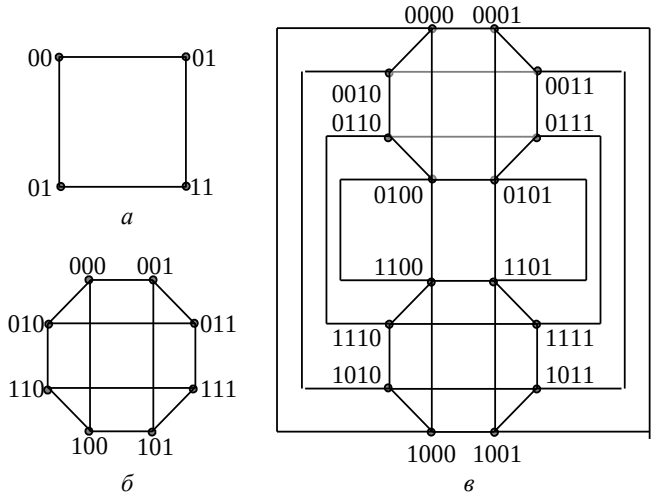


Рис. 8.12. Шаблони для кодування станів автомата
а – дворозрядні коди станів; б – трирозрядні коди станів;
в – чотирирозрядні коди станів

Для синхронного режиму роботи кожній дузі графа приписують синхросигнал. Один такт відповідає півперіоду синхросигналу. Для кожної вершини вхідні дуги повинні бути відзначені різними значеннями синхросигналів. Наприклад, вхідні дуги відзначені c, а вихідні – $\bar{c}$ (або навпаки).

Для відображення часових аргументів використовують залежність:

$$Q_i^{t+1} = F_i \vee Q_i^t \overline{G}_i \tag{8.1}$$

або

$$Q_i^{t+1} = (F_i \vee Q_i^t)\overline{G}_i \tag{8.2}$$

де F_i – функція установки часового аргументу в одиничний стан;
G_i – функція скидання часового аргументу в нульовий стан.

Будуючи схеми в булевому базисі, можна використовувати кожну з наведених формул. Для елементного базису Шеффера використовують формулу (8.1), а для базису Пирса – формулу (8.2). У цьому випадку відповідне операторне представлення часового аргументу можна отримати, застосовуючи правило де Моргана.

Для отримання функцій F_i записують через логічну операцію **„АБО”** всі умови переключення Q_i^t з „0” в „1”. Для функцій G_i записують умови переключення аргументу з „1” в „0”. Для визначення переходів 0→1 і 1→0 доцільно побудувати спочатку структурну таблицю автомата. Мінімізація функцій F_i і G_i виконується незалежно одна від одної. Якщо для кодування станів автомата використовують не всі передбачені шаблоном коди, то не використовувані коди можна розглядати як набори, на яких часові функції не визначено.

Як приклад розглянемо синтез автомата Мура, заданого графом на рис. 8.13.

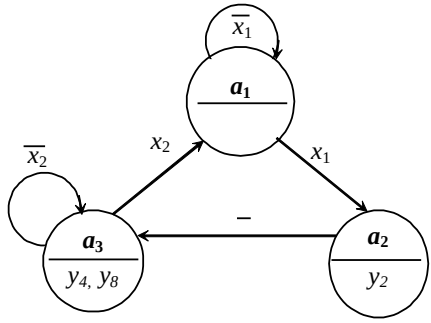


Рис. 8.13. Граф автомата

Використовуючи шаблон для дворозрядних кодів, отримаємо граф, показаний на рис. 8.14, який забезпечує перепади для управляючих сигналів під час зміни стану автомата.

Коди станів наведено в табл. 8.10. Дуги, які замикаються на власні вершини, на графі не відображено, оскільки значення часових аргументів визначено тільки дугами між різними вершинами графа.

Таблиця 8.10

Таблиця кодування станів

Стан	Код стану
	Q_1^t Q_2^t
a_1	0 0
a_2	0 1
a_3	1 0
β	1 1

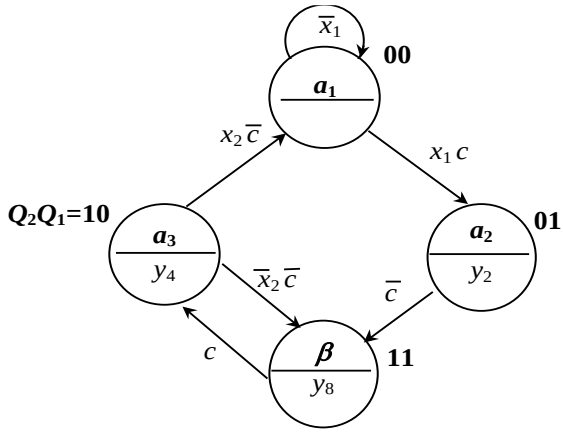


Рис. 8.14. Граф автомата

Для графа, показаного на рис. 8.14, отримаємо таку систему функцій:

$$\begin{cases} F_1 = a_1 x_1 c \vee a_3 \overline{x}_2 \overline{c} = \overline{Q}_2 \overline{Q}_1 x_1 c \vee Q_2 \overline{Q}_1 \overline{x}_2 \overline{c}; \\ G_1 = \beta c = Q_2 Q_1 c; \\ F_2 = a_2 \overline{c} = \overline{Q}_2 Q_1 \overline{c}; \\ G_2 = a_3 x_2 \overline{c} = Q_2 \overline{Q}_1 x_2 \overline{c}. \end{cases} \tag{8.3}$$

Приклад. Нехай треба реалізувати часові функції на елементах **„І-НЕ”**, а вихідні сигнали – на елементах **„І”**. Відповідно до формули (8.1) і системи рівнянь (8.3), використовуючи правило де Моргана, отримаємо операторну форму часових функцій:

$$Q_1^{t+1} = \overline{\overline{\overline{Q}_2 \overline{Q}_1 x_1 c} \cdot \overline{Q_2 \overline{Q}_1 \overline{x}_2 \overline{c}} \cdot \overline{R} \cdot \overline{Q_1^t \overline{R} \cdot \overline{Q_2 Q_1 c}}} \tag{8.4}$$

$$Q_2^{t+1} = \overline{\overline{Q}_2 Q_1 \overline{c} \overline{R} \cdot \overline{Q_2^t \overline{R} \cdot Q_2 \overline{Q}_1 x_2 \overline{c}}} \tag{8.5}$$

де R – сигнал установки часових функцій у нульовий стан.

Вихідні сигнали в автоматі Мура цілком визначаться кодом станів. Для графа, показаного на рис. 8.14, отримаємо: $y_2 = \overline{Q}_2 Q_1$, $y_4 = Q_2 \overline{Q}_1$, $y_8 = Q_2 Q_1$.

Розглянемо приклад проектування синхронного лічильника [14, 30, 38].

Існує багато різних схем лічильників у вигляді інтегральних схем – асинхронних, синхронних і комбінованих (асинхронно-синхронних). Більшість із них використовують для роботи звичайну двійкову послідовність, однак існують ситуації, коли лічильнику для роботи необхідно використати лічильну послідовність, яка не є звичайною двійковою послідовністю, наприклад: 000, 010, 101, 001, 011, 000 і т. д.

Існує кілька методів розробки лічильників, які формують довільну послідовність. Розглянемо деталі простого способу, який передбачає використання *JK*-тригерів у схемі синхронного лічильника. Той самий метод може бути використаний у розробці лічильника на D-тригерах. Методика – одна з кількох процедур розробки, що становлять процес створення цифрових схем, яка називається розробкою послідовних схем і розглядається в розширеному курсі схемотехніки.

У синхронних лічильниках усі тригери тактовані одночасно. Для гарантованої установки тригера в необхідний стан на входах *J* і *K* кожного тригера мають бути встановлені коректні рівні сигналу перед надходженням кожного тактового імпульсу. Наприклад, розглянемо ситуацію, показану в табл. 8.11. Під час надходження кожного наступного тактового імпульсу на входах *J* і *K* кожного тригера мають бути коректні рівні сигналу, оскільки це може стати причиною зміни стану тригера *C* з 1 в 0, тригера *B* з 0 в 1, тригера *A* з 1 в 1 (тобто без змін).

Процес розробки синхронного лічильника стає одним з етапів розробки логічних схем, які декодують змінювані стани лічильника для подачі логічних рівнів на кожний вхід *J* і *K*. На входи цієї декодуючої схеми подається сигнал з виходу одного або кількох тригерів. Так, для синхронного лічильника, зображеного на рисунку, елемент **„І”**, який живить входи *J* і *K* тригера *C*, є декодером станів тригерів *A* і *B*. Подібним чином елемент **„І”**, який живить входи *J* і *K* тригера *D*, декодує стани тригерів *A, B* і *C*.

Таблиця 8.11

Стани автомату

Попередній стан				**Наступний стан**		
C	*B*	*A*		*C*	*B*	*A*
1	0	1		0	1	1

Перед тем як розпочати процес розробки схеми декодування для кожного входу *J* і *K*, необхідно проаналізувати роботу *JK*-тригера, використовуючи особливий метод, що називається таблицею збуджень для *JK*-тригера. Крайній зліва стовпець цієї таблиці описує всі можливі зміни логічних рівнянь на виході тригера. Другий і третій стовпці, які описують поточні стани тригера, позначено відповідно $Q_{(N)}$ і наступного стану – $Q_{(N+1)}$, для кожної зміни стану. В останніх двох стовпцях перераховано логічні рівні на входах *J* і *K*, необхідні для здійснення кожного переключення. Опишемо кожний випадок.

Таблиця 8.12

Таблиця збуджень для *JK*-тригера

Переключення рівня на виході	Поточний стан $Q_{(N)}$	Наступний стан $Q_{(N+1)}$	*J*	*K*
0→0	0	0	0	x
0→1	0	1	1	x
1→0	1	0	x	1
1→1	1	1	x	0

0→0. 0 – поточний стан тригера, який залишається й після надходження тактового імпульсу (це може бути або у випадку, коли *J=K=0* (режим без змін) або *J=0*, а *K=1* (режим скидання)). Таким чином, вхід *J* повинен бути в стані 0, а вхід *K* може бути в будь-якому стані. У таблиці показано 0 під входом *J* і ***x*** нижче *K*. Нагадаємо, що ***x*** (або d) означає *„байдуже”* (будь-який стан).

0→1. 0 – поточний стан тригера, який переходить в 1 або коли *J=1*, а *K=0* (режим переустановлення), або *J=K=1* (режим переключення). Таким чином, для здійснення цього переходу вхід *J* має бути в стані 1, а вхід *K* може бути в будь-якому стані.

1→0. 1 – поточний стан тригера, який переходить у 0. Це відбувається за умови, що або *J=0*, а *K=1*, або *J=K=1*. Таким чином, вхід *K* має бути в стані 1, а вхід *J* може бути в будь-якому стані.

1→1. 1 – поточний стан тригера, який залишається 1. *J= K=0* або *J=1*, а *K= 0*. Таким чином, вхід *K* має бути в стані 0, а вхід *J* може бути в будь-якому стані.

Використання цієї таблиці збуджень для *JK*-тригера є принциповою частиною методики розробки синхронного лічильника.

На наступному прикладі, детально розглянемо процедуру розробки синхронного лічильника. Незважаючи на те, що ця методика розрахована на особливу лічильну послідовність, такі само етапи можна застосувати для будь-якої бажаної послідовності.

Етап 1. Визначаємо необхідну кількість розрядів і потрібну лічильну послідовність. В цьому прикладі, розробимо трибітовий лічильник, який використовуватиме лічильну послідовність, представлену в наступній таблиці. Ця послідовність не містить значення 101, 110 і 111, що належать до небажаних станів:

B	*C*	*A*
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
0	0	0
0	0	1

і т. д.

Етап 2. Накреслимо діаграму переходів станів лічильника, яка вказує всі можливі стани, включаючи ті, що не входять до складу потрібної лічильної послідовності.

Діаграма переходів станів може виглядати так, як показано на рис. 8.15. Стани від 000 до 100 з'єднано в послідовність, яка необхідна для роботи. Включення небажаних станів у цю діаграму є нововведенням. Небажані стани повинні включатися відповідно до цієї методики, оскільки лічильник може випадково встановлюватися в один із цих станів під час включення або під впливом шуму. Під час розробки необхідно вибирати, у який стан після випадкового потрапляння в небажаний стан переходитиме лічильник після надходження наступного тактового імпульсу. Найчастіше вибирають стан 000, тобто при установці в небажаний стан лічильник установлюється в стан 000 і вже з цього стану починається робота в нормальній лічильній послідовності.

Етап 3. Використати діаграму переходів для створення таблиці, яка перераховує всі можливі поточні стани та їх наступні стани. Для цього прикладу дані наведено в таблиці. У лівій частині таблиці перераховано всі можливі стани, навіть ті, які не є частиною необхідної лічильної послідовності. Ці стани позначено як поточні стани. У правій частині таблиці перераховано наступні стани для кожного поточного стану. Наступні стани отримано з діаграми переходів з рисунка. Наприклад, перший рядок показує, що в поточного стану 000 наступним станом буде стан 001, у п'ятому рядку поточний стан – 100, його наступний стан – 000. У шостому, сьомому й восьмому рядках указано всі небажані стани 101, 110 і 111, для яких наступним станом є стан 000.

Таблиця 8.13

Таблиця для JK-тригера

Номер рядка	Поточний стан			Наступний стан		
	C	*B*	*A*	*C*	*B*	*A*
1	0	0	0	0	0	1
2	0	0	1	0	1	0
3	0	1	0	0	1	1
4	0	1	1	1	0	0
5	1	0	0	0	0	0
6	1	0	1	0	0	0
7	1	1	0	0	0	0
8	1	1	1	0	0	0

Етап 4. Додати стовпець до цієї таблиці для кожного входу *J* і *K*. Для кожного поточного стану вказати логічний рівень, який необхідно прикласти для здійснення переходу в наступний стан.

У прикладі розробки лічильника використано три тригери *C, B, A*. Кожен з цих тригерів має по одному входу *J* і *K*. Таким чином, необхідно додати шість стовпців, як показано в таблиці. Ця повна таблиця називається таблицею *збуджень схеми.* Вміст у нових шести стовпцях указує на стани входів *J* і *K* кожного тригера. Вміст нових шести стовпців таблиці отримано з табл. 8.13.

Розглянемо перший рядок таблиці. Поточний стан 000 переходить до наступного стану 001 після надходження тактового імпульсу. У цей момент тригер *C* переходить із стану 0 в стан 0. З таблиці збуджень (8.14) *JK* тригера видно, що для здійснення переходу J_c має бути в стані 0, а K_c може бути в будь-якому стані. Тригер *B* також переходить із стану 0 в стан 0 і, таким чином, $J_B=0$, а $K_B = x$. Тригер *A* переходить із стану 0 в стан 1. З таблиці випливає, що для здійснення цього переходу $J_A = 1$, $K_A = x$.

У четвертому рядку таблиці поточний стан 011 переходить у наступний стан 100. Для цього тригер *C* переходить із стану 0 в стан 1. Із таблиці збуджень *JK*-тригера видно, що для здійснення переходу J_c повинен бути в стані 1, а K_c може бути в будь-якому стані. Обидва тригери *A* і *B* переходять із стану 1 в стан 0 і, таким чином, з таблиці отримуємо, що для здійснення цього переходу $J=x$, а $K=1$.

Таблиця 8.14

Таблиця збуджень для JK-тригера

Номер рядка	Поточний стан			Наступний стан								
	C	*B*	*A*	*C*	*B*	*A*	J_C	K_C	J_B	K_B	J_A	K_A
1	0	0	0	0	0	1	0	x	0	x	1	x
2	0	0	1	0	1	0	0	x	1	x	x	1
3	0	1	0	0	1	1	0	x	x	0	1	x
4	0	1	1	1	0	0	1	x	x	1	x	1
5	1	0	0	0	0	0	x	1	0	x	0	x
6	1	0	1	0	0	0	x	1	0	x	x	1
7	1	1	0	0	0	0	x	1	x	1	0	x
8	1	1	1	0	0	0	x	1	x	1	x	1

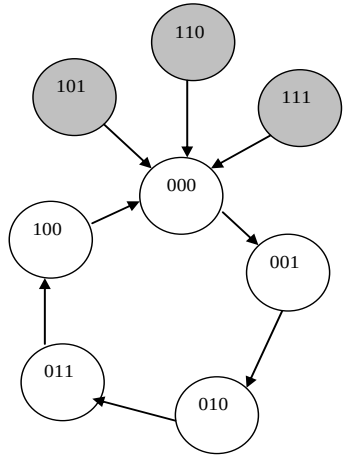


Рис. 8.15. Діаграма переходів

Логічні рівні, які необхідно прикладати на входи тригерів, можна отримати з таблиці аналогічно.

Етап 5. Розробити логічні схеми, які встановлюють логічні рівні на кожен вхід *J* і *K*.

У таблиці перераховано шість входів *J* і *K*: J_c, K_c, $J_в$, $K_в$, J_A і K_A. Необхідно розглянути кожен з них як вихід логічної схеми, на вхід якої підключено виходи тригерів *C, B* і *A*. Необхідно розробити схему для кожного входу. Розглянемо розробку схеми для J_A.

Спочатку необхідно дізнатися поточні стани *C, B* і *A*, а також потрібні логічні рівні, які треба подавати на J_A в кожному випадку. Цю

інформацію можна отримати з таблиці й представити на рисунку, 8.16 а. Ця таблиця істинності показує потрібні логічні рівні на вході J_A для кожного поточного стану. Звичайно, у деяких випадках немає значення, який логічний рівень подано на вхід J_A. Для проектування логічної схеми для J_A необхідно спочатку визначити його представлення через тригери *C, B* і *A*. Це можна зробити за допомогою переведення інформації з таблиці істинності в карту Карно, а потім спростити карту Карно, як показано на рис. 8.16 б.

У цій карті Карно є лише дві „одиниці", які можуть бути об'єднані для отримання виразу $\overline{A} \cdot \overline{C}$. Однак якщо врахувати байдужі стани $A \cdot \overline{B} \cdot \overline{C}$ і $A \cdot B \cdot \overline{C}$ як „одиниці", то можна обвести відповідну склейку й отримати більш простий вираз $\overline{C}$. Таким чином, остаточний вираз матиме вигляд

$$J_A = \overline{C}.$$

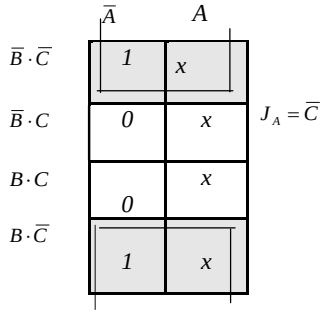
Тепер розглянемо вхід K_A. Можна проробити ті самі дії, які було здійснено для J_A. Втім, треба розглянути вміст стовпця для K_A і відзначити лише одиниці й байдужі стани. Якщо змінити всі байдужі стани на одиниці, то K_A завжди дорівнює 1 і остаточний вираз матиме вигляд

$$K_A = 1.$$

У такий же спосіб отримуємо вирази для J_C, K_C, J_B і K_B. Карти Карно для цих виразів представлено на рис. 8.17. У правильності виразів можна переконатися за допомогою таблиці збуджень схеми.

Поточний стан			**Вхід**
C	*B*	*A*	J_A
0	0	0	1
0	0	1	x
0	1	0	1
0	1	1	x
1	0	0	0
1	0	1	x
1	1	0	0
1	1	1	x

а



б

Рис. 8.16. Частина таблиці збуджень схеми й карта Карно
а -частина таблиці збудженої схеми, яка показує необхідний стан входу J_A для кожного поточного стану; б - спрощена карта Карно для отримання виразу для J_A

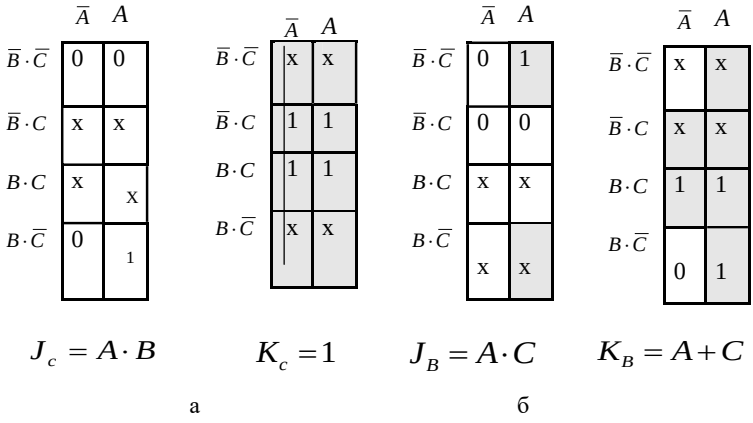


Рис. 8.17. Складання карти Карно
а - для логічних схем J_C і K_B; б - для логічних схем J_B і K_B

Етап 6. Реалізувати остаточні вирази. Логічні схеми для кожного входу *J* і *K* можна реалізувати з виразів, отриманих із складених карт Карно. Розроблений повний синхронний лічильник представлено на рис. 8.18. Слід відзначити, що всі тригери тактовані паралельно й логіка для входів *J* і *K* збігається з представленою на рис. 8.18.

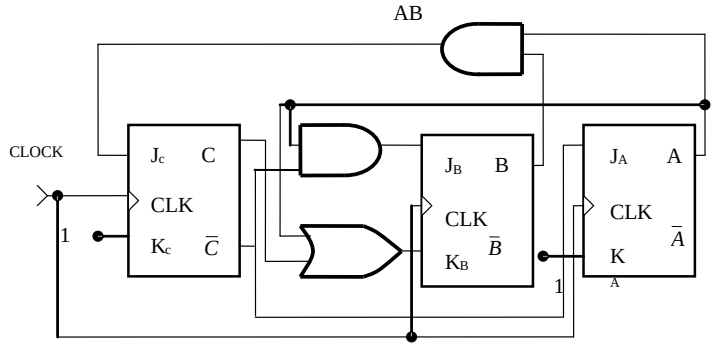
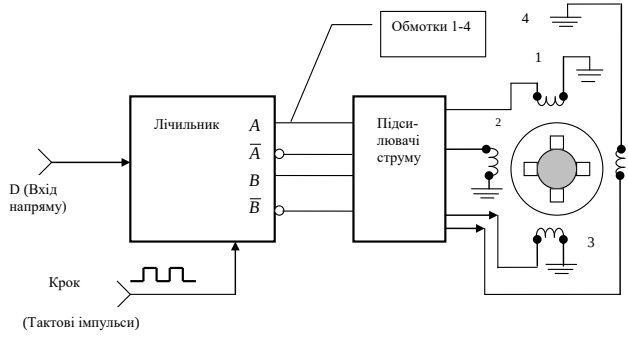


Рис. 8.18. Остаточний результат розробки синхронного лічильника

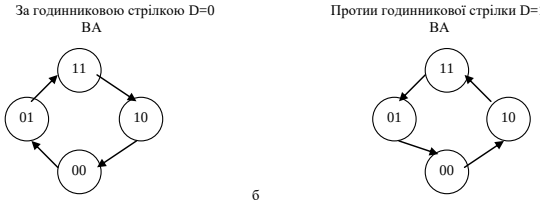
Тепер застосуємо викладену вище методику розробки для конкретного випадку – управляючого елемента крокового двигуна. Кроковий двигун постійно обертається по кроках (15° за крок). Котушка

електромагніта або обмотка всередині двигуна повинні підключатися й відключатися в особливій послідовності, яка й буде організовувати покрокову роботу. Зазвичай для управління струмом в обмотках двигуна застосовують цифрові сигнали. Крокові двигуни часто використовуються у випадках, коли потрібна точна установка частин приладу в певну точку, наприклад при установці головки зчитування (записи на магнітних дисках, управління друкуючою головкою принтера, в автоматах і т. ін.).

На рис. 8.19 а, представлено схему типового крокового двигуна з чотирма обмотками [30]. Для правильного обертання обмотки 1 і 2 повинні бути завжди в протилежних станах. Це означає, що коли на обмотку 1 подано живлення, то обмотку 2 відключено й навпаки. Обмотки 3 і 4 також повинні бути завжди в протилежних станах. Для управління струмом чотирьох обмоток використовують виходи дворозрядного синхронного лічильника. Виходами A і $\overline{A}$ управляють обмотки 1 і 2, а виходами B і $\overline{B}$ – обмотки 3 і 4. Оскільки з виходів тригерів не видаються значення струму, потрібні для безпосереднього управління обмотками, то між виходами тригерів і обмотками слід установити посилювачі струму.



а



б

Рис. 8.19. Крововий двигун
а - синхронний лічильник, який за допомогою потрібної лічильної послідовності керує кроковим двигуном; б - діаграми переходів станів для обох станів входу напряму D

Таким чином цей крокoвий двигун може обертатися й за годинниковою (CW), і проти годинникової стрілки (CCW), для чого є вхід напряму D, який управляє напрямами обертання. На рис. 8.19 б діаграма станів показує два випадки. Для обертання за годинниковою стрілкою потрібен стан D = 0, а стани лічильника BA мають відповідати лічильній послідовності: 11, 10, 00, 01, 11, 10 ... і т. д., як якщо б лічильник тактувався сигналом із входу *Step*. Для обертання проти годинникової стрілки необхідний стан входу D = 1 і лічильник повинен працювати в такій лічильній послідовності: 11, 01, 00, 10, 11, 01 і т. д.

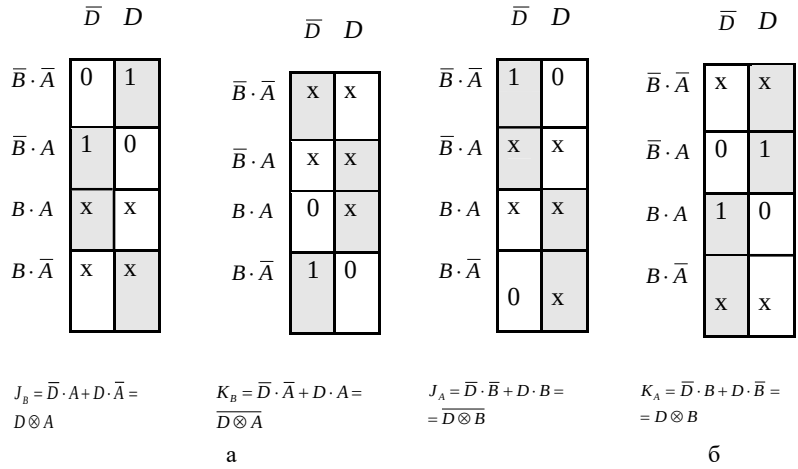


Рис. 8.20. Карти Карно
а - Карти Карно для J_B і K_B; б - карти Карно для J_A і K_A

А тепер пройдемо шість етапів методики розробки синхронного лічильника. Етапи 1 і 2 уже виконано, тому перейдемо до виконання етапів 3 і 4. У табл. 8.15 показано всі можливі поточні стани *D, B* і *A*, а також потрібні наступні стани разом з логічними рівнями, які необхідно подавати на кожен *J* і *K* вхід для здіснення переходів. Відзначимо, що в усіх випадках вхід напряму *D* не змінює свого стану під час переходу від поточного до наступного стану через те, що *B* – незалежний вхід, на якому утримується високий або низький рівень сигналу в процесі роботи лічильника з власною необхідною лічильною послідовністю.

Етап 5 методики розробки представлено на рис. 8.20, де вміст таблиці 8.15 було перенесено на карти Карно, які показують зв'язок кожного сигналу *J* і *K*.

Таблиця 8.15

Таблиця збуджень для JK-тригера

Поточний стан			**Наступний стан**						
D	*B*	*A*	*D*	*B*	*A*	J_B	K_B	J_A	K_A
0	0	0	0	0	1	0	x	1	x
0	0	1	0	1	1	1	x	x	0
0	1	0	0	0	0	x	1	0	x
0	1	1	0	1	0	x	0	X	1
1	0	0	1	1	0	1	x	0	x
1	0	1	1	0	0	0	x	x	1
1	1	0	1	1	1	x	0	1	x
1	1	1	1	0	1	x	1	x	0

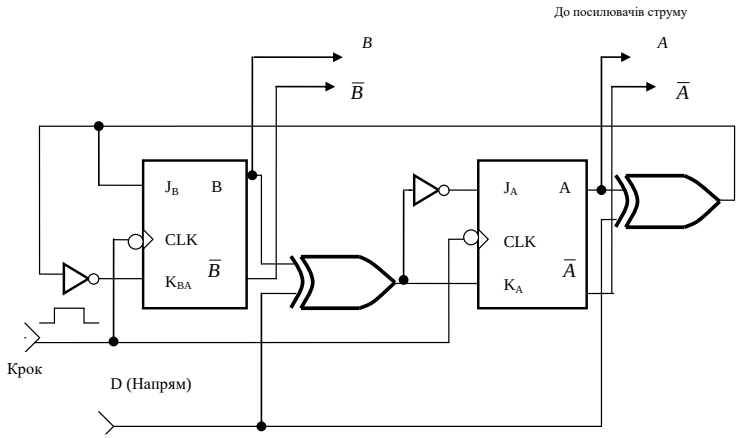


Рис. 8.21. Синхронний лічильник, реалізований з виразів для входів *J* і *K*

8.5. Мікроалгоритми, використовувані в цифрових автоматах

Перетворення інформації в арифметико-логічних пристроях інформаційних систем здійснюється шляхом послідовного виконання мікрооперацій над машинними словами (кодами чисел, символами та іншими об’єктами).

Під *мікроопераціями* розуміють елементарну дію, унаслідок виконання якої можуть змінюватися значення слів.

Машинне слово можна задати перерахуванням розрядів або за допомогою ідентифікаторів. Наприклад, 01011001 – 8-розрядне машинне слово, яке задано переліком усіх розрядів. Ідентифікатори можуть бути представлені рядом символів (букв і цифр), починаючи з букви. Доцільно використовувати ідентифікатори для позначення вузлів, на яких виконуються мікрооперації, наприклад: RG1, CT. Для визначення довжини слів і порядку розрядів використовують додаткові відомості в дужках, наприклад RG1(0.31), CT(7.0).

Алгебраїчні перетворення даних у цифрових пристроях виконують за допомогою таких мікрооперацій, як пересилка, додавання, зсув, підрахунок (декремент, інкремент), інвертування.

Для позначення мікрооперації будемо використовувати оператор присвоювання (: =).

Пересилка слів здійснюється зазвичай між двома регістрами. Результатом пересилки є запис слова в регістр, який є спадкоємцем слова. Джерелом інформації, крім регістрів, можуть бути зовнішні входи пристрою, на які надходять дані, наприклад, із пам'яті, системної шини та ін.

Як приклад на рис. 8.22 показано операційні схеми для виконання мікрооперацій пересилки даних P1:=DATA, P1:=P2, P1[31.24]:=P2[7.0].

Для сумування слів у схемі використовують суматор. Наприклад, операційні схеми на рис. 8.23 відповідають мікрооперціям P1:=P1+P2 і P1:=P2+P3.

Мікрооперації інкременту (CT:=CT+1) і декременту (CT:=CT-1) виконує лічильник.

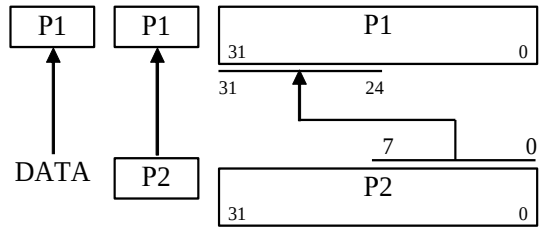


Рис. 8.22. Операційні схеми для виконання мікрооперацій пересилки даних

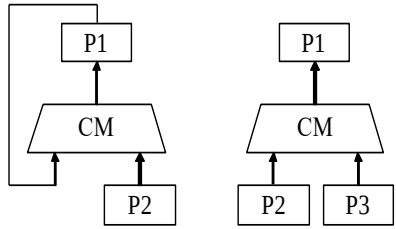


Рис. 8.23. Операційні схеми для виконання мікрооперацій підрахунку даних

Мікрооперації зсуву слів можуть бути виконані регістром зсуву вправо або вліво. Для визначення напряму зсуву використовують символи r (*right* – зсув вправо) і l (*left* - зсув вліво). За допомогою складеного слова можна визначати значення розряду, який заповнюється внаслідок зсуву. Наприклад, мікрооперація P1:=0.r[P1] і P1:=l[P1].P2(7) відповідають операційні схеми, показані на рис. 8.24.

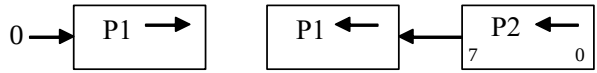


Рис. 8.24. Операційні схеми для виконання мікрооперації зсуву

Інвертування розрядів двійкових чисел можна забезпечити лінійкою логічних елементів (інверторів, елементів **„АБО, що виключає”**).

Для опису мікроалгоритмів можна використовувати графічні схеми мікроалгоритмів (ГСМ).

Мікроалгоритми можуть складатися і в змістовній, і в закодованій формі. У першому випадку мікрооперації записують у змістовній формі (наприклад, через оператори присвоювання). Для цього достатньо скласти лише операційну схему пристрою.

Для закодованого мікроалгоритму необхідна більш детальна схема (наприклад, функціональна або принципова), яка пояснює спосіб управління кожною мікрооперацією, включаючи визначення управляючих сигналів. Змістовні мікрооперації заміняються на сукупність управляючих сигналів, які забезпечують виконання мікрооперацій.

В арифметико-логічному пристрої (АЛП) з розподіленою логікою (інакше називають АЛП із закріпленими мікроопераціями) кожен вузол (регістр, лічильник) використовує власні логічні ланцюги для виконання мікрооперацій.

Побудова цих вузлів здійснюється так [8, 12, 14, 24, 27 та ін.]:

Крок 1. Для кожної операції будують операційну схему й змістовний мікроалгоритм. Рекомендовано вибирати мікроалгоритми (МА), які краще з'єднуються, тобто вимагають однакового напрямку зсуву в регістрах, однакового напрямку суматорів та ін.

Крок 2. Будують функціональну (принципову) схему з указівкою управляючих сигналів для кожного вузла.

Крок 3. Складають закодований мікроалгоритм.

Крок 4. Синтезують пристрій управління.

Розглянемо як приклад процес побудови пристрою з розподіленою логікою для виконання операції $D=2\cdot A^2+0{,}5\cdot B$.

Операційну схему для виконання відзначеної операції показано на рис. 8.25, де *RG1, RG2* – регістри, *SM* – суматор, *CT* – лічильник.

Необхідну тривалість управляючих сигналів визначають з урахуванням затримок в елементах операційного пристрою. Період часу t тактових сигналів зазвичай обирають рівним максимальній тривалості управляючих сигналів або мінімальним. При цьому величина t повинна бути не менша за час переключення автомата з одного стану в інший. У першому випадку всі мікрооперації виконуються в синхронному режимі (тобто за однаковий проміжок часу), а в другому – в асинхронному, причому тривалість управляючих сигналів кратна величині t.

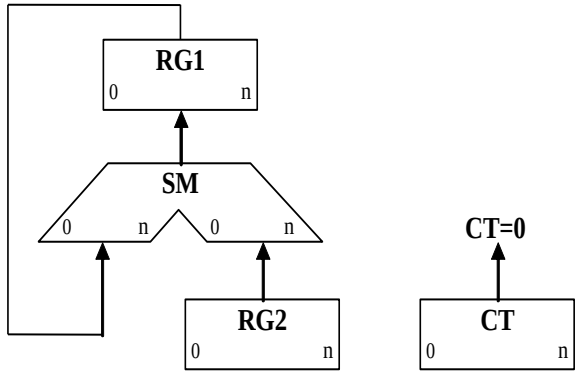


Рис. 8.25. Операційна схема пристрою

Для побудови змістовного мікроалгоритму в операторних вершинах розмістимо мікрооперації Y_i *(i=1,2.)*. При цьому в кожній операторній вершині розміщено мікрооперації, які можуть бути виконані одночасно. У логічних вершинах розмістимо вхідні сигнали, які є логічними умовами.

Змістовний мікроалгоритм виконання операції показано на рис. 8.26.

У вихідному стані операнд B записано в регістр RG2, а операнд A – у регістр RG1 і в лічильник CT. У першому такті шляхом зсуву вліво операнда A в регістрі RG1 здійснено його подвоєння. Шляхом зсуву вправо операнда B у регістрі RG2 здійснено ділення операнда B на 2. Далі до змісту регістра RG2 A раз додається слово, записане в RG1. Після кожного додавання вміст CT зменшується на 1. Обчислення завершуються після виконання умови CT=0. Відповідний цьому сигнал можна отримати, наприклад, дешифруванням нульового стану C. Результат операції формується в регістрі RG2.

На основі операційної схеми й змістовного мікроалгоритму отримаємо функціональну схему пристрою (див. рис. 8.27).

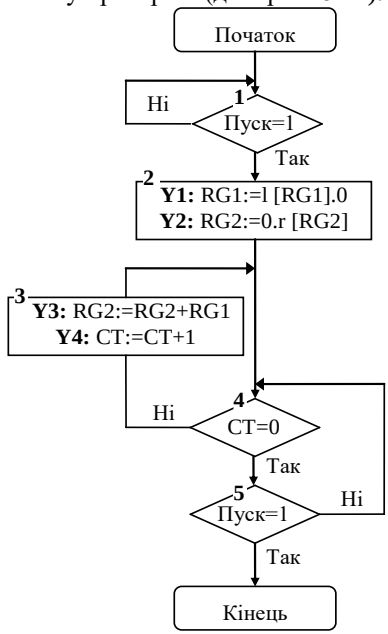


Рис. 8.26. Графічна схема мікроалгоритму операції

На функціональній схемі указано управляючі сигнали для всіх вузлів пристрою й способи формування умов:

W – запис інформації в регістри;

SR – зсув вправо вмісту регістрів;

SL – зсув вліво вмісту регістрів;

dec – зменшення значення лічильника на одиницю (декремент);

inc – збільшення значення лічильника на одиницю (інкремент);

d – сигнал, який дозволяє отримати додатковий код числа для реалізації операції віднімання.

Для виконання мікрооперації на регістри необхідно подати одиничний сигнал на відповідний управляючий вхід. На всі інші управляючі входи цього регістру повинен бути поданий нульовий сигнал.

Визначимо сукупність управляючих сигналів, необхідних для виконання кожної мікрооперації. Мікрооперації *Y1, Y2, Y3* і *Y4* кодуються відповідно управляючими сигналами *SL1, SR2, W1* та *inc*.

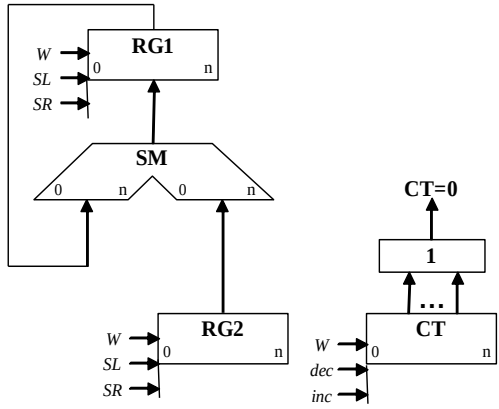


Рис. 8.27. Функціональна схема пристрою

Асинхронний режим можна забезпечити, наприклад, уведенням у мікроалгоритм додаткових операторних вершин з управляючими сигналами, тривалість яких перевищує *t*.

Будемо вважати, що з урахуванням швидкодії елементів для розглянутого прикладу управляючі сигнали *Y3* і *Y8* повинні мати тривалість *2t*, а інші – *t*. Результати виконання перших двох етапів показано в табл. 8.16.

Таблиця 8.16

Таблиця тривалості управляючих сигналів

Мікрооперації	Управляючі сигнали	Тривалість управляючих сигналів
Y_1 :P_1 := L1(P_1)	*SR1,SR2*	*t*
Y_2 :P_2 := R1(P_2)	*W1*	*t*
Y_3 :P_2 := P_2+P_1	y_3, y_8, y_5	2*t*, 2*t*, *t*
Y_4 :C := C-1	y_9	*t*

Для отримання закодованого мікроалгоритму представимо в змістовному мікроалгоритмі описи логічних умов їх позначками (див. табл. 8.17), описи мікрооперацій – відповідними управляючими сигналами.

Таблиця 8.17

Таблиця логічних умов

Логічні умови	Позначення логічних умов
Пуск=1	x_1
С=1	x_2

Отриманий закодований алгоритм буде вихідним у побудові цифрового автомата для управління операційним пристроєм.

Управляючі сигнали *SL1* і *SR2* генеруються управляючим автоматом у той самий такт автоматного часу, отже, можуть бути замінені одним управляючим сигналом *Y1*, сигнали *W2, inc* замінимо сигналом *Y2*. Будемо вважати, що прилад управління необхідно побудувати у вигляді автомата Мілі. У цьому випадку закодований мікроалгоритм матиме вигляд, наведений на рис. 8.28.

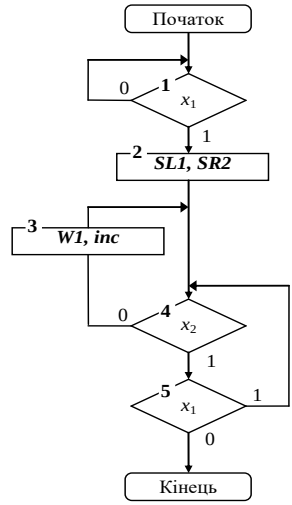


Рис. 8.28. Графічна схема закодованого мікроалгоритму автомата Мілі

Розглянемо приклад проектування пристроїв для множення чисел. Під час множення чисел у прямих кодах знакові й основні розряди обробляють окремо. Для визначення знака добутку здійснюють сумування

за *mod2* цифр, записаних у знакових розрядах множників. Будемо вважати, що множене Y і множник X – правильні двійкові дробі вигляду X=0,x_1, x_2,...,x_n Y=0, y_1, y_2,...,y_n, де x_i, $y_i \in \{0,1\}$. Тоді добуток Z модулів чисел дорівнює

$$Z = Y \cdot X = Yx_1 2^{-1} + Yx_2 2^{-2} + ... + Yx_i 2^{-i} + ... + Yx_n 2^{-n}. \qquad (8.6)$$

Множення Y і X може бути реалізовано шляхом виконання певного циклічного процесу, характер якого залежить від конкретної форми залежності (8.6). Один цикл множення складається з додавання чергового часткового добутку до суми часткових добутків. Розрізняють чотири способи множення [24, 35].

Перший спосіб множення

Вираз (8.6) можна представити у вигляді

$$Z = Y \cdot X = ((...((0 + Yx_n)2^{-1} + Yx_{n-1})2^{-1} + ... + Yx_i)2^{-1} + ... + Yx_1)2^{-1}.$$

Звідси випливає, що отримання суми часткових добутків в *i-му* циклі ($i = \overline{1,n}$) зведено до обчислення $Z_i = (Z_{i-1} + Yx_{n-i+1})2^{-1}$ з початковими значеннями $i=1$, $Z_0=0$, причому $Z_n=Z=YX$.

Множення починається з молодших розрядів множника, сума часткових добутків зсувається вправо, а множене залишається нерухомим.

Другий спосіб множення

Запишемо вираз (8.1) у вигляді

$$Z = ((...((0 + Y2^{-n}x_n) + Y2^{-n+1}x_{n-1}) + ... + Y2^{-1}x_1.$$

Очевидно, що процес множення може бути зведено до *n-кратного* виконання циклу $Z_i = Z_{i-1} + Y_i x_{n-i+1}, \quad Y_i = 2Y_{i-1}$ з початковими значеннями i=1, $Y_0=Y2^{-n}$, Z_0=0. Множення здійснюється з молодших розрядів, множене зсувається вліво, а сума часткових добутків залишається нерухомою.

Третій спосіб множення

Запишемо вираз (8.1) у вигляді

$$Z = ((...((0 + Y2^{-n}x_1)2 + Y2^{-n}x_2)2 + ... + Y2^{-n}x_i)2 + ... + Y2^{-n}x_n.$$

Отже, суму часткових добутків у *i-му* циклі ($i = \overline{1,n}$) можна отримати за формулою $Z_i = 2Z_{i-1} + Y2^{-n}x_i$.

Початковими значеннями є $i=1$, $Z_0=0$. Множення починається зі старших розрядів множника, сума часткових добутків зсувається вліво, а множене нерухоме.

Четвертий спосіб множення

Запишемо вираз (8.1) у вигляді

$$Z = ((...((0 + Y2^{-1}x_1) + Y2^{-2}x_2) + ... + Y2^{-i}x_i) + ... + Y2^{-n}x_n .$$

Процес множення може бути зведено до *n-кратного* виконання циклу $Z_i = Z_{i-1} + Y_{i-1}x_i, \quad Y_i = Y_{i-1}2^{-1}$ з початковими значеннями $i=1$, $Y_0=Y2^{-1}$, $Z_0=0$. Множення починається зі старших розрядів множника, сума часткових добутків залишається нерухомою, а множене зсувається вправо.

Для формування й накопичення суми часткових добутків можна використовувати або комбінаційний суматор (SM) і регістр добутку, або лише накопичувальний суматор, який у функціональному відношенні можна розглядати як композицію комбінаційного суматора й регістра. Принцип побудови пристроїв, які реалізують різні способи множення, показано на рис. 8.29, де RG3 – регістр множеного, RG1 – регістр добутку, RG2 – регістр множника. Цифрами відмічено номери розрядів SM і регістрів, а стрілками показано напрям зсуву кодів у регістрах. Цифри, записані в молодших розрядах регістрів RG3 і RG1, при реалізації першого способу мають вагу 2^{-n}, а при реалізації інших способів – 2^{-2n}. Перед початком множення будь-яким способом регістр RG1 установлюють у нульовий стан. Підрахунок кількості циклів множення забезпечують лічильники СТ, відповідно з чим обирають його розрядність q.

При множенні першим способом (див. рис. 8.29 а) у першому такті *i-го* циклу аналізується значення RG2(n) – молодшого (*n-го)* розряду регістра RG2, у якому знаходиться чергова цифра множника. Вміст регістра RG3 додається до суми часткових добутків, які знаходяться в регістрі RG1, якщо RG2(n)=1, або не додається, якщо RG2(n)=0. У другому такті здійснюється зсув вправо в регістрах RG1 і RG2, що еквівалентно множенню їх вмісту на 2^{-1}. При зсуві цифра молодшого розряду регістра RG1 записується в старший розряд регістра RG2, який звільнюється. Після виконання n циклів молодші розряди 2n-розрядного добутку буде записано в регістр RG2, а старші – у RG1.

Час множення, якщо не застосовують методи прискорення операції, визначається виразом $t_y=n(t_т+t_з)$, де t_T і $t_з$ – відповідно до тривалості тактів сумування і зсуву.

Перед початком множення другим способом (див. рис. 8.29 б) X записують у регістр RG2, а Y – у молодші розряди регістра RG3 (тобто в

регістрі RG3 встановлюють $Y0=Y2^{-n}$). У кожному *i-му* циклі множення додаванням кодів RG3 і RG1 управляє цифра *RG2(n)*, а в регістрі RG3 здійснюється зсув уліво на один розряд, унаслідок чого формується величина $Yi=2Y_{i-1}$. Оскільки сума часткових добутків у процесі множення нерухома, зсув у регістрі RG3 можна з'єднати в часі із сумуванням (зазвичай, $t_T \geq t_з$). У цьому випадку $t_y=nt_T$. Множення закінчується за нульовим вмістом регістра RG2, що також зумволює збільшення швидкодії, якщо множник ненормалізований.

При множенні третім способом (див. рис. 8.29 в) вага молодшого розряду RG3 дорівнює 2^{-2n}, тому код у регістрі RG3 становить собою значення Y^{2-n}. На початку кожного циклу множення здійснюється зсув уліво в регістрах RG1 і RG2, а потім виконується додавання, яким управляє RG2(1). Унаслідок сумування вмісту регістрів RG3 і RG1 може виникнути перенос у молодший розряд регістра RG2. Збільшення довжини RG2 на один розряд усуває можливість поширення переносу в розряди множника.

Перед множенням четвертим способом (див. рис. 8.29 г) множник записують у регістр RG2, а множене – у старші розряди регістра RG3 (тобто в RG3 установлюють $Y_0=Y^{2-1}$). У кожному циклі цифра RG2(1), яка знаходиться в старшому розряді регістра RG2, керує сумуванням, а в RG3 здійснюється зсув вправо на один розряд, що еквівалентно множенню вмісту цього регістра на 2^{-1}. Час виконання множення четвертим способом становить $t_y=n \cdot t_T$.

Для підрахунку циклів у пристроях використовують лічильники СТ.

В інформаційних системах під час роботи з дробовими числами часто треба обчислювати не 2^n, а лише $n+1$ цифр добутку й округляти його до n цифр. У цьому випадку при реалізації другого способу можна зменшити довжину SM і RG1, а при реалізації четвертого – довжину SM, RG1 і RG3. Для того щоб похибка від відкидання молодших розрядів не перевищила половини ваги *n-го* розряду результату, у перерахованих вузлах достатньо мати лише по l додаткових молодших розрядів, де l вибирається з умови $l \geq 1 + \log_2(n-l-1).$

Операцію округлення здійснюють зазвичай шляхом додавання одиниці до *n+1-го* розряду результату й відкидання всіх розрядів, розташованих правіше *n-го*. При цьому похибка стає знакоперемінною, а максимальне абсолютне її значення не перевищує половини ваги молодшого розряду.

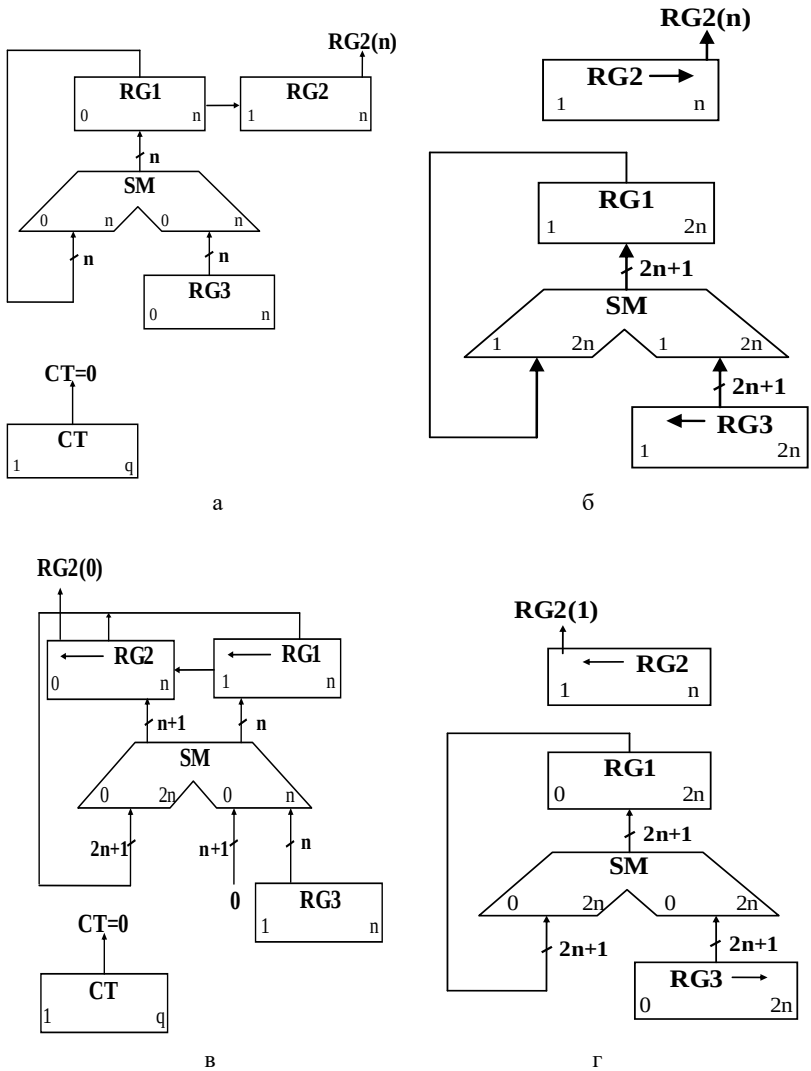


Рис. 8.29. Операційні схеми множення
а – перший спосіб; б – другий спосіб; в – третій спосіб;
г – четвертий спосіб

У процесі формування суми часткових добутків код з регістра RG1 видається на суматор SM, а з виходів SM знову записується в регістр RG1. У зв'язку з цим при використанні потенційних елементів регістр RG1 будують на тригерах з внутрішньою затримкою. Характер управляючих сигналів у ланцюгу визначено конкретною теоретичною реалізацією вузлів і використовуваною елементною базою.

У пристроях, які реалізують другий і четвертий способи множення, можна без пересилок кодів між регістрами обчислювати вирази виду $\sum X_i Y_i$, де ($i = \overline{1,n}$), для чого достатньо черговий результат операції залишати в регістрі RG2, який у цьому випадку повинен мати додаткові старші розряди.

У пристрої, який реалізує третій спосіб, можна без пересилок обчислювати, наприклад, функції виду X_i. Для цього X перед початком обчислення записують у регістр RG3 і в молодші розряди регістра RG2, а потім *i-1* раз виконують операцію множення з округленням проміжних результатів до n розрядів. Після кожної чергової операції регістр RG1 установлюють у нульовий стан. Остаточний результат буде знаходитися в n молодших розрядах регістра RG2. Найпростішими є пристрої, що реалізують перший спосіб, а найбільш швидкодіючими – другий і четвертий. Однак другий спосіб не має особливих переваг порівняно з четвертим і, крім того, вимагає великих апаратурних витрат під час реалізації.

Структурну схему пристрою для реалізації операції множення першим способом наведено на рис. 8.29 а.

Цифрову діаграму наведено на рис. 8.30.

№ такту	RG1		RG2		RG3		СТ
ПС	0	0000	0	1011	0	1111	100
1	0 0 0 0	0000 1111 1111 0111	1	1101	0	1111	011
2	0 0 1 0	0111 1111 0110 1011	0	0110	0	1111	010
3	0	0101	1	1011	0	1111	001
4	0 0 1 0	0101 1111 0100 1010	0	0101	0	1111	000

Рис. 8.30. Цифрова діаграма виконання операції множення

Змістовний алгоритм виконання операції множення наведено на рис. 8.31. Функціональну схему пристрою для виконання операції множення першим способом показано на рис. 8.32.

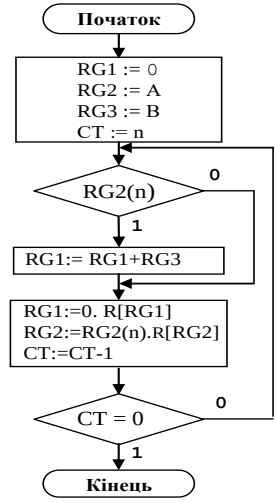


Рис. 8.31. Мікроалгоритм виконання операції множення першим способом

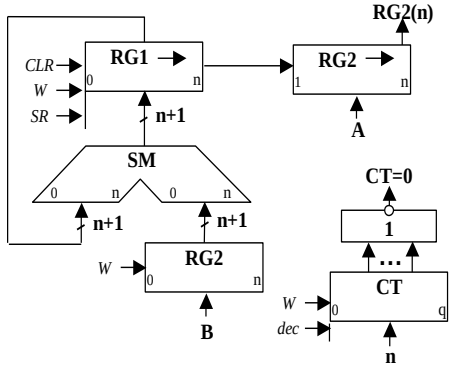


Рис. 8.32. Функціональна схема пристрою для виконання операції множення першим способом

8.6. Синтез мікропрограмних автоматів за граф-схемою алгоритму

Будуючи пристрій, що складається з операційного (ОА) й управляючого (УА) автоматів, необхідно виділити функції операційного й управляючого автоматів з ГСА. Зазвичай мікропрограму представлено у вигляді змістовної ГСА. У цьому випадку для задавання функцій операційного автомата необхідно перерахувати всі виконувані мікрооперації й усі логічні умови цієї мікропрограми, що підлягають перевірці, а також описати розрядність слів, які обробляють операційним пристроєм. Для ініціалізації виконання тієї чи іншої мікрооперації на операційному автоматі повинні надходити в потрібний момент часу управляючі сигнали Y_i, згідно з ГСА.

При проектуванні операційного автомата приймають певний спосіб кодування мікрооперацій (найчастіше код, який містить стільки розрядів, скільки всього різних мікрооперацій) і для розробки операційного автомата вважають, що управляючий автомат видає код мікрооперацій, які повинні виконуватися в цей момент часу [24].

Для керуючого автомата важливою є послідовність видачі відповідних кодів мікрооперацій залежно від логічних умов, які виробляє операційний автомат й аналізує операційного й керуючого автоматів у потрібні моменти часу. Якщо прийнято спосіб кодування мікрооперацій, то функції УА задає кодована ГСА. Тому для різних змістовних ГСА, які мають однакову кодовану ГСА, операційні автомати будуть різними, але керуючий автомат буде тим самим.

Кінцевий автомат, який інтерпретує мікропрограму роботи дискретного пристрою, називається *мікропрограмним автоматом***.** Ту саму ГСА можна інтерпретувати й автоматом Мілі, і автоматом Мура.

Абстрактний синтез мікропрограмного автомата за ГСА здійснюють у два етапи: 1) отримання відміченої ГСА; 2) побудова графу автомата або таблиць переходів і виходів.

Синтез автомата Мілі

На етапі отримання відзначеної ГСА входи вершин, які слідують за операторними, відмічають символами a_1, a_2, за такими правилами:

1) символом a_1 відмічають вхід вершини, яка слідує за початковою, а також вхід кінцевої вершини;

2) входи всіх вершин, які слідують за операторними, повинні бути відмічені;

3) входи різних вершин, за винятком кінцевої, відмічаються різними символами;

4) якщо вхід вершини відмічають, то лише одним символом.

Очевидно, що для проведення відміток потрібна скінченна кількість символів $a_1,...,a_m$. Результатом першого етапу є відмічена ГСА, яка слугує основою для другого етапу – переходу до графу або таблиць переходів-виходів. Приклад ГСА, відміченої для автомата Мілі, представлено на рис. 8.33.

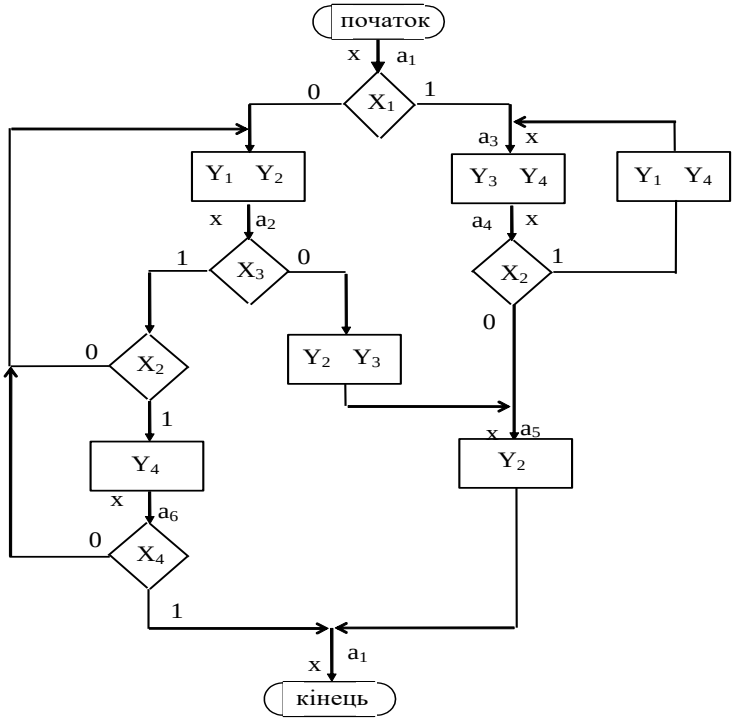


Рис. 8.33. ГСА, відмічена для автомата Мілі

На другому етапі з відміченої ГСА будують граф автомата або таблиці переходів-виходів. Для цього вважають, що в автоматі буде стільки станів, скільки символів a_i було потрібно для відмітки ГСА.

На площині рисунка відмічаємо всі стани автомата a_i. Для кожного із станів a_i визначаємо за відміченою ГСА всі шляхи, які ведуть в інші стани й проходять **обов'язково лише через одну операторну вершину**. Наприклад, із стану a_1 (рис. 8.33) є перехід у стан a_2 (шлях проходить через операторну вершину y_1 y_2) і в стан a_4 (шлях проходить через вершину y_3 y_4). Переходу з a_1 в a_3 немає, оскільки на цьому шляху немає жодної

операторної вершини. Будемо вважати, що автомат здійснює перехід, наприклад, з a_1 в a_2 за умови $x_1=0$ або $\bar{x}_1$ (див. ГСА, рис. 8.33) і виробляє на цьому переході вихідні сигнали y_1 y_2 (те, що записано в прохідній операторній вершині ГСА). Значення умов x_2, x_3, x_4 на цьому переході не впливає на автомат.

Виняток становить лише шлях, який веде в кінцеву вершину, він може не містити жодної операторної вершини (наприклад, перехід з a_6 в a_1), тобто не супроводжується виробленням вихідних сигналів.

Відмічаємо на графі всі вказані шляхи для всіх станів у вигляді дуг, яким приписуємо умови переходу й вихідний сигнал, що виробляється на цьому переході. Отримаємо граф автомата (рис. 8.34).

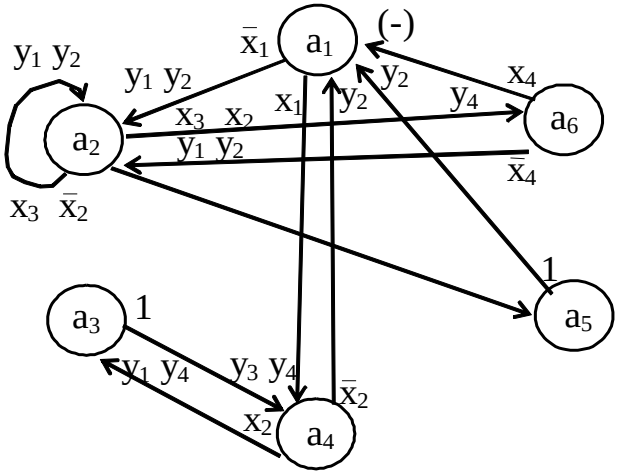


Рис. 8.34. Граф автомата Мілі

На цьому графі переходам типу $a_3 \rightarrow a_4$, $a_5 \rightarrow a_1$ приписано умову переходу 1, оскільки ці переходи є безумовними й виконуються завжди, коли автомат потрапляє в стан a_3 (або a_5). На підставі відзначеної ГСА або графу автомата можна побудувати таблицю переходів-виходів. Для мікропрограмних автоматів таблицю переходів-виходів будують у вигляді списку, розрізняють пряму й зворотну таблиці. Для цього автомата пряму таблицю представлено в табл. 8.18, зворотну – у табл. 8.19.

У наведених таблиця x a_m – початковий стан, a_S – стан переходу, X – умова (вхідний сигнал), яка забезпечує перехід із стану a_m в стан a_s, Y – вихідний сигнал, що виробляється автоматом при переході з a_m в a_S.

Таблиця 8.18

Пряма таблиця переходів – виходів автомата Мілі

a_m	a_s	x	y
a_1	a_2 a_4	$\bar{x}_1$ x_1	$y_1 \cdot y_2$ $y_3 \cdot y_4$
a_2	a_2 a_5 a_6	$x_3 \cdot \bar{x}_2$ $\bar{x}_3$ $x_3 \cdot x_2$	$y_1 \cdot y_2$ $y_2 \cdot y_3$ y_4
a_3	a_4	1	$y_3 \cdot y_4$
a_4	a_1 a_3	$\bar{x}_2$ x_2	y_2 $y_1 \cdot y_4$
a_5	a_1	1	y_2
a_6	a_1 a_2	x_4 $\bar{x}_4$	- $y_1 \cdot y_2$

Таблиця 8.19

Зворотна таблиця переходів – виходів автомата Мілі

a_m	a_s	x	y
a_4 a_5 a_6	a_1	$\bar{x}_2$ 1 x_4	y_2 y_2 -
a_1 a_2 a_6	a_2	$\bar{x}_1$ $x_3 \cdot \bar{x}_2$ $\bar{x}_4$	$y_1 \cdot y_2$ $y_1 \cdot y_2$ $y_1 \cdot y_2$
a_4	a_3	x_2	$y_1 \cdot y_4$
a_1 a_3	a_4	x_1 1	$y_3 \cdot y_4$ $y_3 \cdot y_4$
a_2	a_5	$\bar{x}_3$	$y_2 \cdot y_3$
a_2	a_6	$x_3 \cdot x_2$	y_4

Синтез автомата Мура

Для автомата Мура на етапі отримання відміченої ГСА розмітку здійснюють згідно з такими правилами:

1) символом a_1 відзначають початкову й кінцеву вершини;

2) різні операторні вершини відмічають різними символами;

3) усі операторні вершини повинні бути відмічені.

Приклад ГСА, відміченої для автомата Мура, представлено на рис. 8.35.

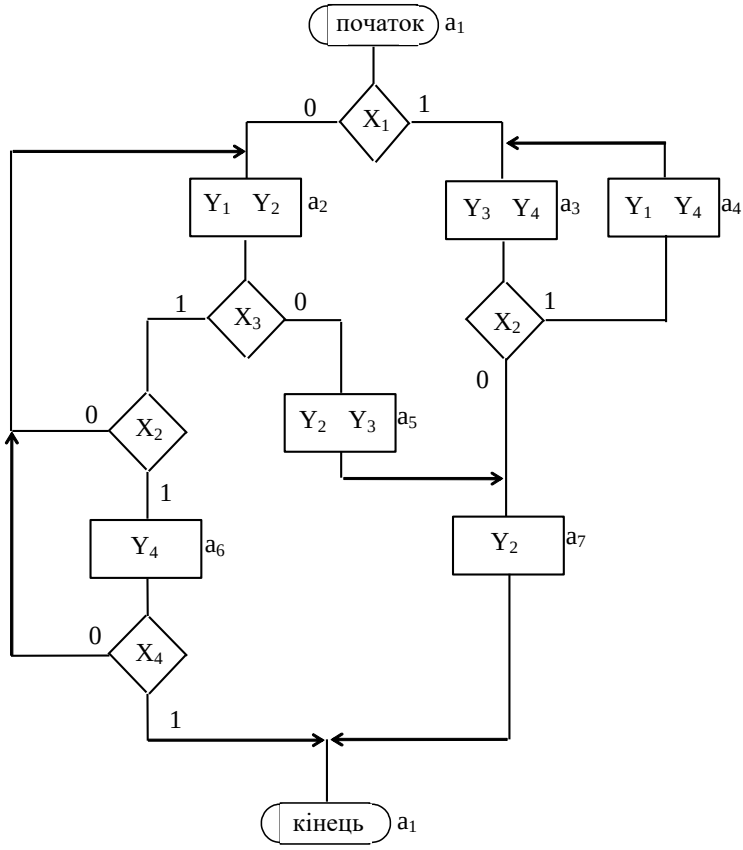


Рис. 8. 35. ГСА, відмічена для автомата Мура

Граф автомата Мура, що відповідає відміченій ГСА (рис. 8.35), представлено на рис. 8.36. Побудова його аналогічна побудові графа для автомата Мілі [6, 8, 9, 14, 15, 25, 27 та ін.].

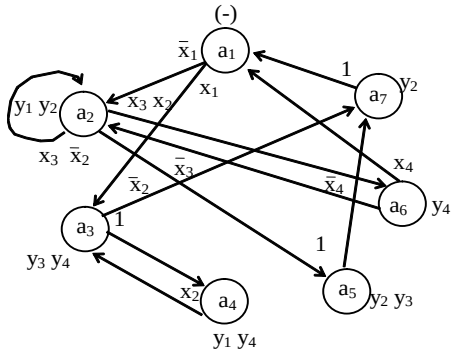


Рис. 8.36. ГСА, відмічена для автомата Мура

Таблиці переходів-виходів автомата Мура представлено в табл. 8.20 і 8.21. Зазвичай для автомата Мура в таблиці переходів-виходів додатковий стовпець для вихідних сигналів не використовують, і вихідний сигнал записують у стовпці, де вказано початковий стан a_m або стани переходу a_S.

Таблиця 8.20

Пряма таблиця переходів автомата Мура

$a_m(y)$	a_s	x
$a_1(--)$	a_2 a_3	$\bar{x}_1$ $\bar{x}_1$
$a_2(y_1y_2)$	a_2 a_5 a_6	$x_3 \cdot \bar{x}_2$ $\bar{x}_3$ $\bar{x}_3 \cdot x_2$
$a_3(y_3y_4)$	a_4 a_7	x_2 $\bar{x}_2$
$a_4(y_1y_4)$	a_3	1
$a_5(y_2y_3)$	a_7	1
$a_6(y_4)$	a_1 a_2	x_4 $\bar{x}_4$
$a_7(y_2)$	a_1	1

Таблиця 8.21

Зворотна таблиця переходів автомата Мура

a_m	$a_s(y)$	x
a_6 a_7	$a_1(-)$	x_4 1
a_1 a_2 a_6	$a_2(y_1y_2)$	$\bar{x}_1$ $x_3 \cdot \bar{x}_2$ $\bar{x}_4$
a_1 a_4	$a_3(y_3y_4)$	x_1 1
a_3	$a_4(y_1y_4)$	x_2
a_2	$a_5(y_2y_3)$	$\bar{x}_3$
a_2	$a_6(y_4)$	$x_3 \cdot x_2$
a_3 a_5	$a_7(y_2)$	$\bar{x}_2$ 1

Отриманням графа або таблиць переходів-виходів завершується етап абстрактного синтезу мікропрограмного автомата. Як і для скінченних автоматів, на етапі абстрактного синтезу можна виконати мінімізацію кількості внутрішніх станів автомата.

Розглянемо наступний приклад синтезу мікропрограмного автомата для виконання операції множення двійкових чисел.

На рис. 8.37 показана ГСА множення двох чисел. Відповідно, на рис. 8.38, показана ГСА закодованого мікроалгоритму.

На етапі отримання відзначеної ГСА входи вершин, які слідують за операторними, відмітимо символами a_1, a_2, a_3, a_4 . Отримаємо граф МПА (див. рис. 8.39.).

Далі отримаємо структурну таблицю графа МПА(див. табл. 8.22.).

Автомат має тільки чотири стани, для їхнього кодування достатньо застосувати два елементи пам'яті (Q_1 *і* Q_2), використовуючи асинхронні елементи типа *RS*, а кодування провести за табл. 8.23.

На підставі таблиці переходів асинхронного RS-тригера заповнюються колонки для **R** і **S**. Оператори $Y_1 = y_1, y_6$; $Y_2 = y_2$; $Y_3 = y_3, y_4, y_5, y_7$.

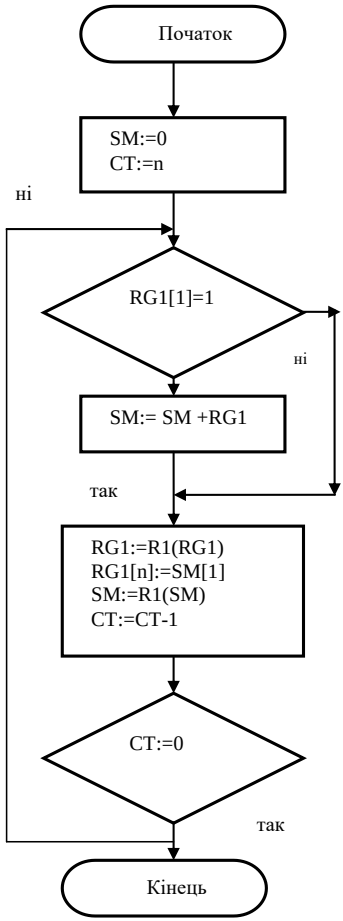


Рис. 8.37. ГСА множення двох чисел

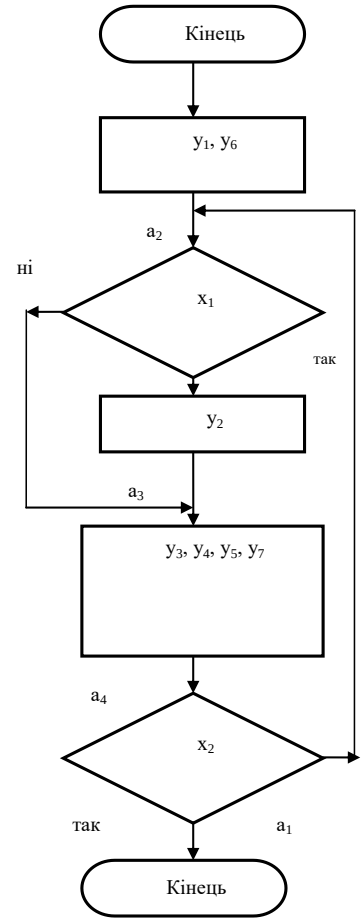


Рис. 8.38. ГСА закодованого мікроалгоритму

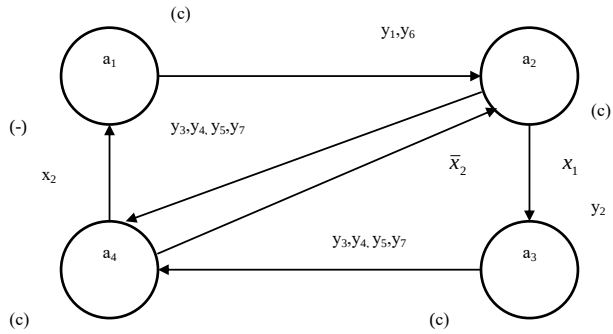


Рис. 8.39. Граф МПА

Таблиця 8.22

Структура графа МПА

Початковий стан (a_m)	Код початкового стану (Q_m)		Стан переходу (a_s)	Код стану переходу $K(a_s)$		Вхідний сигнал $x(a_m, a_s)$	Вихідний сигнал $y(a_m, a_s)$	Функції збудження тригера (a_m, a_s)	
	Q_2	Q_1		Q_2	Q_1			R	S
a_1	0	0	a_2	0	1	1	y_1	-	S_1
a_2	0	1	a_3	1	0	x_1	y_2	R_2	S_1
a_3	1	0	a_4	1	1	1	y_3	-	S_1
a_4	1	1	a_2	0	1	$\bar{x}_2$	-	R_2	-
a_5	1	1	a_1	0	0	x_2	-	R_1R_2	-
a_2	0	1	a_4	1	1	$\bar{x}_1$	y_3	-	S_2

Таблиця 8.23

Кодування елементів пам'яті

Стан автомату (a_m)	*Стан тригера*	
	Q_2	Q_1
a_1	0	0
a_2	0	1
a_3	1	0
a_4	1	1

Таким чином, на підставі табл. 8.22 отримаємо:

$$\left.\begin{aligned} y_1 &= \overline{Q}_1 \cdot \overline{Q}_2; & R_1 &= Q_1 \cdot Q_2 \cdot x_2; \\ y_2 &= Q_1 \cdot \overline{Q}_2 \cdot x_1; & R_2 &= Q_1 \cdot Q_2 \cdot \overline{x}_2 \vee Q_1 \cdot Q_2 \cdot x_2 \vee Q_1 \cdot Q_2 \cdot x_1; \\ y_3 &= \overline{x}_1 \cdot Q_1 \cdot \overline{Q}_2 \vee \overline{Q}_1 \cdot Q_2; & S_1 &= \overline{Q}_1 \cdot \overline{Q}_2 \vee \overline{Q}_1 \cdot Q_2 \vee Q_1 \cdot Q_2 \cdot x_1; \\ & & S_2 &= \overline{Q}_2 \cdot Q_1 \cdot \overline{x}_1. \end{aligned}\right\} \quad (8.7)$$

Додамо дешифратор станів автомата з наступними вихідними сигналами:

$$a_0 = \overline{Q}_1 \cdot \overline{Q}_2; \quad a_1 = \overline{Q}_1 \cdot Q_2; \quad a_2 = Q_1 \cdot \overline{Q}_2; \quad a_3 = Q_1 \cdot Q_2;$$

Отже, система рівнянь (8.7) дещо спроститься:

$$\left.\begin{aligned} R_1 &= a_3 \cdot x_2 \cdot C; \\ R_2 &= a_3 \cdot \overline{x}_2 \cdot C \vee a_3 \cdot x_2 \cdot C \vee Q_1 \cdot \overline{Q}_2 \cdot x_1 = a_3 \cdot C \vee a_2 \cdot x_1 \cdot C; \\ S_1 &= a_0 \cdot C \vee a_1 \cdot C \vee x_1 \cdot a_2 \cdot C; \\ S_2 &= a_2 \cdot x_1 \cdot C; \\ y_1 &= a_0 \cdot C; \quad y_2 = x_1 \cdot a_2 \cdot C; \quad y_3 = \overline{x}_1 \cdot a_2 \cdot C \vee a_1 \cdot C. \end{aligned}\right\} \quad (8.8)$$

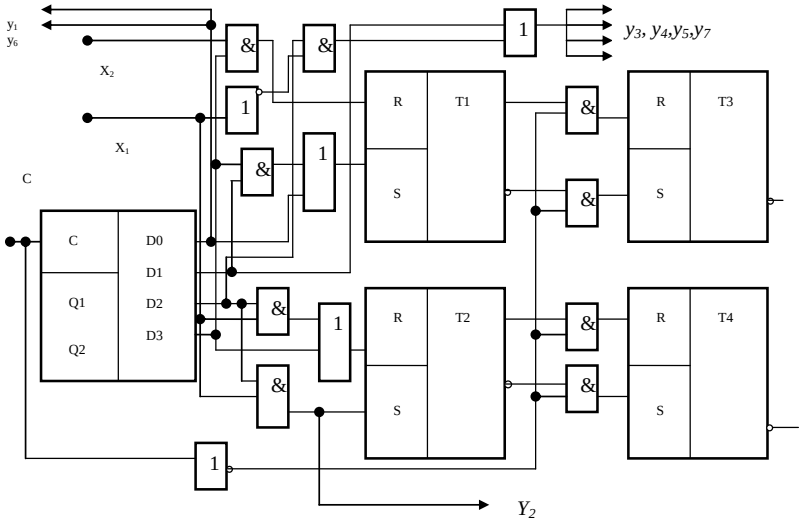


Рис. 8.40. Функціональна схема виконання операції множення двійкових чисел

На підставі одержаних виразів побудуємо функціональну схему (див. рис. 8.40.). Як дешифратор DC використовуємо демультиплексор.

Запитання для самоперевірки

1. У чому полягає відмінність між автоматами Мілі й Мура?
2. З яких етапів складається синтез цифрового автомата?

Література для самостійної підготовки за темою:

1, 2, 3, 6, 8, 9, 14, 24, 27, 31, 33.

Розділ 9. ВИКОРИСТАННЯ ЦИФРОВИХ ПРИСТРОЇВ У СИСТЕМАХ ЗАХИСТУ ІНФОРМАЦІЇ

Захист інформації в інформаційних системах - це комплекс інформаційно-технічних, організаційно-управлінських та організаційно-правових заходів, засобів, методів, які здійснюються відповідними суб'єктами інформаційних відносин для запобігання заподіянню шкоди інтересам власника інформації чи інформаційної системи та правомірних користувачів інформацією, яка обробляється, передається і/або зберігається на електронних носіях (в електронно-цифровому виразі); в разі виникнення делікту (правопорушення) застосування правового впливу для відновлення реєституції попереднього стану, покарання винного і відшкодування завданої матеріальної і моральної шкоди у відповідності з законодавством.

Одним з напрямів захисту інформації з обмеженим доступом в інформаційних системах є технічний захист інформації з обмеженим доступом із застосуванням аналогових, цифрових або аналогово-цифрових пристроїв. У свою чергу, питання технічного захисту інформації з обмеженим доступом розбиваються на два великих класи задач:

- захист інформації від несанкціонованого доступу;
- захист інформації від витоку технічними каналами.

У даному розділі розглянуті окремі питання проектування та використовування цифрових пристроїв в системах захисту інформації з обмеженим доступом, зокрема таких пристроїв, як цифрові фільтри, шифратори і дешифратори, генератори псевдовипадкових послідовностей та ін.

9.1. Використання цифрових фільтрів

В цифровій обробці сигналів цифрова фільтрація є одим із провідних напрямів досліджень. Результати науково-технічних проблем цифрової фільтрації сигналів подані у значній кількості публікацій, в тому числі [36]. У даному підрозділі наведений ряд структурних схем цифрових фільтрів.

Відомо [12], що в загальному випадку цифровий фільтр як лінійна дискретна система з постійними в часі параметрами описується наступним лінійним різницевим рівнянням:

$$Y_k = \sum_{m=0}^{M} A_m \cdot X_{k-m} - \sum_{n=1}^{N} B_n \cdot Y_{k-n}, \qquad (9.1)$$

де відповідні множини числових коефіцієнтів $\{A_m\}$ і $\{B_n\}$ є параметрами досліджуваного цифрового фільтра, а часові ряди $\{X_m\}, \{Y_n\}$ є значеннями відповідно вхідного та вихідного сигналів.

У цифрових фільтрах постійні коефіцієнти, значення вхідного та вихідного сигналів є квантованими величинами і представляються в двійковій системі числення.

При дослідженнях цифрових фільтрів виділяють:

- рекурсивні;
- нерекурсивні.

До *рекурсивних цифрових фільтрів* відносяться лінійні дискретні системи з постійними в часі параметрами, які описуються лінійним різницевим рівнянням (9.1), якщо хоча б один з коефіцієнтів B_n *(n=1,2,...N)* не дорівнює нулю. *Рекурсивний фільтр* – є пристроєм із зворотнім зв'язком, у якому кожне наступне значення Y_k вихідного сигналу залежить не тільки від значень X_{k-m} вхідного сигналу, але й від попередніх значень Y_{k-n} вихідного сигналу.

Порядок рекурсивного цифрового фільтра дорівнює $max\{M,N\}$.

Нерекурсивні (трансферні) цифрові фільтри описуються рівнянням (9.1), якщо усі B_n*=0 (n=1,2,...N)*. Подібні пристрої не мають зворотного зв'язку, а кожне значення Y_k вихідного сигналу залежить тільки від поточного і передуючого значення X_{k-m} вхідного сигналу.

Порядок нерекурсивного цифрового фільтра дорівнює M.

У класах рекурсивних і нерекурсивних цифрових фільтрів виділяють фільтри, які мають скінченну імпульсну характеристику (CIX) (Finite Impulse Responce – FIR) і нескінченну (HIX) (Infinite Impulse Response – IIR).

При синтезі цифрових фільтрів, наприклад для комплексів систем захисту інформації з обмеженим доступом, на практиці використовують Z-перетворення та його властивості для задання частотних і імпульсних характеристик фільтрів [12, 36].

Так, наприклад, прямим Z – перетворенням, або Z – образом числової послідовності $\{A_m\}$ є вираз:

$$Z[\{A_m\}] = A(z) = \sum_{m=0}^{M} A_m \cdot z^{-m}, \qquad (9.2)$$

де z – комплексна змінна.

Розглянемо ряд структурних схем цифрових фільтрів.

Цифровий фільтр, схема якого представлена на рис. 9.1, складається з наступних елементів:

- елементи для множення значень сигналів на постійні коефіцієнти;
- суматора з $M+N+1$ входами;
- елементів затримки ($M+N$), кожен з яких забезпечує затримку сигналу на один інтервал дискретизації T.

У якості елементів затримки використовують регістри (тригери). Коефіцієнти A_m, B_n та числа M, N виразу (9.1) можуть бути розраховані із заданої передатної функції [12, 36]:

$$H(\omega) = H(\omega)\exp[j\Phi(\omega)] = \frac{\sum_{m=0}^{M} A_m \exp(-jm\omega T)}{1+\sum_{n=0}^{N} B_n \exp(-jn\omega T)},$$

де $|H(\omega)|$ - амплітудно-частотна характеристика фільтра;

$\Phi(\omega)$ - фазочастотна характеристика фільтра;

T – інтервал дискретизації сигналів.

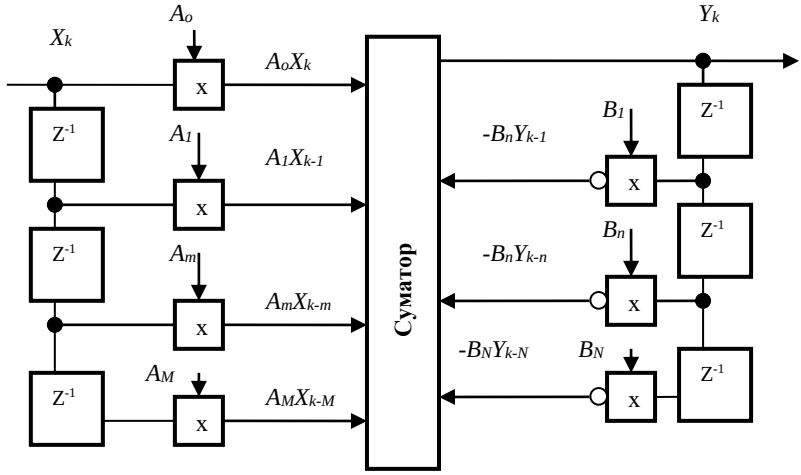


Рис. 9.1. Пряма форма реалізації цифрового фільтра

Передатна функція рекурсивного цифрового фільтра (РЦФ) розраховується за формулою [36]:

$$H(z) = \frac{\sum_{m=0}^{M} A_m z^{-m}}{1 + \sum_{n=0}^{N} B_n z^{-n}}. \qquad (9.3)$$

Якщо передаточну характеристику (9.3) цифрового фільтра представити у наступному вигляді

$$H(z) = H_1(z) \cdot H_2(z), \qquad (9.4)$$

де $H_1(z) = \frac{1}{1 + \sum_{n=1}^{N} B_n z^{-n}} = \frac{V(z)}{X(z)};$

(9.5)

$H_2(z) = \sum_{m=1}^{M} A_m z^{-m} = \frac{Y(z)}{V(z)};$

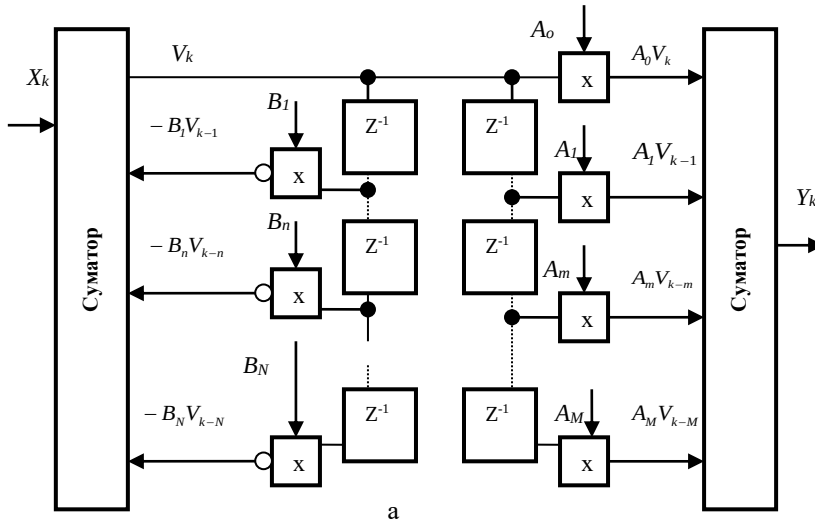
$V(z)$ - Z – образ допоміжного сигналу $\{V_k\}$.

Вирази (9.4) та (9.5) описують двохланцюговий цифровий фільтр з послідовним (каскадним) включенням ланцюгів, кожному з яких відповідає наступна система рівнянь:

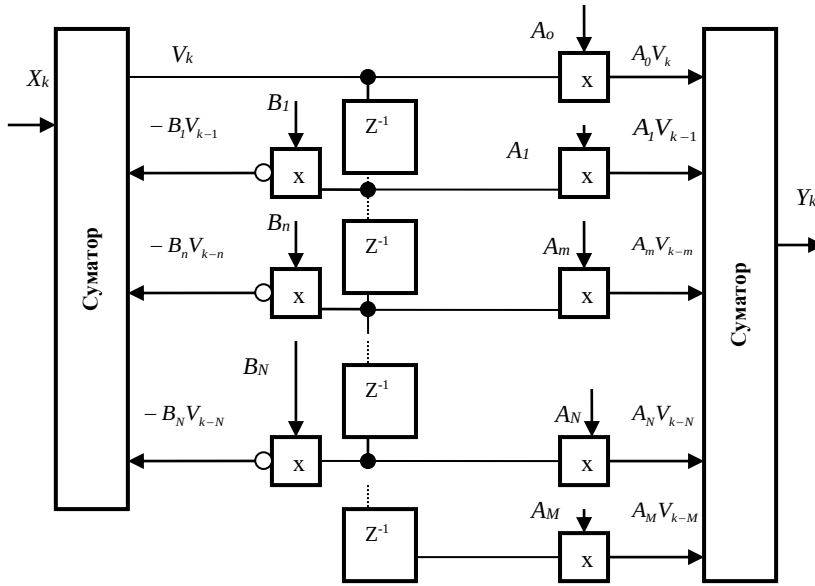
$$V_k = X_k - \sum_{n-1}^{N} B_n \cdot V_{k-n}; \quad Y_k = \sum_{m=0}^{M} A_m \cdot V_{k-m.} \qquad (9.6)$$

Реалізуючи кожен з ланцюгів у відповідності до системи (9.6), отримаємо схему фільтра, яка представлена на рис. 9.2 а, при цьому частина елементів затримки може бути об’єднана. Для випадку $M > N$ схема канонічного цифрового фільтру із зменшеною кількістю елементів затримки, представлена на рис. 9.2 б.

На практиці при великих значеннях M, N рівняння (9.1) рекурсивні цифрові фільтри будують з сукупності простих ланцюгів, що, у свою чергу, призводить до зменшення вихідних шумів, які обумовлені кінцевою розрядністю кодованих сигналів. У якості ланцюгів використовують біквадратні блоки $M = N = 2$, придатні для побудови довільних цифрових фільтрів. При цьому передатну функцію (9.3) цифрових фільтрів представляють у вигляді суми або множення передатних функцій окремих ланцюгів, що відповідає послідовній або паралельній структурі багато-ланцюгового фільтра.



а



б

Рис. 9.2. Канонічні форми реалізації цифрового фільтра

Біквадратний ланцюг РЦФ, див. рис. 9.3, на запам'ятовуючому пристрої може бути описаний наступним виразом:

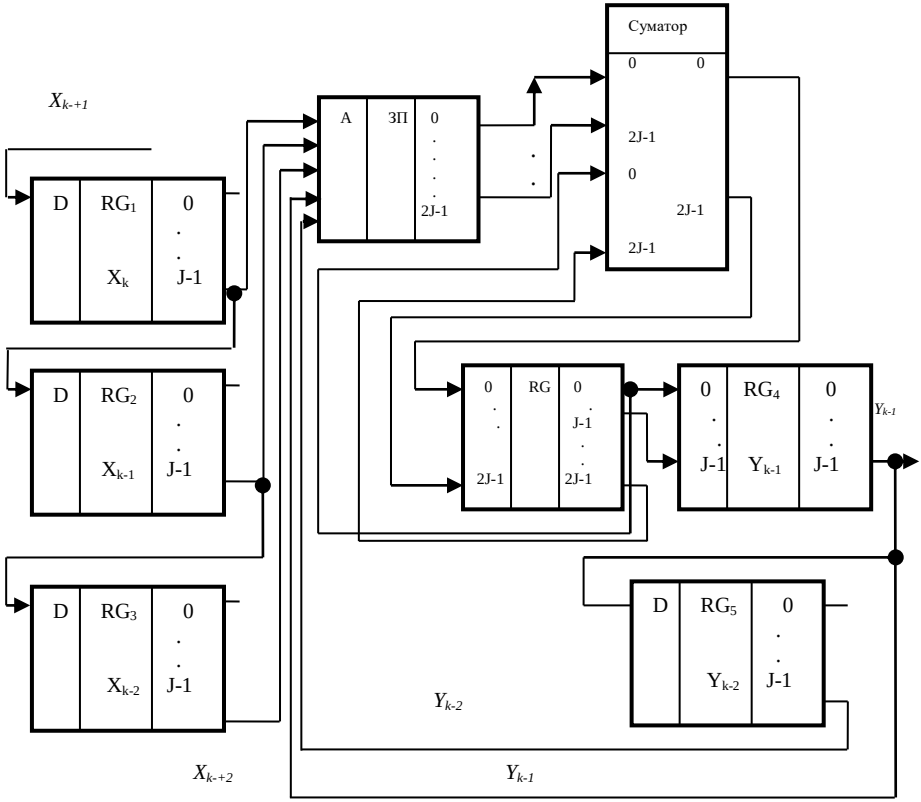


Рис. 9.3. Схема біквадратного ланцюга рекурсивного фільтра на запам'ятовуючому пристрої

$$Y_k = \sum_{m=0}^{2} A_m \cdot X_{k-m} - \sum_{n=1}^{2} B_n \cdot Y_{y-n} \,. \qquad (9.7)$$

Пристрій містить такі компоненти:

- запам'ятовуючий пристрій (ЗП) для зберігання 32-х значень функції F^j у вигляді додаткових $2J$ – розрядних кодів;
- суматор $2J$ – розрядних операндів;

- чотири *J* – розрядних послідовних регістрів (RG_1, RG_2, RG_3, RG_5) та один *J* – розрядний паралельно - послідовний регістр RG_4 для фіксації рахунку вхідних та вихідних сигналів та визначення адреси гнізда пам'яті запам'ятовуючого пристрою, у якому зберігається значення функції F^j;

- один *2J* – розрядний паралельно - послідовний регістр RG для паралельного запису результатів та їхнього зсуву вправо із збереженням змісту першого тригера регістру.

Виходи молодших розрядів тригерів регістрів $RG_1 - RG_5$ приєднані до адресного входу запам'ятовувального пристрою, який визначає гніздо із значенням функції F^j. Регістри RG_1, RG_2, RG_3 створюють *3J* - розрядний, а регістри RG_4, RG_5 - *2J* – розрядні зсувні регістри.

У початковому стані додаткові коди відліку $X_k, X_{k-1}, X_{k-2}, Y_{k-1}, Y_{k-2}$ зафіксовані на регістрах $RG_1, RG_2, RG_3, RG_4, RG_5$. Регістр RG знаходиться у нульовому стані.

На кожному такті зміст регістрів $RG_1 - RG_5$ зсувається вправо на один розряд, в результаті чого визначається адреса гнізд запам'ятовуючого пристрою, та з його виходів послідовно зчитуються додаткові коди функцій $F^{j-1}, F^{j-2}, ..., F^0$.

У суматорі на першому такті виконується додавання $F^{j-1} + 0.$ Отриманий результат заноситься у регістр RG та зсувається вправо $(F^{j-1} + 0) \cdot 2$ на один розряд.

На другому такті виконується дадавання $F^{j-2} + F^{j-1} \cdot 2$ та зсув $(F^{j-2} + F^{j-1} \cdot 2) \cdot 2$ і т.д. до $(J - 1)$-го такту включно.

На J-м такті з виходу запам'ятовуючого пристрою знімається додатковий код функції F^0, у суматорі здійснюється віднімання $Y_k = -F^0 + \sum_{j=1}^{J-1} F^j \cdot 2^{-j}$ і у регістрі RG фіксується кінцевий результат. Цей результат перезаписується у регістр RG_4 та з виходу знімається у зовнішній ланцюг. В RG здійснюється скидання тригеру. Після J тактів у

регістрах $RG_1, RG_2, RG_3, RG_4, RG_5$ будуть зафіксовані додаткові коди звітів $X_{k+1}, X_k, X_{k-1}, Y_{k1}, Y_{k-1}$.

Слід зазначити, що одним із головних елементів захисту корпоративних інформаційних мереж є міжмережеві екрани, які є однокомпонентним або комплексним засобом контролю та фільтрації пакетів даних.

Технологія фільтрації пакетів даних спочатку використовувалася на мережевому рівні, саме тому фільтрації піддавалися лише IP – адреси джерела та призначення. Зараз аналіз трафіку при фільтрації пакетів даних виконується також і на транспортному рівні.

Кожен IP – пакет досліджується на відповідність встановленим правилам. Ці правила дають дозвіл на зв'язок за змістом заголовків мережевого та транспортного рівнів моделі TCP/IP, а також на аналіз та вибір маршруту руху пакета.

Фільтри пакетів контролюють:

- фізичний інтерфейс, звідки прийшов пакет;
- IP – адресу джерела;
- IP – адреси призначення;
- тип транспортного рівня (TCP, UPD, ICMP);
- транспортні порти джерела та призначення.

При фільтрації пакетів у разі, якщо пакет відповідає правилам, відбувається його переміщення по мережевому стеку для подальшої обробки або передачі.

Всі вхідні пакети перевіряються на відповідність правилам фільтрації. Пакет знищується або йому дозволяється переміститися у мережевий стек для доставки. Фільтр пакетів не розбирає, який прикладний протокол буде використовуватися. Правила містять два списки: список заборони (*deny*) та список дозволу (*permit*).

Мережевий пакет проходить перевірку за двома списками.

Загальна схема дослідження пакетів:

- якщо правило дозволяє, пакет пропускається;
- якщо правило забороняє, пакет видаляється;
- у випадку, якщо обидва попередніх правила не можливо виконати, пакет видаляється.

Технологія фільтрації пакетів є основою для створення різноманітних засобів захисту інформації та реалізована практично у всіх типах маршрутизаторів [4, 19].

Основним недоліком технології фільтрації пакетів є той факт, що фільтри не в змозі обмежувати доступ підмножині протоколів для основних служб (команди *put, get FTP*), а також не відстежує з'єднання (не

містить інформацію про сеанс). Фільтри пакетів також мають слабкі можливості обробки інформації у самому пакеті.

9.2. Типові компоненти цифрових систем захисту інформації

Одним з найбільш поширених методів захисту інформації у телекомунікаційних мережах є маскування та шифрування.

Маскування - метод захисту інформації з обмеженим доступом із використанням інженерних, технічних засобів, а також шляхом криптографічного закриття інформації. Для маскування використовують скремблери – маскувальники аналогово-цифрові динамічні та вокодери - пристрої, що передають мову в цифровому і зашифрованому вигляді.

Шифрування - оборотне перетворення даних, з метою приховання інформації.

Для реалізації шифрування за допомогою змішаного алфавіту використовується перестановка окремих розрядів в межах одного або декількох символів.

На рис. 9.4 показана схема апаратної реалізації пристрою для шифрування, у якій використовує операція перестановки розрядів в межах одного байта інформації. Для розшифровки повідомлень використовується симетрична перестановка. Блок перестановки може бути змінним або керованим. Блок управління синхронізує роботу шифрувального пристрою. Можливе число перестановок для n-розрядних символів складає $(n!-1)$. Для шифрування за допомогою ключових слів використовується операція додавання по модулю 2. Схема шифрування показана на рис. 9.5.

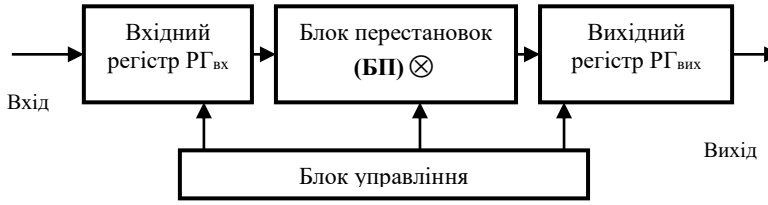


Рис. 9.4. Схема апаратної реалізації пристрою для шифрування

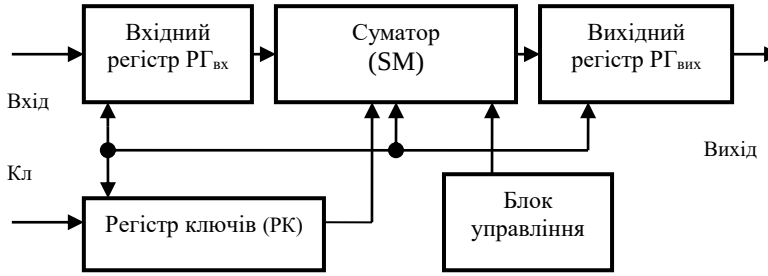


Рис. 9.5. Схема для шифрування із ключем

Ключове слово зберігається у 64-розрядному регістрі ключа (РК). Інформація, що підлягає шифруванню, записується у 64-розрядний інформаційний регістр $РГ_{вх}$. Після заповнення цього регістра виконується операція додавання mod2 з бітами регістра ключа. В результаті отримаємо зашифровану інформацію, яка надходить у вихідний 64-розрядний регістр $РГ_{вих}$. На практиці використовуються більш складні схеми з кількома суматорами mod2. Приклад такої схеми наведено на рис. 9.6.

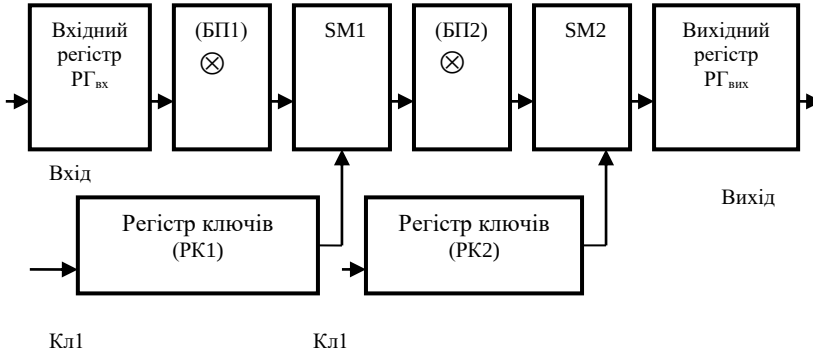


Рис. 9.6. Схема для шифрування із подвійним ключем

Ключові слова для роботи схем шифрування вибираються за допомогою спеціальних генераторів випадкових чисел і передаються в

приймальний пристрій в зашифрованому вигляді за попередніми ключами. Дешифрування інформації виконується у зворотній послідовності.

Слід зазначити, що суматори, помножувачі, шифратори та дешифратори, які використовують у цифрових системах захисту інформації з обмеженим доступом, належать до типових комбінаційних пристроїв.

Скремблери і дескремблери (шифратори і дешифратори), які використовуються у комплексах систем захисту інформації з обмеженим доступом, зазвичай побудовані на основі генераторів псевдовипадкових послідовностей бітів. Генератори частіше за все виконуються з використанням LFSR (Linear Feedback Shift Register). Фільтри цього класу давно застосовуються в телекомунікаціях для кодування та декодування двійковими захищеними кодами.

9.2.1. Суматори та помножувачі

Широке застосування спеціалізованих комп'ютерних систем в засобах криптографічного захисту інформації відіграє важливу роль при розв'язанні певного алгоритму чи класу алгоритмів, оскільки досягається висока продуктивність та надійність системи при невеликих затратах обладнання на її реалізацію та збільшується можливість реалізації даної системи на одному кристалі замовної інтегральної схеми. Операції додавання, віднімання, зміни знака виконуються за допомогою схеми суматора, а операція множення - за допомогою комбінаційного помножувача. Операція ділення, модуля, залишку і піднесення в ступінь, які відповідають зсуву об'єкта, представленого двійковим числом, реалізуються в схемі зсувача. При цьому, якщо аргумент ціле число, то зсув вправо реалізується з урахуванням знака.

При виконанні операцій додавання й віднімання розрядність результату повинна дорівнювати максимальній розрядності аргументів. Аргументи операції множення повинні мати сумарну розрядність, що дорівнює розрядності добутку.

Послідовний багаторозрядний суматор складається з одного однорозрядного суматора та елемента затримки, наприклад, D-тригера, що здійснює затримку сигналу переносу на один робочий такт – до надходження до входів суматору наступних старших розрядів доданків. Схема такого суматора показана на рис. 9.7.

Регістри зсуву в схемі призначені для передавання до входів суматора розрядів доданків у послідовному коді, починаючи з молодшого розряду, та приймання обчислених суматором розрядів суми.

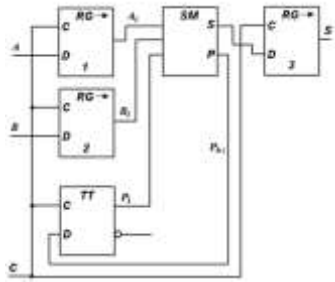


Рис. 9.7. Послідовний багаторозрядний комбінаційний суматор

З першим тактовим синхроімпульсом на входи суматору надходять із регістрів 1 і 2 цифри молодшого розряду доданків A_i і B_i, а до входу переносу подається із D - тригера логічний 0. Додаючи подані до входу цифри, однорозрядний суматор формує молодший розряд суми, який надходить до входу регістру зсуву 3 і перенос, який надходить до D-тригера. Другий тактовий імпульс реалізує в усіх регістрах зсув чисел, що зберігаються, на один розряд праворуч; при цьому на входи однорозрядного суматору надходять цифри наступних старших розрядів доданків, і цифра переносу із D-тригера. В подальшому робочі цикли повторяються до тих пір, поки не будуть підсумовані усі розряди доданків.

Явна перевага суматору послідовної дії полягає у малих апаратних витратах на його побудову. Але, такий суматор, у порівнянні з більш складним паралельним, має меншу швидкодію.

В багатьох випадках виникає потреба виконання операції одночасного додавання декількох чисел, при побудові швидкодіючих пристроїв множення, ділення, обчислення тригонометричних та інших більш складних функцій. В більшості цифрових мікропроцесорних пристроїв такі операції реалізуються циклічним виконанням послідовності елементарних операцій додавання, віднімання, зсуву та інших логічних операцій над одним або двома операндами [44, 54]. Суттєвого збільшення швидкодії можна досягти шляхом апаратної реалізації багатододанкових суматорів (БДС), призначених для додавання L (L>2) n-розрядних чисел. Для побудови таких БДС використовуються два основних підходи [15, 24, 27]:

- *багатотактний БДС* на основі накопичувального 2-входового суматора;

- *однотактний БДС,* в основу побудови якого покладене послідовне з'єднання 2-входових суматорів.

Схема БДС на основі накопичувального 2-входового суматора показана на рис. 9.8.

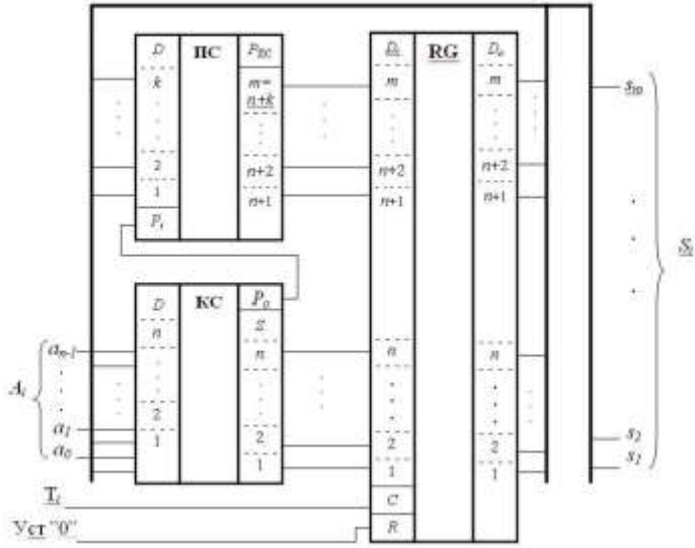


Рис. 9.8. Накопичувальний суматор

Такий суматор складається із *n*-розрядного комбінаційного суматора КС та *k*-розрядного півсуматора ПС, а також *m*-розрядного *(m=n+k)* регістра *RG* із динамічним тактовим входом *C*. Сигнал перенесення P_o із старшого розряду КС подається на вхід перенесення P_i молодшого розряду ПС. Розрядність ПС визначається таким чином, щоб забезпечити представлення максимального числа D_{max}, яке може бути отримане на виході накопичувального суматора, а саме: $D_{max}=L(2n-1)$ ($2n$-1 – найбільше можливе *n*-розрядне число *Ai*, яке може поступати на вхід КС).

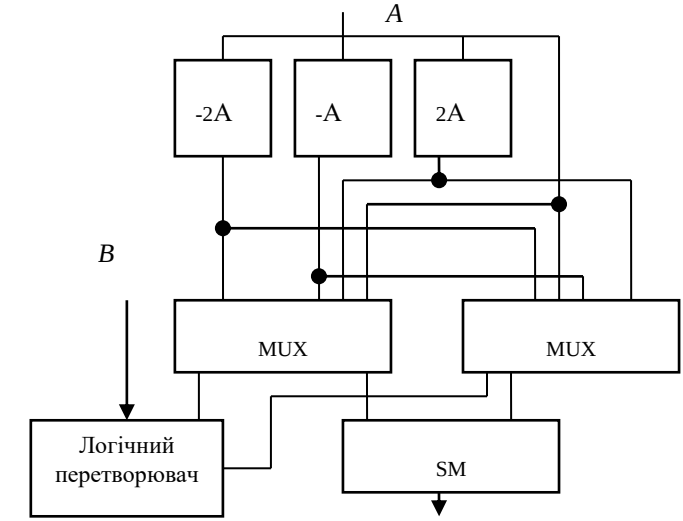
Сума *L* чисел (доданків) обчислюється за *L* тактів. Враховуючи, що для надійної роботи тривалість одного такту повинна бути в 1,5-2 рази більшою від часу формування вихідних сигналів (суми і переносу), швидкодія такого суматора невелика. Очевидно, що шляхи для підвищення швидкодії БС полягають в синтезі комбінаційних схем, в яких сума декількох чисел формується за один такт.

В [8, 14, 15, 20, 24, 27 та ін.] описана побудова матричних багатододанкових суматорів**.** Такі *L* -доданкові суматори мають багатоступеневу структуру (*L-1* ступенів) і складаються із послідовно з'єднаних 2-доданкових *m* – розрядних суматорів, створюючи при цьому

прямокутну матрицю однорозрядних суматорів. При цьому розрядність m 2-доданкових суматорів визначається розрядністю максимального значення суми. Якщо розрядність доданків дорівнює n , то розрядність суми L доданків визначається згідно виразу $m = n + k = \lfloor \log_2 L \rfloor + \lfloor \log_2 (2^n - 1) \rfloor$, де «$\lfloor\ \rfloor$» означають округлення до найближчого цілого числа, а числа k – це кількість *додаткових* розрядів для представлення суми всіх доданків.

Така побудова призводить до надлишкових апаратних витрат, тому що значення сум, які формуються на виходах проміжних ступенів, мають розрядність меншу, ніж m.

Операція множення є другою по застосуванні в обчислювальній техніці після додавання. Існує декілька методів прискорення множення: спосіб в зміні системи кодування співмножників (A, B), за рахунок чого можна скоротити кількість сумуючих часткових добутків R (алгоритм Бута) [24, 38], використання більш ефективних варіантів сумування часткових добутків, які виключають затрати часу на розповсюдження переносу та метод паралельного обчислення всіх часткових добутків [24]. Всі дані три підходи, як правило реалізуються за допомогою комбінаційних пристроїв, див. рис. 9.9.



$$P = A \cdot B = A \cdot (b_{n-1} \cdot 2^{2n-1} + b_{n-2} \cdot 2^{2n-2} + ... + b_0 \cdot 2^0)$$

часткові добутки $R = A \cdot (-2b_{i+1} + b_i + b_{i-1}) \cdot 2^i.$

Рис. 9.9. Схема швидкого множення Е.Бута

Паралельне обчислення часткових добутків має місце в усіх схемах множення великих чисел, наприклад в цифрових системах асиметричної криптографії. Різниця спостерігається в основному в способі сумування отриманих часткових добутків, і з цієї позиції використання схем множення можна поділити на матричні та багатошарові із деревоподібною структурою. Різниця між матричними та багатошаровими перемножувачами виражається в кількості використовуваних однорозрядних суматорів, їх виді та способі розповсюдження переносів, які виникають в процесі сумування.

В матричних перемножувачах сумування здійснюється матрицею суматорів, які складаються із послідовних рядків однорозрядних суматорів із збереженням переносу, див. рис. 9.10. По мірі руху даних вниз по масиві суматорів кожний рядок суматора із збереженням переносу додає до суми часткових добутків черговий частковий добуток.

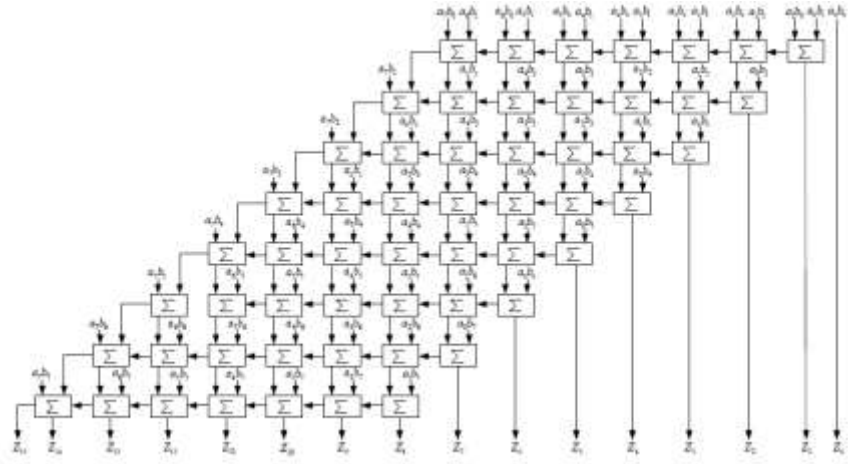


Рис. 9.10. Схема матричного перемножувача двійкових чисел з горизонтальним розповсюдженням переносу

Маючи високу швидкодію важливим досягненням матричних перемножувачів є їхня регулярність, що особливо суттєво при реалізації таких перемножувачів у вигляді інтегральної схеми. З другої сторони, подібні схеми займають велику площу на кристалі мікросхеми, причому із збільшенням розрядності співмножників ця площа збільшується пропорційно квадрату числа розрядів. Другим недоліком матричних

перемножувачівє низький рівень утилізації апаратури. По мірі руху суми часткових добутків вниз, кожний рядок задіється тільки однократно, коли його перетинає активний фронт обчислень. Ця обставина, може бути застосована для підвищення ефективності обчислень шляхом конвеєризації процесу множення, при якій по мірі звільнення рядка суматорів, останній може бути використаний для перемноження чергової пари чисел.

Зменшити затримку, яка властива матричним перемножувачам можна в схемах побудованих за деревоподібною структурою. Хоча багатошарові перемножувачі швидші матричних, однак при їх реалізації потрібні додаткові зв'язки для об'єднання розрядів, які мають однакову вагу, із-за чого площа, яку займає схема на кристалі мікросхеми може бути навіть більшою ніж у випадку матричної організації суматорів.

Апаратна реалізація алгоритму множення, в першу чергу, направлена на отримання алгоритмічного операційного пристрою (АОП) з оптимальним співвідношенням між затратами обладнання та продуктивністю даного пристрою.

9.2.2. Арифметичні операції за модулем 2^m

У цифрових автоматах операція додавання, як правило, реалізована як додавання по модулю $n = 2^m$, де *m* - ціле (зазвичай *m* = розрядності ЦА). Для отримання в двійковій системі $A + B \bmod 2^m$ досить скласти числа, після чого відкинути розряди починаючи з *m*-го і старше.

У загальному випадку операцію додавання по модулю можливо записати наступним чином - $(A + B) \bmod n$ - це залишок від ділення суми *A* + *B* на *n*, де *A* і *B* – відповідні числа.

Додавання двох чисел по модулю *n* представляється в двійковій системі числення, як *S*-блок (див. главу 10, рис. 10.3), у якого на вхід подається число *A*, а в якості системи комутації *S*-блоку використовується циклічний зсув вліво на *B* розрядів.

Операція множення по модулю *n* $(A \cdot B) \bmod n$ - це залишок від ділення добутку $(A \cdot B)$ на *n*.

У персональних комп'ютерах на платформі x86 при перемножуванні двох *m*-розрядних чисел виходить число розрядністю $2 \cdot m$. Щоб отримати залишок від ділення на 2^m потрібно відкинути *m* старших біт.

Розглянемо приклад. Суматор за модулем обчислює суму $Z = (A + B)_q$, де число *Z* дорівнює залишку від ділення суми *(A+B)* на число *q*. Числа *Z, A, B*, і *q* зображають в двійковій формі і мають

розрядність n. Необхідно синтезувати суматор за модулем q для будь-якого значення n.

Розглянемо суму $S = (A + B) + (2^n - 1),$ де S має $n+1$ двійкових розрядів. Очевидно, що сума S може приймати значення $S < 2^n$ *і* $S \geq 2^n$ залежно від значень A і B. Якщо сума $S < 2^n$, то $(n+1)$ розряд числа S дорівнює **0**, отже, $Z=A+B$. Якщо ж сума $S \geq 2^n$, то $(n+1)$ розряд числа S дорівнює **1,** а отже, $Z=A+B_{-q}$.

Таким чином існує співвідношення

$$Z = (A + B)_q = \begin{cases} A + B, \textit{ якщо} \quad A + B < q; \\ A + B - q, \textit{ якщо} \quad A + B \geq q. \end{cases}$$

На основі останнього співвідношення може бути побудована схема суматора за модулем, де q – будь-яке просте число. На рис. 9.11 показана схема суматора для n-розрядного числа q. Суматор *SM1* виконує обчислення суми чисел A і B, суматор *SM2* віднімає від суми $A+B$ значення q, оскільки 2^n-q – доповнення числа q до 2^n. Мультиплексор подає на вихід суму $Z=(A+B)_q$ залежно від розряду s_{n+1} (а ним буде розряд переповнення одного з суматорів).

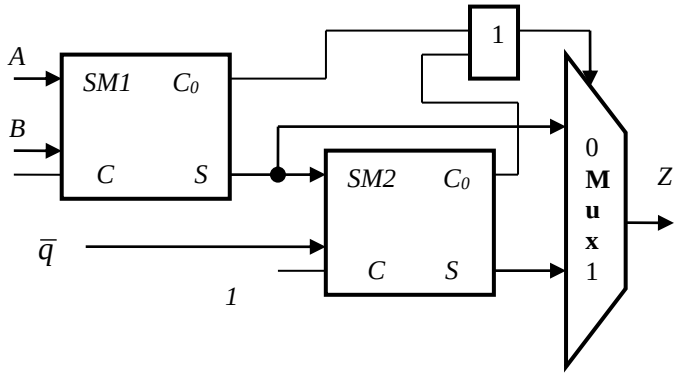


Рис. 9.11. Схема суматора для n-розрядного числа q

Для добутку чисел $A \cdot B$ існує співвідношення

$$A \cdot B = \sum_{p=1}^{n} y_p \cdot A \cdot 2^{p-1},$$

де $y_p = 0$ *або* 1. З цього випливає, що для побудови пермножувача за модулем *q* необхідно синтезувати

типову схему, яка виконує операцію $(2 \cdot A)_q$ – перемноження на 2 за модулем *q*. Правило побудови схеми такого перемножувача отримуємо з попередніх співвідношень для суматора за модулем, якщо візьмемо *A*=*B* і $S = 2 \cdot A + (2^n - q)$. На рис. 9.12 показана схема перемножувача на 2 за модулем для *n*-розрядного *q*. Перемноження числа *A* на 2 досягається зсувом розрядів числа *A* на один розряд відносно входів суматора, а

тому потрібний лише один суматор. На виході мультиплексора отримуємо величину $Z = (2 \cdot A)_q$.

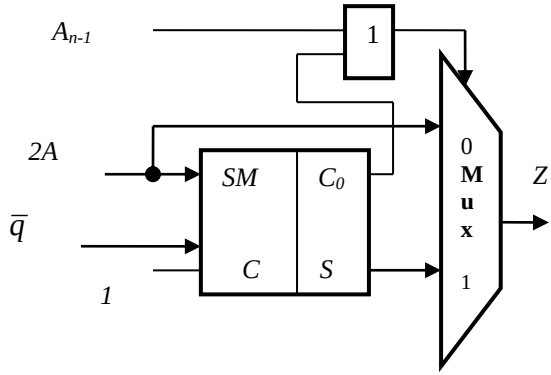
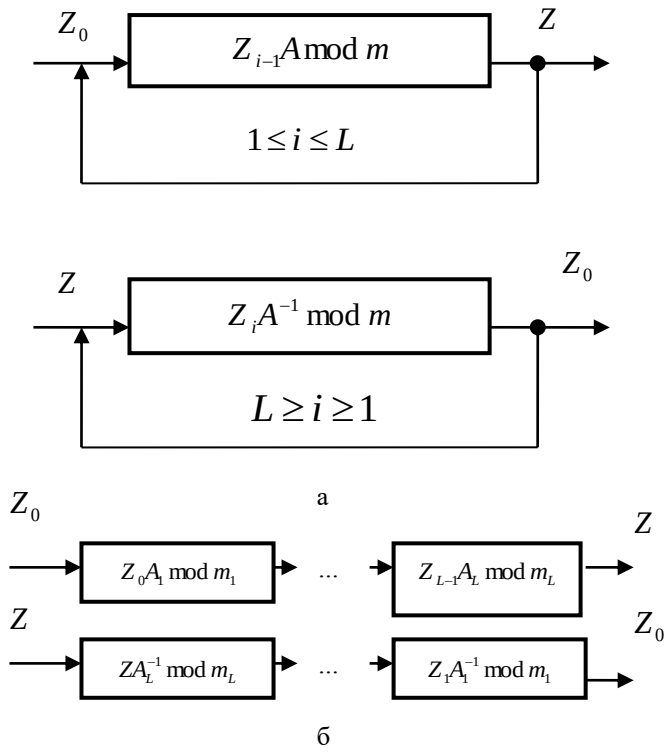


Рис. 9.12. Схема перемножувача на 2 за модулем для *n*-розрядного *q*

Тоді $Z = (A \cdot B)_q = (\sum_{p=1}^{n} y_p \cdot A \cdot 2^{p-1})q,$ де *q* і *Z* – *n*-розрядні двійкові числа.

Відомі блокові шифри [4, 19, 26, 39, 40, 48, 49 та ін.] реалізуються на основі сукупності арифметичних і логічних операцій та зсувів кодів. Однак найефективніше сучасні мікропроцесори реалізують арифметичні операції. Тому розробники перспективних блокових шифрів намагаються використовувати саме арифметичні операції за модулем 2^m для шифрування інформації.

Схематично процес шифрування і розшифрування із використанням операції додавання за модулем 2^m показано на рис. 9.13.



а - шифрування та розшифрування *A,m=const;*
б –шифрування та розшифрування *A,m=var*

Рис. 9.13. Процес шифрування і розшифрування із використанням операції додавання за модулем 2^m

Де Z_0 – *n*-розрядний блок, тоді його шифрування відбувається шляхом обчислень за формулою: $Z = Z_0 A \bmod m$, для НСД *(A,m)=1*. Відповідно розшифрування блоку Z виконується за формулою: $Z_0 = ZA^{-1} \bmod m$. При цьому ключ для шифрування є конкатенацією $K = A \mid m$, , а ключ розшифрування – $K = A^{-1} \mid m$.

Якщо розрядність інформаційного блоку n_x, то розрядність коефіцієнта *A* доцільно обирати такою ж самою $n_x = n_A = n$, тоді розрядність модуля *m*

має дорівнювати $n_m=n+1$, тобто в деяких випадках зашифроване повідомлення може мати на один розряд більше порівняно з початковим блоком.

Інший підхід до побудови блокового шифру полягає в тому, що замість виконання L раундів обчислень з однаковими значеннями A і m пропонується здійснювати L раундів обчислень, в кожному з яких використовується інше значення m і A. При цьому має виконуватись умова $m_1<m_2<...<m_L$. У даному випадку для кожного раунду використовується свій окремий секретний ключ $K_i = A_i \Big| m_i, i = \overline{1 \div L}.$

Для багатьох цифрових систем важливою задачею є мінімізація витрат обчислювальних ресурсів на допоміжну обробку інформації, до якої належить і захист інформації. В розглянутих засобах захисту основна частина обчислювальних ресурсів витрачається на реалізацію булевих функціональних перетворень. Підвищення ефективності може бути досягнуто за рахунок використання булевих функціональних перетворень великої розрядності. При цьому необхідно вирішити дві взаємопов'язані задачі – побудови перетворень великої розрядності з певними властивостями та продуктивної реалізації таких перетворень програмними і апаратними засобами (оскільки традиційна таблична реалізація в даному випадку технологічно неможлива).

9.2.3. Шифратори

Шифратор $M \times N$ (Coder - CD) - це комбінаційний пристрій із M входами та N виходами, який перетворює M – розрядний унітарний код у N - розрядний двійковий код [14, 21, 30].

За кількістю входів розрізняють наступні шифратори:

- повні шифратори, в яких кількість входів $M = 2^N$;
- неповні шифратори, які мають $M < 2^N$ кількість входів.

За рівнями вхідних та вихідних сигналів розрізняють:

- шифратори високого рівня, активні сигнали на входах та виходах яких мають рівень логічної 1;
- шифратори низького рівня, активні сигнали на входах та виходах яких мають рівень логічного 0.

За функціональною значущістю входів шифратори можуть бути поділені на дві групи:

- шифратори з рівнозначними функціями входів;
- приоритетні шифратори.

Структурна формула шифратора у мінімальній диз'юнктивній нормальній формі у базисі **«І-НЕ»** виглядає так:

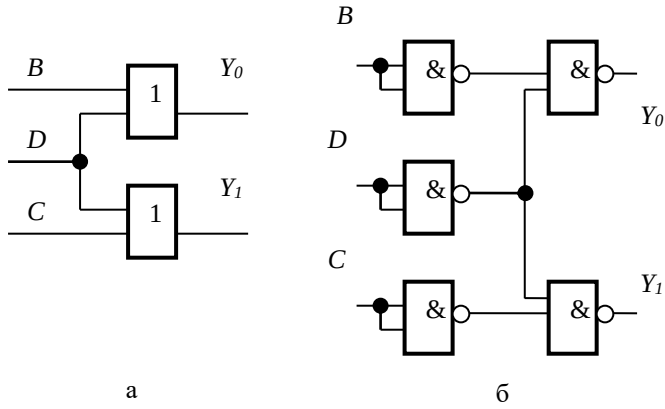
$$Y_1 = D + C = \overline{\overline{D} \cdot \overline{C}}, \quad Y_0 = D + B = \overline{\overline{D} \cdot \overline{B}}.$$

Схеми шифраторів, побудованих за наведеними структурними формулами, представлені на рис. 9.14.

Структурні формули для пріоритетного шифратора виглядають таким чином:

$$Y_1 = D + \overline{D} \cdot C = D + C = \overline{\overline{D} \cdot \overline{C}},$$

$$Y_0 = D + \overline{D} \cdot \overline{C} \cdot B = D + \overline{C}B = \overline{\overline{D} \cdot \overline{\overline{C} \cdot B}}.$$



а - шифратор на елементах **«АБО»**; б - шифратор на елементах **«І-НЕ»**
Рис. 9.14. Схеми шифратора

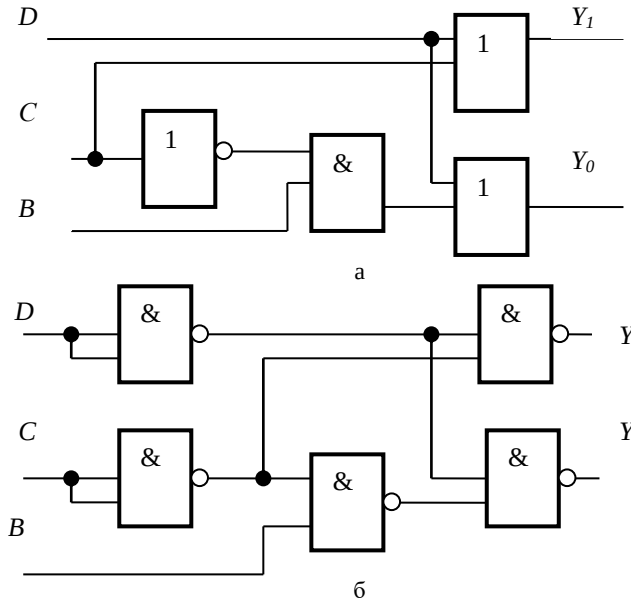
Схеми шифраторів, побудованих за наведеними структурними формулами, представлені на рис. 9.15.

У цифрових пристроях систем захисту інформації в шифраторах на інтегральних схемах часто використовують наступні додаткові сигнали:

вхідний сигнал включення шифратора – *EI (Enable Input).* Сигнал *EI=1* дозволяє включити шифратор для прийому інформаційних вхідних сигналів $X_m = \{A, B, C, D.....\}$. Сигнал *EI=0* дозволяє вимкнути шифратор для зміни вхідних сигналів;

вихідний сигнал (груповий сигнал) – *GS (Group Signal),* фіксує при включеному шифраторі (*EI=1*) та (*GS =1*) активний стан одного з входів ($X_m = 1$) шифратора та появи вихідного коду $Y_{N-1}...Y_1Y_0$;

вихідний сигнал дозволу – *EO (Enable Output).* При *EO=1* та включеному шифраторі (*EI=1*) сигнал вказує на пасивний стан усіх входів шифратора.



а - шифратор 4×2 на елементах **«НЕ», «АБО»** та **«І»**;
б - шифратор на елементах **«І-НЕ»**
Рис. 9.15. Схеми пріоритетного шифратора 4×2

Структурні формули для пріоритетного шифратора мають такий вигляд (де $X_o = A, X_1 = B, X_2 = C, X_3 = D.$)

$$EO = EI \cdot \overline{D} \cdot \overline{C} \cdot \overline{B} \cdot \overline{A} = \overline{\overline{EI} + D + C + B + A};$$

$$GS = EI \cdot (\overline{EI} + D + C + B + A) = EI \cdot \overline{EO} = \overline{\overline{EI} + EO};$$

$$Y_1 = EI \cdot (D + \overline{D} \cdot C) = EI \cdot (D + C);$$

$$Y_0 = EI \cdot (D + \overline{D} \cdot \overline{C} \cdot B) = EI \cdot (D + \overline{C} \cdot B).$$

Схема пріоритентого шифратора показана на рис. 9.16.

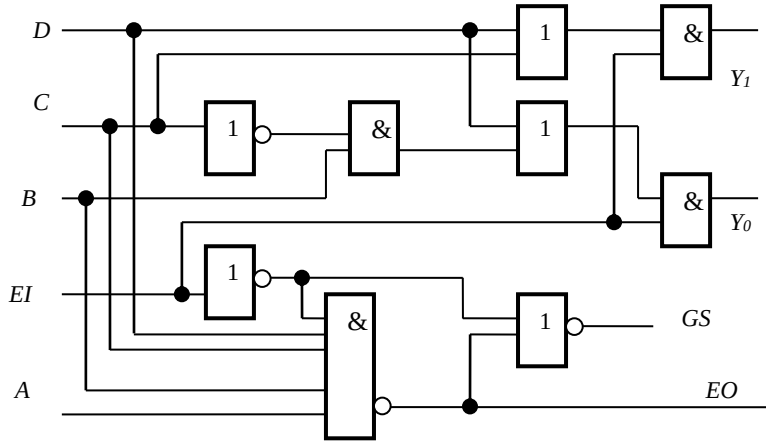


Рис. 9.16. Схема пріоритентого шифратора

9.2.4. Дешифратори

Дешифратор $M \times N$ (Decoder - DC) - це комбінаційний пристрій із *M* входами та *N* виходами, який перетворює *M* - розрядний двійковий код у *N* - розрядний унітарний код [12,22,30]. У дешифраторах високого рівня унітарний код містить тільки одну 1, в дешифраторах низького рівня тільки один 0. Максимальна кількість виходів - $N = 2^M$ відповідає усім можливим наборам сигналів на виході дешифратора або *M* - розрядним двійковим кодам. Дешифратор із максимальним значенням $N = 2^M$, має назву повного, а із $N < 2^M$ - неповним.

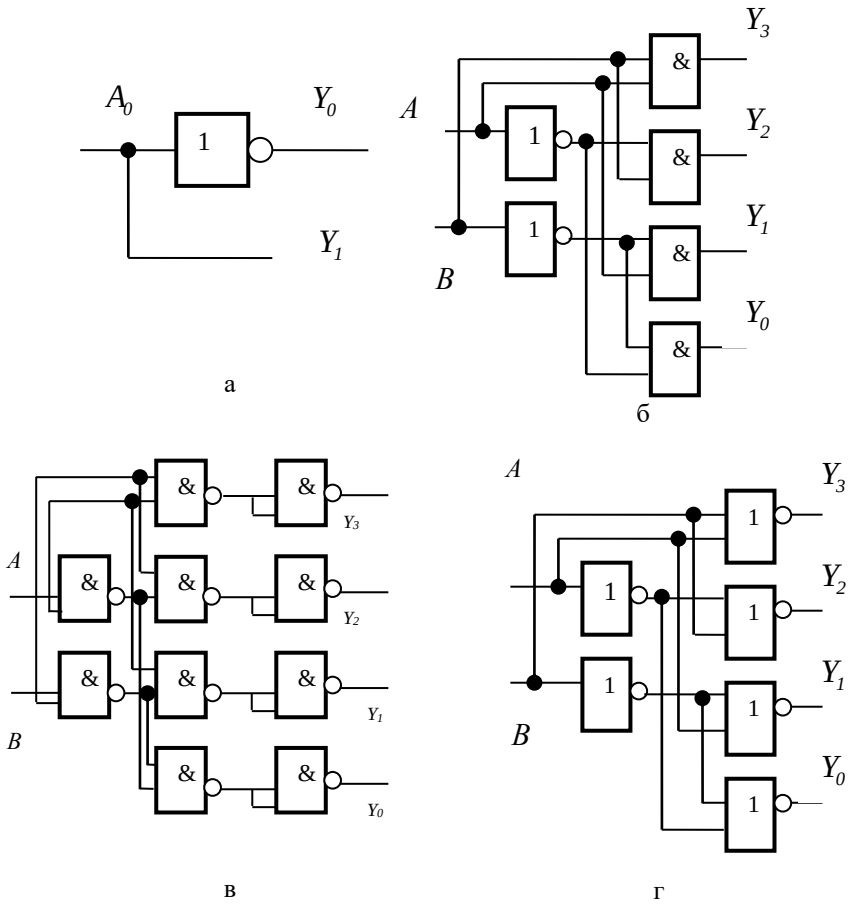
Схеми дешифраторів 1×2 та 2×4 наведені на рис. 9.17.

Структурна формула для дешифратора 2×4 у досконалій диз'юнктивній нормальній формі виглядає так (де $X_o = A, X_1 = B, X_2 = C, X_3 = D.$) $Y_o = \overline{B} \cdot \overline{A}$, $Y_1 = \overline{B} \cdot A$, $Y_2 = B \cdot \overline{A}$, $Y_3 = B \cdot A$.

Наведені вирази відповідають схемі дешифратора, побудованій на логічних елементах **«НЕ»** та **«І»**, див. рис. 9.15 а.

Для дешифратора, побудованого на елементах **«АБО-НЕ»**(див. рис. 9.15 г), структурна формула виглядає таким чином:

$$Y_o = \overline{B + A}, \quad Y_1 = \overline{B + \overline{A}}, \quad Y_2 = \overline{\overline{B} + A}, \quad Y_3 = \overline{\overline{B} + \overline{A}}.$$



а - дешифратор 1×2; б - дешифратор 2×4 на елементах **«НЕ»** та **«І»**;
в - дешифратор на елементах **«НЕ-І»**; г - дешифратор на елементах **«АБО-НЕ»**

Рис. 9.17. Схеми повних дешифраторів

Дешифратори низького рівня формують макстерми $M_{i,M}$, які можливо виконати у вигляді логічної схеми суми M вхідних змінних X_m $(m = 0,1,2,...,M-1)$. Змінна X_m входить до складу макстерму у прямому вигляді, якщо її значення на *i*-му наборі дорівнює 0, та в інверсному вигляді, якщо значення змінної дорівнює 1.

9.3. Генератори псевдовипадкової послідовності

Псевдовипадкова послідовність (ПВП) – це послідовність бітів, у якій ймовірності появи логічного «0» або «1» рівні між собою і не залежать від попередніх бітів. Тобто псевдовипадкова послідовність – послідовність двійкових символів, схожа на випадкову. Цю схожість можна перевірити за цмовірнісними (статистичними) характеристиками відомими методами.

Псевдовипадкові послідовністі мають ще одну властивість, не менш важливу щодо придатності використання в захищених телекомунікаційних системах. Це можливість точного відтворення (генерації) псевдовипадкової послідовністі будь-яку кількість разів по можливості простими апаратними або програмними засобами. Саме таким класичним засобом є лінійні цифрові фільтри зі зворотними зв'язками по модулю 2. Не зупиняючись на теоретичних аспектах синтезу LFSR, зазначимо лише інші важливі сфери їх застосування в телекомунікаційних системах.

Маскування реальних статистичних характеристик повідомлення ("забілювання") шляхом побітового складання по модулю двох псевдовипадкових послідовністей до двійкового повідомлення. Ця процедура суттєво зменшує труднощі тактової синхронізації в мережах цифрової передачі даних та ускладнює криптоаналіз в процесі шифрування.

Діагностування технічного стану цифрових пристроїв методами сигнатурного аналізу. У цьому випадку до псевдовипадкових послідовністей висуваються досить помірні вимоги: довжина послідовності $2^{20}...2^{30}$ біт. Ці вимоги без проблем задовольняються реалізацією на LFSR.

Потокове шифрування повідомлень шляхом скремблювання. По суті, сама процедура співпадає із "забілюванням" (п.1), але мета і вимоги до псевдовипадкових послідовністей принципово інші. У цьому випадку довжина послідовності повинна бути якомога більшою, оскільки саме псевдовипадкова послідовність є ключем шифрування. А "зламати" такий шифр можна лише шляхом підбору ключа або обчисливши параметри генератора псевдовипадкових послідовнісої.

Для того щоб на виході генератора формувалась псевдовипадкова послідовність бітів з періодом повторення рівним $2^{TP}-1$ (TP – степінь твірного поліному), необхідно обирати точки підключення кола зворотного зв'язку відповідно до степенів примітивних твірних поліномів, що описують ряд генераторів різної розрядності.

Псевдовипадкова послідовність бітів з періодом повторення, рівним $2^{TP}-1$, має такі властивості [26, 30]:

– у повному циклі ($2^{TP}-1$ тактів) число логічних одиниць, що формуються на виході генератора, на одиницю більше. ніж число логічних нулів. Додаткова логічна одиниця з'являється за рахунок виключення стану, при якому в регістрі був би присутній нульовий код. У результаті ймовірності появи логічного нуля і логічної одиниці на виході генератора практично однакові;

– у повному циклі половина серій з послідовності логічних одиниць має довжину 1, четверта частина серій – довжину 2, восьма частина – довжину 3 і т.д. Такі ж ознаки властиві логічному нулю з урахуванням одного пропущеного логічного нуля. Це свідчить про те, що ймовірність появи 1 чи 0 не залежить від попередніх значень. Тому ймовірність того, що серія з послідовних логічних одиниць або нулів закінчиться при наступному кроці, складає 0,5;

– якщо послідовність повного циклу порівнювати з цією ж послідовністю, але циклічно зсунутою на будь-яке число тактів W (W не є нулем, або числом, кратним), то число неспівпадань буде на одиницю більшим, ніж число співпадань.

LFSR є гарними генераторами псевдовипадкових послідовностей, але вони мають деякі суттєві недоліки. Так, для LFSR довжини TP внутрішній стан представляє собою попередні TP вихідних бітів генератора. Навіть якщо схема зворотного зв'язку зберігається в секреті, вона може бути визначена по $2TP$ вихідним бітам генератора за допомогою деяких високоефективних алгоритмів, наприклад Berlekamp-Massey [26]. Існують також інші методи "зламу" шифрів на основі LFSR шляхом так званих "алгебраїчних атак" [39]. Очевидним шляхом захисту від таких атак є передусім суттєве збільшення довжини псевдовипадкової послідовності, яка використовується для потокового шифрування.

Використовуючи стандартну схему, збільшення періоду генерації псевдовипадкових послідовностей більше ніж до 2^{TP} не представляється можливим. Але можна запропонувати процедуру, при якій після закінчення періоду генерації для одного примітивного полінома, в роботу системи включався б інший. Таким чином для системи певної розрядності період генерації збільшувався б удвічі. Отже, можна створювати системи, в яких період генерації залежав би тільки від кількості примітивних поліномів. Схема такої системи представлена на рис. 9.18.

В такому генераторі в регістрі пам'яті зберігаються потрібні примітивні поліноми. Вузол управління в свою чергу, залежно від розрядності системи, вирішує, коли саме "дістати" з регістру пам'яті наступний примітивний поліном і задіяти його в системі.

Корисним є застосування реверсивності в генераторах псевдовипадкових послідовностей [26]. Реверсивне функціонування

регістрів зсуву дуже просто реалізується і при цьому не викликає ніяких незручностей. При реверсивному функціонуванні число псевдовипадкових послідовністей може бути збільшене вдвічі. При цьому, якщо зсув інформації у вказаних регістрах відбувається вправо, то пристрій генерує одні псевдовипадкові послідовності, а при зсуві вліво – інші псевдовипадкові послідовності, “зворотні” до перших.

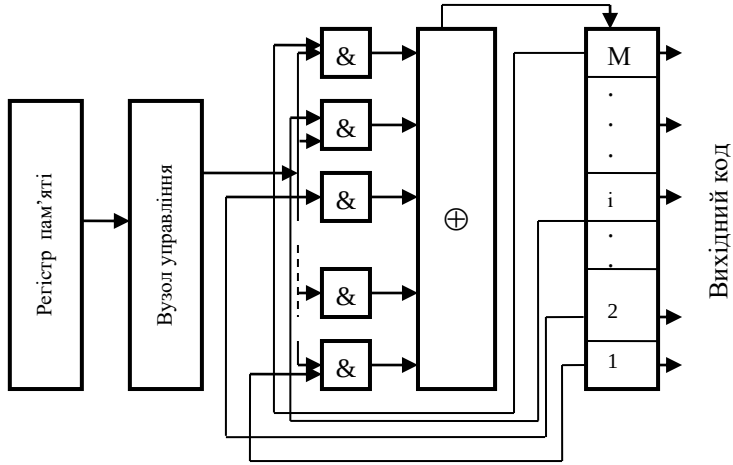


Рис. 9.18. Генератор псевдовипадкової послідовності

Функцію вузла управління може брати на себе розподілювач імпульсів [30]. При цьому зникає проблема виявлення твірних поліномів або певних закономірностей в утвореній послідовності. Розподілювач імпульсів використовується разом з комутатором. Комутатор представляє собою набір логічних елементів **«АБО»** із заздалегідь визначеною логікою. Розподілювач імпульсів, до складу якого також входить регістр зсуву, за рахунок комутатора створює унікальну кодову послідовність. Функцію розподілювача може виконувати мікроконтролер. Послідовність, утворена за рахунок використання такого ключа, має досить низький рівень корельованості розрядів.

Викриття структурної схеми генератора псевдовипадкових послідовністей з розподілювачем генератора шляхом аналізу вихідної послідовності представляє собою надзвичайно складну задачу, та у межах

даного навчальний посібника не розглядається. Загальна структурна схема генератора представлена на рис. 9.19.

Пристрій для формування двійкових псевдовипадкових послідовністей на основі такого генератора дозволяє отримати одну або декілька форм псевдовипадкових послідовністей при значно менших технічних витратах в порівнянні з відомими аналогічними пристроями.

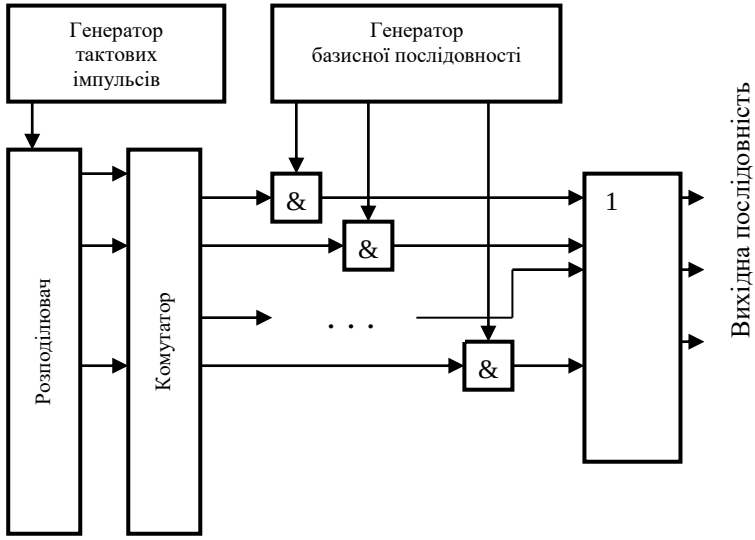


Рис. 9.19. Структурна схема генератора псевдовипадкової послідовності

Ще один варіант генератора псевдовипадкових послідовністей [26, 37, 39] формує псевдовипадкову послідовність бітів з періодом повторювання, рівним 2^{TP}. Це здійснюється за допомогою складання нульового стану регістра зсуву. В регістрі **RG** в певному порядку формуються всі можливі коди, включаючи нульовий.

Генератор додатково містить елемент **«АБО-НЕ»**, інвертор і мультиплексор **MS**.

Сигнал *Z* на виході елементу **«АБО-НЕ»** задає напрям передачі даних через мультиплексор. При *Z=0* на вихід мультиплексора транслюється

сигнал з виходу елемента **«АБО, що виключає»**, а при *Z=1* – сигнал з виходу інвертора, див. рис. 9.20.

До моменту, доки на входах елементу **«АБО-НЕ»** присутня хоч одна логічна одиниця. на його виході буде сигнал *Z=0*. В цьому випадку мультиплексор **MS** в кожному такті передає в звільнений (нижній) розряд регістра зсуву біт з виходу елемента **«АБО, що виключає»**.

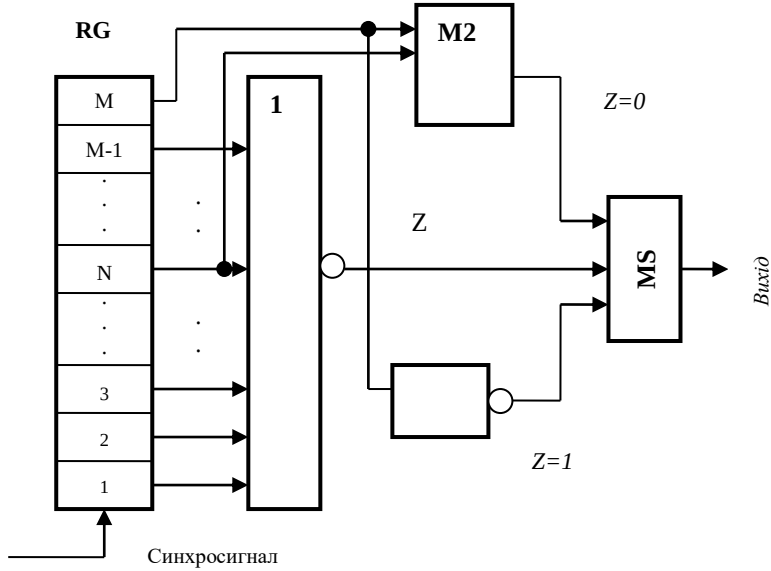


Рис. 9.20. Генератор псевдовипадкової послідовності з періодом повторення 2^M

В деякому такті i в регістрі фіксується код, що містить єдину логічну одиницю, розміщену в розряді $TP-1$. Оскільки в розрядах TP і N присутні логічні нулі. то на виході елемента **«АБО, що виключає»** сформований сигнал логічного нуля. який до початку такту $i+1$ надходить на вхід регістра. На початку такту $i+1$ логічна «1» переміщується з розряду $TP-1$ в розряд TP, на входах **«АБО-НЕ»** формується нульовий код. Сигнал *Z=1* переводить мультиплексор **MS** в стан, при якому на вхід нижнього розряду регістра зсуву надходить біт з виходу інвертора. В даному випадку цей біт рівний нулю, тому в такті $i+2$ в регістрі фіксується нульовий код.

До початку такту $i+3$ на вхід регістра зсуву з виходу інвертора надходить логічна **«1»**, тому по фронту синхросигналу в регістрі фіксується код, що містить логічні **«0»** в усіх розрядах, окрім першого. Сигнал Z знову приймає нульове значення, мультиплексор переключається в стан передачі сигналу з виходу елемента **«АБО, що виключає»** і т.д. Таким чином регістр проходить через усі стани, включаючи нульовий стан.

На завершення розглянемо схему з двома фільтрами LFSR $F1$ та $F2$, де $F1$ працює як генератор псевдовипадкових послідовністей, а $F2$ змінює кожну згенеровану двійкову комбінацію шляхом фільтрації через сукупність двоходових логічних схем $f_1, f_2, ..., f_i, f_M$, див. рис. 9.21. В залежності від виду конкретних функцій $f_i = f_i(x_i, y_i)$, $i = 1,2,...M$ та конкретної послідовності, яка створюється $F2$, сумарна псевдовипадкова послідовність може змінюватися в широких межах.

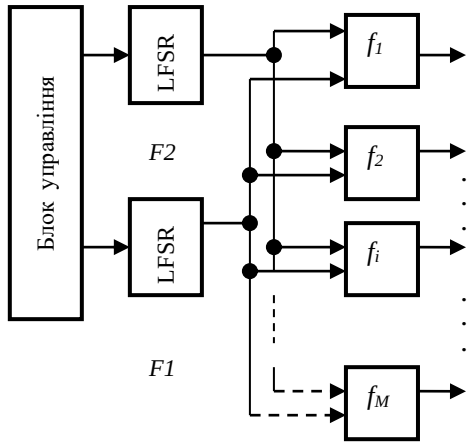


Рис. 9.21. Генератор псевдовипадкової послідовності з двома LFSR

Інші булеві функції (див. розділ 5) двох змінних потребують аналізу отриманих псевдовипадкових послідовністей «на випадковість». По суті, мова піде про перевірку появи в псевдовипадкових послідовністей легко прогнозованих закономірностей.

Зазначимо також, що у будь-якому випадку розглянуті варіанти нових процедур генерації псевдовипадкових послідовністей дають можливість суттєво ускладнити криптоаналіз як за рахунок збільшення довжини циклу псевдовипадкових послідовністей, так і комбінаторного

різноманіття саме процедур генерації. Можна сподіватися, що застосування нових алгоритмів генерації псевдовипадкових послідовністей дозволить ефективно протистояти атакам на потокові шифри класу скремблерів.

Таким чином, наведені в даному розділі схеми генерації псевдовипадкових послідовністей тісно пов'язані з системами скремблювання даних. Запропоновані схеми реалізації псевдовипадкових послідовністей представляють собою надзвичайно прості і, водночас, дієві засоби захисту інформації з обмеженим доступом у випадку, якщо час актуальності інформації обмежений. Тому їх застосування може бути корисним при реалізації систем захисту інформації з обмеженим доступом в телекомунікаційній апаратурі.

9.4. Генератори на основі регістрів зсуву

В криптосхемах потокових шифрів широко застосовуються криптовузли, засновані на регістрах зсуву із зворотним зв'язком [26, 39, 40, 48, 49 та ін.].

Регістр зсуву із зворотним зв'язком складається з двох частин: регістра зсуву і функції зворотного зв'язку.

Двійковий регістр зсуву - це послідовність бітових осередків. Їх кількість називається завдовжки регістра. Під час роботи вміст осередків змінюється. В результаті одного такту роботи регістра генерується один біт. Новий біт обчислюється як функція від бітів, вибираних з осередків регістра з наперед певними номерами. Вказані осередки називаються осередками зворотного зв'язку, а функція - функцією зворотного зв'язку. Номери осередків зворотного зв'язку називаються точками знімання зворотного зв'язку.

В такті роботи обчислюється значення функції зворотного зв'язку, потім регістр зсовується, скажімо, вліво, втрачаючи лівий крайній розряд і звільняючи крайній правий осередок. В цей осередок поміщається значення функції зворотного зв'язку. Виходом регістра є біт, знятий з фіксованого (звичайно, з крайньою правою) осередку.

В потокових шифрах генератори гамми, в більшості випадків, складаються з типових вузлів, заснованих на комбінаціях регістрів зсуву і функціях ускладнення [40, 48, 49].

Найпростішим вузлом є т.з. регістр зсуву з лінійними зворотними зв'язками (РЗЛЗЗ), що генерує рекурентну послідовність вигляду $x_{i+0} \oplus x_{i+k} \oplus \ldots \oplus x_{i+t} = x_{i+n}$.

Приклад розгортки РЗЛЗЗ з рекурентними співвідношенням $x_{i+0} \oplus x_{i+2} \equiv x_{i+5} \bmod 2$:

110001101110101000010010110011**11000**1101110101000010010

Де $n=5$ – довжина регістра (кількість елементів в початковому заповненні), 0, 2 – параметри рекурентного закону (точки знімання зворотного зв'язку регістра).

Подібні послідовності періодичні. При відповідному виборі параметрів РЗЛЗЗ можна досягти максимально можливих значень періоду рівних $2^n - 1$.

Безпосередньо для генерації гамми РЗЛЗЗ не підходять. В сучасних криптографічних системах застосовуються комбінації залежних РЗЛЗЗ, взаємно впливаючих на формування своїх послідовних заповнень.

Для синтезу криптосхеми потокових шифрів у цифрових системах необхідно враховувати не тільки загрозу дешифрування для відомих типів криптоатак, але й можливість так званої компрометації шифру. Поняття компрометації полягає в тому, що невдалий вибір деяких параметрів криптографічної системи часто дозволяє зловмиснику ідентифікувати шифр за шифротекстом.

Проілюструємо суть поняття компрометації шифру на елементарному прикладі.

Припустимо, що криптосхема потокового шифру гамування по $\bmod 2$ генерує гамму як вихід з РЗЛЗЗ, а ключем є початкове заповнення. В цьому випадку шифротекст є спотвореною рекурентної послідовністю і задача дешифрування зводиться до відновлення початкового заповнення регістра.

При побудові криптосхем застосовуються різні комбінації регістрів зсуву з лінійними зворотними зв'язками. Найбільш часто зустрічаються вузли, звані комбінуючими генераторами і (нелінійними) фільтр-генераторами.

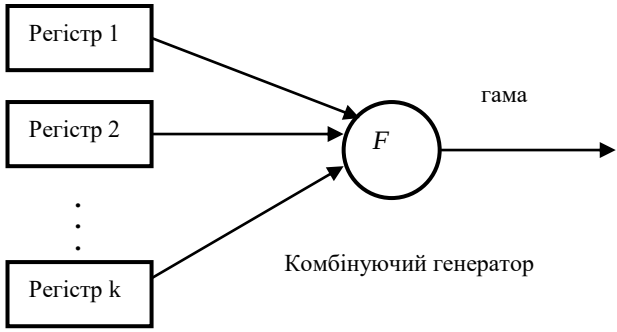
У комбінуючих генераторів в кожному такті роботи чергові елементи вихідних послідовностей декількох регістрів зсуву поступають на вхід деякої функції. Значення цієї функції є виходом генератора (елементом гамми).

Нелінійні фільтр-генератори генерують вихідну послідовність як нелінійну функцію від станів одного і того ж регістра, див. рис. 9.22.

При об'єднанні в криптосхему окремих вузлів, або їх частин, вони можуть впливати один на одного, змінюючи заповнення деяких регістрів, а також управляючи їх рухом. Звичайно регістри зсуву змінюють свій стан регулярно, просуваючись по орбіті на один крок протягом такту роботи

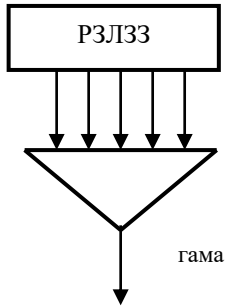
генератора, див. рис. 9.22 а. Якщо ж рух регістра протягом такту роботи генератора залежить від стану схеми, то такий рух називається керованим. нерівномірний рух регістрів, як правило, істотно ускладнює вихідну послідовність.

Окрім регістрів зсуву з лінійними зворотними зв'язками в криптографії використовуються регістри зсуву з нелінійними функціями зворотного зв'язку, у тому числі не обов'язково з двійковими елементами. В самому загальному випадку функція зворотного зв'язку задається табличний.



Комбінуючий генератор

а



Фільтр - генератор

б

а - комбінуючий генератор; б – фільтр генератор

Рис. 9.22. Схеми комбінуючого генератора та фільтра генератора

Реалізація шифрів на ґрунті регістрів зсуву з нелінійними зворотними зв‘язками має певні проблеми, зокрема:

– у вихідних послідовностях можуть бути зміщення кількості одиниць відносно кількості нулів;

– найбільший період послідовностей може виявитися меншим ніж очікувалось;

– періоди послідовностей для різних початкових значень можуть відрізнятись;

– послідовність деякий час може виглядати випадковою, а потім зациклюватись на одному, або декількох значеннях.

Перевагою використання цих шифрів у криптографії виявляється те, що не існує теорії аналізу таких регістрів і тому існує дуже обмежена кількість можливостей криптоаналізу шифрів побудованих за допомогою цих схем.

Необхідно враховувати, що теорія регістрів зсуву з нелінійними функціями зворотного зв'язку розроблена недостатньо. При обґрунтовуванні вибору конкретного типу нелінійного зв'язку можуть виникнути істотні труднощі.

9.5. Цифрові маскувальники

Пристрої захисту інформації з обмеженим доступом використовують два типа перетворень – блочне та поточне. У цифрових маскувальниках використовується поточне перетворення відкритого тексту. Операція маскування полягає у порозрядному складанні по модулю два двійкових символів інформаційного потоку із двійковими символами маскуючої послідовності, яка породжується ключем, див. рис. 9.23.

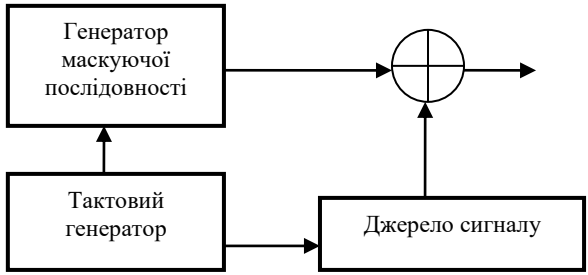


Рис. 9.23. Схема цифрового маскуючого генератора

Символи відкритого тексту формуються джерелом сигналу – (ДС), у якості якого можуть бути – мовна інформація, зображення або дані. Маскуюча послідовність створюється генератором (ГМП). До схеми також включено сумматор по модулю два ($\oplus$), та тактовий генератор.

Цифрові маскувальники відрізняються способом побудови генератора маскуючої послідовності. Найбільш поширеним є генератор m - послідовності або послідовності максимального періоду, яка формується на базі регістру зсуву із лінійними зворотними зв'язками. При кількості гнізд регістру зсуву n - максимальний період послідовності дорівнює $N_p = 2^n - 1$ т.ч. значення маскуючої послідовності у i - му тактовому відлуку та у момент $(i + N_p)$ - однакові. У цьому випадку перетворення, яке здійснює маскування відкритого тексту, є періодичним.

Генератори псевдовипадкових послідовностей ми розглянули у попередньому розділі. Генератор m - послідовності визначається структурою його зворотних зв'язків.

Розглянемо формальні правила побудови генератора m - послідовності:

1. Визначаються коефіцієнти поліномів для заданого n.
2. Будують зображення n розрядів регістру та виконують нумерацію прямокутників (розряди регістру) у лівому напрямку.
3. Перший (молодший) розряд з'єднують із входом каналу та одночасно від нього будують зворотній зв'язок.
4. Від першої одиниці двійкового представлення полінному справа виконується відлік і, у разі наявності першої одиниці, з'єднують із сумматором по модулю два. Другій вхід суматора з'єднується із молодшим розрядом і т.д.
5. Вихід остатнього суматора по модулю два підключається до старшого розряду.

Так, наприклад, для $n = 9$ один з поліномів має такий вигляд : [26] - $F(X) = X^9 + X^4 + 1$, або у двійковій формі – 1000010001. Відповідний генератор m - послідовності показано на рис. 9.24.

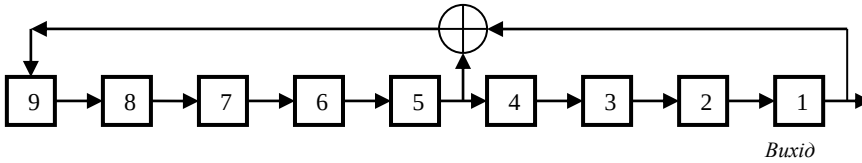


Рис. 9.24. Схема генератора m - послідовності для $n = 9$

На практиці при проектуванні цифрових систем маскування інформації слід орієнтуватися на максимальну кількість зворотних зв'язків у генераторі. Це пов'язано з тією обставиною, що у випадку, коли на боці прийому існує канальна помилка, вона буде розмножуватися у генераторі m - послідовності пропорційно кількості цих зв'язків. Це, у свою чергу, веде к втраті стійкості до перешкод.

Більш ефективним є нелінійний генератор маскуючої послідовності. Розглянемо принцип роботи маскувальника, у якому використовується нелінійний генератор на базі мажоритарної функції, див. рис. 9.25.

На схемі двійкові символи ключа $K=(K_1,K_2,...,K_z)$ записуються у ключовий регістр, а потім у паралельній формі поступають на z суматорів по модулю два, де z - бітова довжина ключа. На другі входи суматорів поступають двійкові символи з z - розрядного регістру зсуву. У $(i+1)$ тактовий момент роботи алгоритму зміст регістру зсуву виглядає так:

$$Y_{i+1}=(y_{i-z},y_{i-z+1},...,y_{i-1},y_i), \tag{9.8}$$

де $y_j\ (j=\overline{i-z,i})$ - двійкові символи маскованого тексту, який поступає у регістр зсуву по ланцюгу зворотного зв'язку.

Таким чином, з виходу j-го суматора отримуємо символ ключа z_j (якщо $y_j=0$), або його інверсію $\overline{z_j}$ (якщо $y_j=1$). На вхід мажоритарного елементу ($\geq M$) у кожен тактовий момент поступає паралельний набор з z двійкових символів, значення яких визначається обраними символами ключа та значеннями z попередніх символів маскуємого тексту.

На виході мажоритарного елементу формується ключовий потік

$$K^1=(..K^1_{i-1},K^1_i,K^1_{i+1}...), \tag{9.9}$$

де $$K^1_{i+1}=M_{aj}(y_{i-z}\oplus K_1,y_{i-z+1}\oplus K_2,...,y_i\oplus K_m). \tag{9.10}$$

Відкритий текст $X=(..X_{i-1},X_i,X_{i+1}...)$ поступає на вхід суматора по модулю два, на другий вхід якого подаються символи ключового потоку (K^1). На виході суматору формується маскований текст Y, який поступає у канал зв'язку та по зворотному ланцюгу у регістр зсуву. Таким чином, у $(i+1)$ тактовий момент роботи алгоритму символ y_{i+1} маскованого тексту Y дорівнює

$$y_{i+1} = x_{i+1} \oplus K_{i+1}^1 = x_{i+1} \oplus M_{aj}(y_{i-z} \oplus K_1, y_{i-z+1} \oplus K_2, ..., y_i \oplus K_m). \quad (9.11)$$

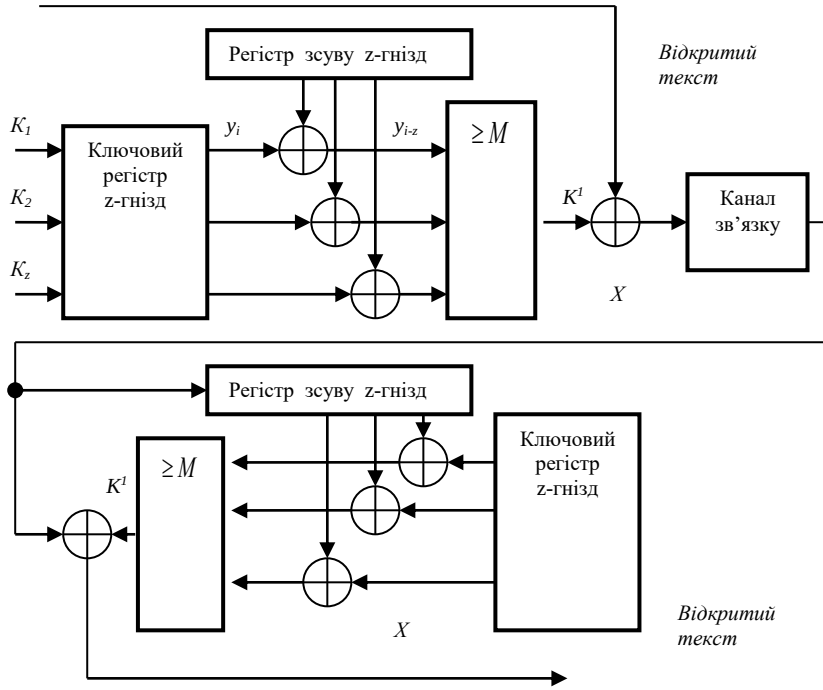


Рис. 9.25. Схема маскувальника із нелінійним генератором

На боці прийому здійснюється демаскування шляхом знаходження суми по модулю два символів ключового потоку, аналогічного тому який використовується на боці передачі, та маскованого тексту Y, який поступає з каналу зв'язку. Як бачимо, у відповідності до виразів (9.10-9.11), отримаємо наступну залежність $y_{i+1} \oplus K_{i+1}^1 = x_{i+1}$. Таким чином, на виході отримуємо початковий текст.

Для спрощення практичної реалізації маскувальників, побудованих за принциповою схемою, наведеною на рис. 9.25, можна застосовувати каскадування [30, 38] (див. рис. 9.26). У цьому випадку, мажоритарний елемент реалізує логічну операцію абсолютної більшості над трьома двійковими змінними - X_1, X_2, X_3. Символи ключа у схемі розподілені на три групи (три символи у кожній). Кожна трійка символів додається по

модулю два до символів регістру зсуву (який заповнюється по зворотному зв'язку).

Кожна трійка замикається на свій мажоритарний елемент - ($\geq M$). Виходи першого каскаду (дев'ять) подають на входи другого каскаду (три). Виходи другого каскаду поступають на третій каскад, на виході якого і формується маскуючи послідовність.

Після виконання операції сума по модулю два із символами відкритого тексту отримуємо замаскований текст.

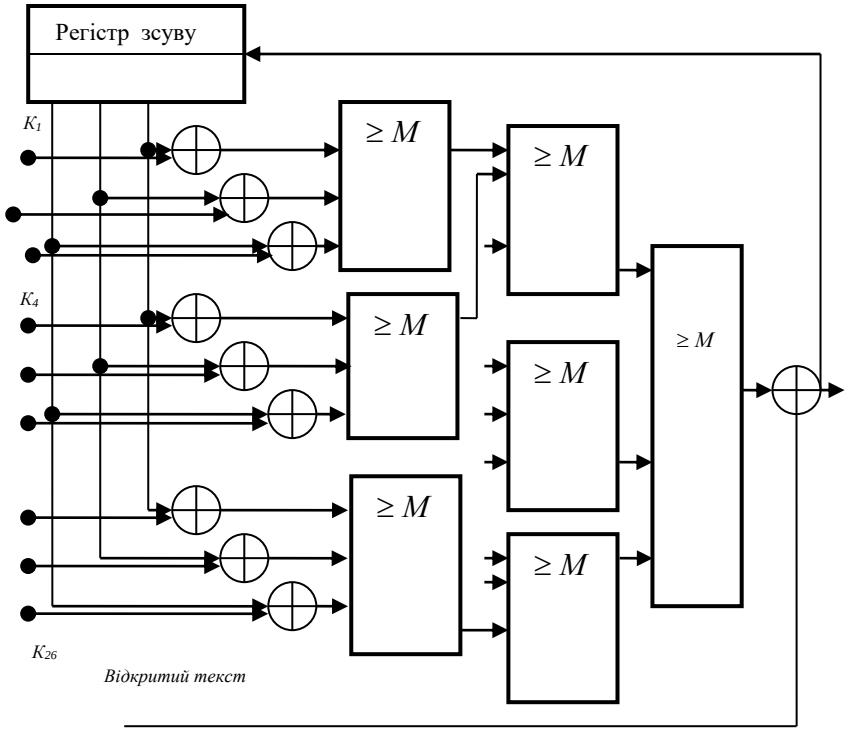


Рис. 9.26. Схема маскувальника із каскадуванням

Таким чином, маскувальники на базі нелінійного генератора маскуючої послідовності дозволяють збільшити кількість ключів у порівнянні із маскувальником на базі m - послідовності. Але описані у цьому розділі цифрові маскувальники мають певні недоліки, зокрема при кожному сеансі маскуюча послідовність починається з тих самих символів незалежно від способів формування ГМП. Таким чином, при

професійному перехопленні криптоаналітик може використати накладення маскованих потоків різних сеансів із урахуванням статистики переданих даних.

Запитання для самоперевірки

1. Які логічні елементи використовуються у цифрових пристроях комплексів систем захисту інформації з обмеженим доступом?
2. Які завдання виконують дешифратори і шифратори у цифрових системах захисту інформації з обмеженим доступом?
3. З яких етапів складається синтез шифратора?
4. З яких етапів складається синтез дешифратора?
5. Яке завдання виконують маскувальники інформації з обмеженим доступом у цифрових пристроях комплексів систем захисту?
6. Які недоліки мають цифрові маскувальники інформації з обмеженим доступом?
7. Які функції у системах скремблер-дескремблер відіграють генератори псевдовипадкової послідовності?
8. Який пристрій має назву рекурсивного цифрового фільтра?
9. Назвіть основні етапи побудови цифрового фільтра.
10. Які логічні елементи використовуються у цифрових фільтрах?
11. Зберіть комбінаційні схеми шифраторів та дешифраторів, які наведені на рис. 9.14-9.17, у EBW або Multisim та перевірте її працездатність.
12. У чому полягає принцип каскадування при проектуванні цифрових компонентів систем захисту інформації з обмеженим доступом?

Література для самостійної підготовки за темою:

4, 19, 36, 45.

Розділ 10. ВИКОРИСТАННЯ КЕРОВАНИХ ПЕРЕСТАНОВОЧНИХ ОПЕРАЦІЙ У ЦИФРОВИХ СИСТЕМАХ КРИПТОГРАФІЧНОГО ЗАХИСТУ

10.1. Поняття про криптографічний захист інформації з обмеженим доступом

Сучасні технології інформаційних систем та мереж на підприємствах та в державних установах дають розвиток для широкого діапазону нових інформаційних сервісів та служб. У недалекому майбутньому ці сервіси та служби будуть інтегровані в багатофункціональні інформаційні мережі. Захист інформації від несанкціонованого доступу - одне з головних завдань із забезпечення конфіденційності, цілісності та автентичності даних, що обробляються й передаються за допомогою цифрових пристроїв.

Головним та важливим засобом для забезпечення захисту вищенаведених служб та додатків є криптографічні алгоритми. Сучасний термін «криптографія» означає сукупність математичних та логічних засобів для забезпечення захисту інформаційних систем та мереж. Криптографічна технологія призначена для захисту логічного рівня фізичних каналів інформаційних мереж.

Передача інформації в інформаційних мережах зумовлює такі головні види загроз безпеки мережевої взаємодії: перехоплення даних, які передають мережею, з метою викрадення, модифікування чи переадресування, несанкціоноване відсилання даних від імені іншого користувача, заперечення користувачами автентичності даних і фактів відсилання-отримання інформації. Одним із можливих способів усунення загроз інформаційної безпеки є використання криптографічних перетворень [39, 40, 48].

Реалізація повного та комплексного захисту інформації з обмеженим доступом в інформаційних системах та мережах повинна відповідати трьом криптографічним вимогам: конфіденційність, автентичність та цілісність даних.

Конфіденційність - це властивість інформації бути відомою лише допущеним та тим суб`єктам, хто пройшов перевірку. Для інших суб`єктів інформація є закритою.

Автентичність - процес підтвердження ідентичності. Система чи об`єкт повинні знати, що отримана інформація надійшла від дійсного джерела повідомлення.

Цілісність даних - властивість даних не змінюватися при функціонуванні системи.

Для задоволення цих трьох вимог у сучасних інформаційних системах та мережах існують три основні криптографічні системи захисту інформації з обмеженим доступом - шифрування з закритими ключами, шифрування з відкритими ключами та хеш-функції.

Шифри забезпечують конфіденційність шляхом перетворення даних у повідомлення, яке важко зрозуміти. На вхід шифру надходить відкритий текст (плантекст), а на виході отримується шифротекст. У шифрі відкритий текст перетворюється в шифротекст із застосуванням ключа шифрування. В алгоритмі дешифрування шифротекс перетворюється на відкритий текст зворотним чином. Сукупність процесів шифрування та дешифрування має назву шифри. В ідеалі без наявності ключа відсутня можливість перетворити шифротекст на відкритий текст. На практиці потужні криптографічні алгоритми значно ускладнюють відкриття відкритого тексту з шифротексту за відсутності ключа. Тому спроби зламу шифру за відсутності ключа, шляхом перебору, займають дуже багато часу, навіть із використанням потужних інформаційних систем.

Для прикладу нехай довжина ключа шифрування дорівнює 128 бітів. Тому для зламу шифру необхідно виконати 2^{128} операцій. Якщо обчислювальна машина виконує біля 2^{49} операцій за секунду, то процес зламу займе 2^{79} секунд. В році 2^{25} секунд, таким чином необхідно 2^{54} роки для зламу шифру, що має ключ шифрування 2^{128} біти. Але вік всесвіту становить лише 2^{34} роки.

Криптографічні перетворення класіфікують різними способами, але найчастіше їх розподіляють в залежності від способу використання та за типом ключа:

- безключові - не використовуються ключі (хеш-функції, генерація псевдовипадкових чисел, односторонні перестановки);

- перетворення з таємним ключем - використовується ключовий параметр - секретний ключ (симетричне шифрування, цифровий підпис, хеш-функції, ідентифікація);

- перетворення з відкритим ключем - використовують у своїх обчисленнях два ключі - відкритий та закритий (асиметричне шифрування, цифровий підпис).

За наявності ключа шифротекст перетворюється системою шифрування на відкритий текст. Тому доступ до ключа шифрування повинен бути обмежений.

Відзначимо деякі основні характеристики шифрів [4, 39, 40, 48]:

• стійкість шифру - здатність шифру протистояти будь-яким несанкціонованим атакам на нього. Це поняття - центральне для криптографії. Якщо наявність криптограм у супротивника не зменшує невизначеності можливого вибору відкритого тексту, що відповідає криптограмі, то такий шифр називають теоретично стійким;

• ефективність шифру, яка визначається швидкістю шифрувальних і дешифрувальних відображень;

• довжина ключа, точніше, об'єм ключового простору (безліч можливих значень ключів);

• складність виконання операцій шифрування і дешифрування (повинні бути простими по можливості);

• збільшення кількості помилок (для деяких шифрів помилка в одній букві при шифровці викликає значну кількість помилок у дешифрованому тексті). При виборі шифру для зв'язку прагнуть мінімізувати цю властивість шифру;

• перешкодостійкість шифру - здатність шифру при дешифруванні криптограми протидіяти утворенню помилок, які виникли під час шифровки через перешкоди на лініях зв'язку;

• імітаційна стійкість шифру - здатність шифру протидіяти спробам супротивника нав'язати абоненту неправдиву інформацію шляхом спотворення криптограми на каналах зв'язку.

Симетричні шифри, у свою чергу, поділяються на блочні та поточні шифри. Ці шифри використовують однаковий ключ для шифрування та дешифрування (рис. 10.1).

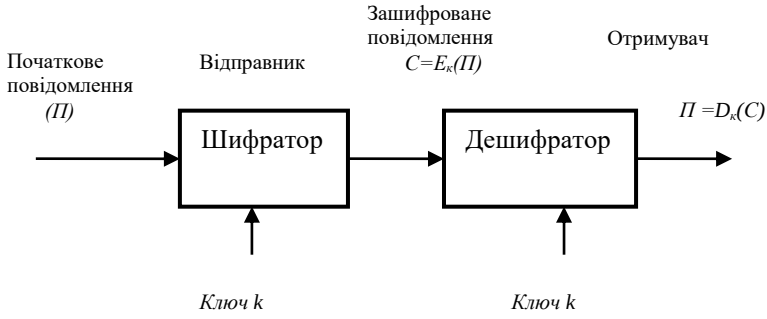


Рис. 10.1. Шифрування-дешифрування з закритим ключем

За наявності ключа шифротекст перетворюється системою шифрування у відкритий текст. Тому доступ до ключа шифрування повинен бути обмежений.

Процес шифрування описується виразом:

$$C = E_k(П),$$

де $П$ - відкритий текст (повідомлення);
k – ключ шифрування;
C – шифротекст.

Процес дешифрування описується виразом:

$$П = D_k(C),$$

де D – декодований текст.

У системах шифрування з відкритим ключем (асиметричні шифри) на відміну від симетричних шифрів використовуються два ключі – відкритий (публичний) та закритий (секретний) (рис. 10.2). Ці ключі математично пов`язані між собою. Шифротекст отримується з відкритого тексту відкритим ключем шифрування k_1. Відкритий текст отримується з шифротексту закритим ключем дешифрування k_2. Ця система визначається трьома алгоритмами: генерація ключів, шифрування та розшифрування. Алгоритм генерації відкритий.

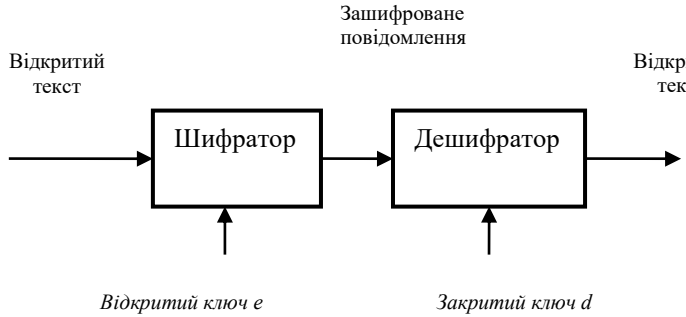


Рис. 10.2. Система шифрування-дешифрування з відкритим ключем

Асиметрична система шифрування на відміну від симетричної забезпечує не лише конфіденційність, а ще й може використовуватися для

виконання автентифікації. Так віддалений вузол надсилає серверу зашифроване закритим ключем повідомлення. Після того як сервер отримає шифротекст, він дешифрує його відкритим ключем. Якщо дешифрування пройшло правильно, то автентичність вузла коректна.

У сучасних цифрових системах конфіденційність інформації забезпечується симетричним (алгоритми ГОСТ 28147-89, DES, 3DES, AES, IDEA, Blowfish) та асиметричним (алгоритми RSA, El Gamal) шифруванням.

Як відомо, стійкі блокові шифри дозволяють забезпечити обчислювально стійкі схеми аутентифікації. Прикладом є блочно-симетричні шифри DES, AES, ГОСТ 28147-89.

Блокові шифри бувають двох основних видів: шифр перестановки (transposition, permutation, *P*-блоки) і шифр заміни (підстановки, substitution, *S*-блоки).

Блок підстановки (*S*-блок), див. рис. 10.3, складається з дешифратора, що перетворює *n*-розрядний двійковий сигнал в однорозрядних сигнал з базисом 2^n, системи комутаторів внутрішніх з'єднань (всього з'єднань $2^n!$) і шифратора, який переводить сигнал з однорозрядного 2^n-ричного в *n*-розрядний двійковий. На практиці блок підстановок використовується як частина більш складних систем.

У загальному випадку *S*-блок може мати різну кількість входів/виходів, в цьому випадку в системі комутації від кожного виходу дешифратора може йти не одне з'єднання, а 2 чи більше або не йти зовсім. Те ж саме справедливо і для входів шифратора.

Блок перестановок (*P*-блок) змінює положення цифр і є лінійним пристроєм. Цей блок може мати дуже велику кількість входів-виходів. Криптоаналіз ключа для *n*-розрядного *P*-блоку проводиться шляхом подачі на вхід *n-1* різних повідомлень, кожне з яких складається з *n-1* нуля («0») і 1 одиниці («1»), див. рис. 10.4.

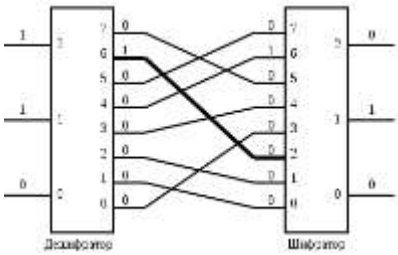


Рис. 10.3. Принципова схема 3-розрядного S - блоку

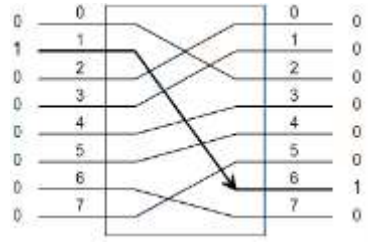


Рис. 10.4. Принципова схема 8-розрядного P - блоку

Окремим випадком *P*-блоку є циклічний зсув. Наприклад, у разі зсуву на 1 біт, крайній біт відщеплюється і переміщається на інший кінець регістру або шини. Зсув на більшу кількість біт можна розглядати, як багаторазове використовування зсуву на 1.

DES (англ. Data Encryption Standard) - це симетричний алгоритм шифрування даних, стандарт шифрування прийнятий урядом США із 1976 до кінця 1990-х, з часом набув міжнародного застосування. Зараз DES вважається ненадійним в основному через малу довжину ключа (56 біт) та розмір блоку (64 біти). У 1999 ключ DES було публічно дешифровано за 22 години 15 хвилин. Вважається, що алгоритм достатньо надійний для застосування у модифікації 3-DES, хоча існують розроблені теоретичні атаки. DES поступово витісняється алгоритмом AES, що з 2002 року є стандартом США.

Advanced Encryption Standard (AES), також відомий під назвою Rijndael - симетричний алгоритм блочного шифрування (розмір блока 128 біт, ключ 128/192/256 біт), фіналіст конкурсу AES і прийнятий в якості американського стандарту шифрування урядом США. Станом на 2010 рік AES є одним із найпоширеніших алгоритмів симетричного шифрування.

Серед поширених криптографічних алгоритмів блочного типу слід назвати алгоритм ГОСТ 28147-89 «Системи обробки інформації. Захист криптографічний. Алгоритм криптографічного перетворення».

Алгоритм ГОСТ 28147-89 використовує ключ розміром 256 біт, завдяки чому він має досить високу крипостійкість. Для його розкриття потрібно використати 2^{256} переборів, не враховуючи S - блоків. Він також є досить стійким до диференціального та лінійного розкриття.

У середині 90-х років минулого століття відбулися значні зміни в криптографії та мікроелектрониці. У криптографії були розроблені ефективні методи криптаналізу [39, 48, 49]: диференційний, лінійний, та інші. Ці методи дозволили отримувати відкритий текст з шифротексту без знання ключа шифрування. В мікроелектрониці збільшилися можливості електронних пристроїв. Швидкодія та об`єм пам`яті збільшилися на декілька порядків. Це призвело до пропорційного збільшення можливостей екстенсивних методів криптаналізу, таких як повний перебір можливих даних.

В результаті попередніх досліджень встановлено, що булеві (логічні) функції є ефективним інструментом аналізу криптографічних перетворень. При цьому відповідно до апарату булевих функцій, кожне відображення, зокрема і підстановку $GF(2)^n \to GF(2)^m$, можна подати у вигляді векторної булевої функції, компонентами якої є звичайні булеві функції з областю визначення $GF(2)^n$ і областю значення $GF(2):$

$$Y = F(X) = \{y_1, y_2, ..., y_m\} = \{f_1(X), f_2(X), ..., f_m(X)\},$$

де $\forall i \in \{1,..,m\}$, $y_i = f_i(X)$, $X \in CF(2)^n$ та $y \in CF(2)$.

На процес синтезу алгоритмів блочного шифрування суттєво впливають параметри швидкості роботи алгоритму та складність його реалізації. На розв'язання однакових задач можна витратити більше часу, використовуючи меншу кількість ресурсів, та навпаки, збільшивши ресурси - можна зменшити час розв'язання задачі.

Моделювання обчислювальних процесів виконується у вигляді комбінаційних схем, для реалізації яких використовується базис логічних елементів (вентилів), що реалізують логічні операції. Клод Шеннон [40] показав важливість булевої алгебри для аналізу та синтезу перемикальних схем.

У якості складності схемотехнічної реалізації комбінаційної схеми $C_\Omega(F)$ розумітимемо кількість вентилів, які реалізують схему F.

Комбінаційна складність суттєво залежить від базису Ω. В навчальний посібнику ми дамо оцінку комбінаційних схем для базису $\Omega = \{\wedge, \vee, \neg\}$.

До кількісних оцінок схемотехнічної реалізації варто підходити обережно, оскільки для отримання точних оцінок необхідні спеціальні дослідження із урахуванням елементної бази цифрових пристроїв систем криптографічного захисту інформацій, що постійно розвивається.

Для оцінки швидкісних параметрів схеми F використовують час, який було витрачено на виконання найповільнішої операції, тобто - такт затримки $\tau_{зат}$. Наприклад, $\tau_{\Omega зат} = t_\vee$ - для базису $\Omega = \{\wedge, \vee, \neg\}$ та $\tau_{\Omega зат} = t_\otimes$ - для $\Omega = \{\otimes, \vee, 1\}$. У наступному розділі навчальний посібника ми детально зупинимося на прикладі проектування комбінаційних схем цифрових пристроїв криптографічного захисту з використанням блоку керованих перестановок, як найбільш простого з точки зору схемотехнічної реалізації.

В даний час при розробці криптографічних примітивів наукові спільноти провідних світових країн дотримуються підходу, який передбачає відкрите дослідження та обговорення розглянутих на конкурсній основі схем (алгоритмів) криптоперетворень. В останні кілька років у відкритій пресі [44-48, 53,54] багато уваги стали приділяти питанням практичної реалізації схем потокового шифрування. Актуальність застосування методів поточного шифрування в комерційних

додатках і відкритих системах обумовлена, в першу чергу, істотним зростанням обсягів трафіку передачі даних в мережах зв'язку.

Наприклад, розглянемо схему SNOW [53] (синхронна схема, заснована на ідеї класичного підсумовуючого генератора. Належить до класу схем з рівномірним рухом регістру). Структурна схема шифру наведена на рис. 10.5. Вона складається з лінійного рекурентного регістра довжиною 16 над *GF* (2^{32}). Перші 32 біта лінійного рекурентного регістра (ЛРР) задають стани кінцевого автомата (КА). У свою чергу КА складається (рис. 10.6) з блоку підстановки *S* і двох 32-розрядних регістрів *R1, R2*.

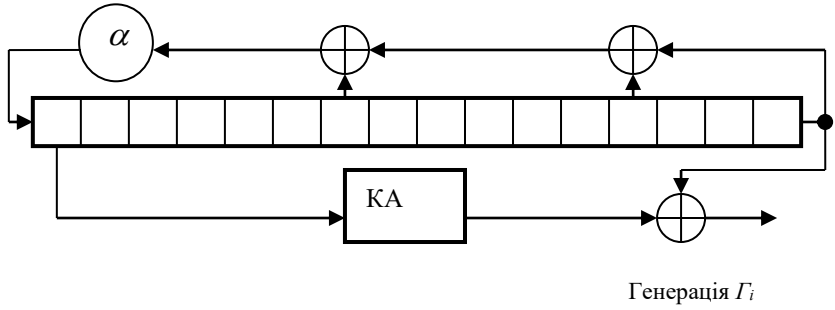


Рис. 10.5. Схема SNOW

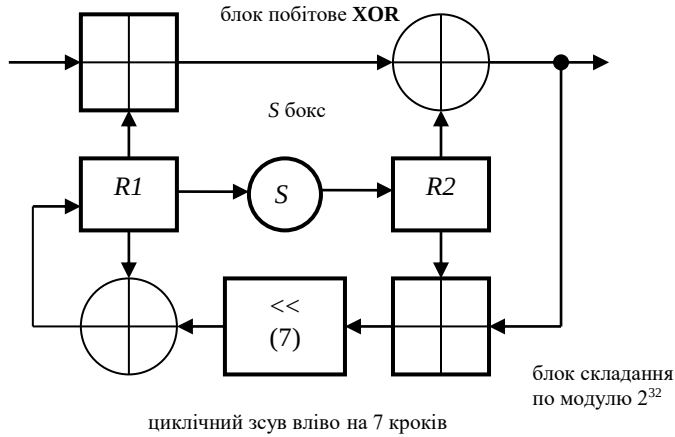


Рис. 10.6. Схема кінцевого автомату

На виході SNOW формується гамма шифру Γ_i, яка в явному вигляді використовується для потокового шифрування. Процес генерації Γ_i здійснюється наступним чином. Спочатку здійснюється установка (ініціалізація) ключа. В процесі установки ключа проводиться початкове заповнення зсувного регістра і регістрів *R1, R2* кінцевого автомата. Потім обчислюються перші 32 біта гами шифрувальної Γ_i шляхом побітового складання виходу КА та вмісту останньої клітинки зсувного регістру.

Далі здійснюється зсув вмісту регістра, і обчислюються наступні 32 біта Γ_i побітовим складанням виходу КА та вмісту останньої клітинки зсувного регістра, і т.д.

Лінійний рекурентний регістр (ЛРР) будується з використанням полінома $p(x) = x^{16} + x^{13} + x^7 + \alpha$, який задає зворотній зв'язок із полем $GF(2^{32})$.

Елементи поля $GF(2^{32})$ формуються за рахунок використовування поліному $\pi(x) = x^{32} + x^{29} + x^{20} + x^{15} + x^{10} + x + 1$, де $\pi(\alpha) = 0$. Комірки зсувного регістру (блоки) - $s(1), s(2),..., s(16) \in GF(2^{32})$.

Після зсуву регустра, значення комірки $s(1)$ поступає на вхід *КА*. Вихід з *КА* визначається наступним виразом:

$$KA[i] = (s(1) \boxplus R1[i]) \oplus R2[i].$$

Для формування ключа Γ_i вихід *КА* складається по *mod2* із $s(16)$, тобто

$$\Gamma_i = KA[i] \oplus s(16).$$

10.2. Проектування цифрових пристроїв криптографічного захист інформації із використанням блоку керованих перестановок

Ряд перспективних досліджень формування розсіювання запропоновано та створено на основі методу керованих перестановок [39].

Сучасні процесори інформаційних систем орієнтовані на роботу з байтами, їхні інструкції обмежено підтримують роботу з даними меншого розміру. На сучасних процесорах інформаційних систем бітові перестановки можуть бути реалізовані з використанням методу логічних операцій чи методу табличних підстановок [39,40].

У методі логічних операцій, кожний біт вилучається інструкцією логічного множення (інструкція AND – **«І»**), зсувається на свою нову позицію (інструкція SHIFT), а потім конкатенується з попередньо переставленими бітами (інструкція OR- **«АБО»**).

Розглянемо наступний приклад. Потрібно виконати формальний варіант синтезу блоку керованих перестановок який показано на рис. 10.7.

Схема працює таким чином. Двійковий дешифратор D_m, на вхід якого поступає m - розрядний керуючий вектор $V=(v_1,v_2,...,v_m)$, формує двійковий вектор $U=D_m(V)$ довжини 2^m із вагою Хемінга «1», тобто 1 виробляється тільки в одному розряді:

$$u_s=\begin{cases}1, & якщо \quad s=1+v_1+v_2\cdot 2+...+v_m\cdot 2^{m-1}\\ 0, & в \quad інших \quad випадках.\end{cases}$$

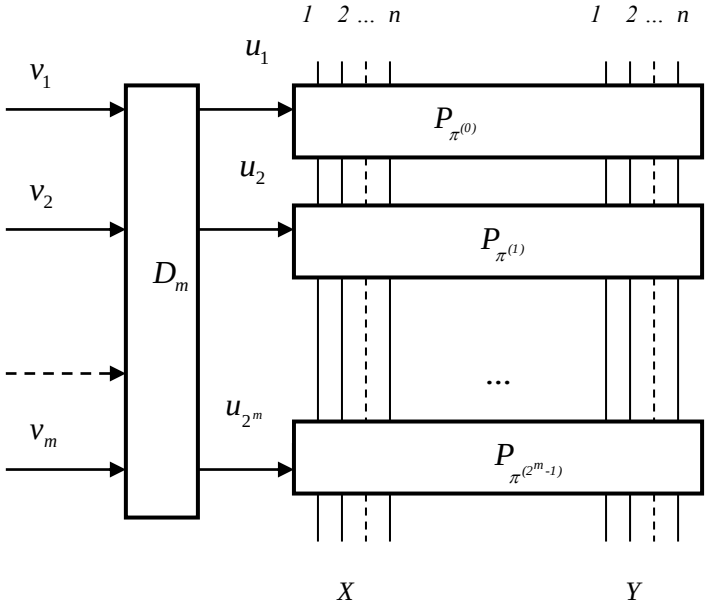


Рис. 10.7. Формальна схема синтезу блоку керованих перестановок

Комутація розрядів входу та виходу за заданою фіксованою перестановкою $\pi^{(s-1)}$ реалізується відповідно до схеми, яка наведена на рис. 10.8.

У схемі розряди вектора *X* поступають на вхід *n* логічних елементів «**І**» одночасно із керуючим сигналом u_s. Комутація здійснюється тільки в тому випадку, якщо $u_s = 1.$

Загалом, багато положень прикладної теорії цифрових автоматів, зокрема кінцевих автоматів з псевдовипадковими переходами з одного стану в інший, широко використовуються в елементах комплексів систем захисту інформації з обмеженим доступом. Наприклад, такі кінцеві автомати є генераторами шифру заміни (підстановки) для символів вхідного алфавіту. Підстановки, що формуються таким автоматом, дозволяють для одного й того ж символу генерувати множину підстановок різної довжини (генеруються префіксні коди). Вибір підстановок для того ж самого символа вхідного алфавіту здійснюється псевдовипадковим способом. Різноманітність алгоритмів функціонування кінцевих автоматів із псевдовипадковими переходами, які є стійкими до віртуальних послідовностей, і використання запропонованих кодів значно ускладнює процес розкриття шифротексту.

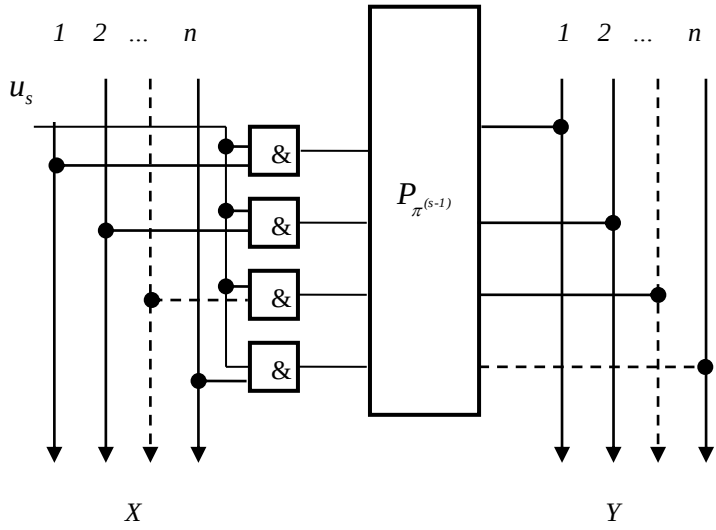


Рис. 10.8. Загальний вигляд реалізації схеми керованого вибору фіксованої модифікації $\pi^{(s-1)}$

Наприклад, у блоці перестановок $\pi_{4/4}$ вектору $V=(1,1,1,0)$ відповідає модифікація

$$\pi^{(7)}=\begin{pmatrix}1 & 2 & 3 & 4\\ 4 & 1 & 3 & 2\end{pmatrix}.$$

Реалізація модифікації $\pi^{(7)}$ показана на рис. 10.9.

Схема, представлена на рис. 10.8, має назву **матричної структури.** Аналізуючи швидкісні параметри такої реалізації, бачимо, що $t(P_{n/m})=t(D_m)+t_\vee$, де $t(P_{n/m})$ - час виконання схеми, $t(D_m)$ - час формування керуючого вектора $(t(D_m)\approx log_2 m\cdot t_\vee$, $t_\vee$ - час виконання логічної операції **«І».**

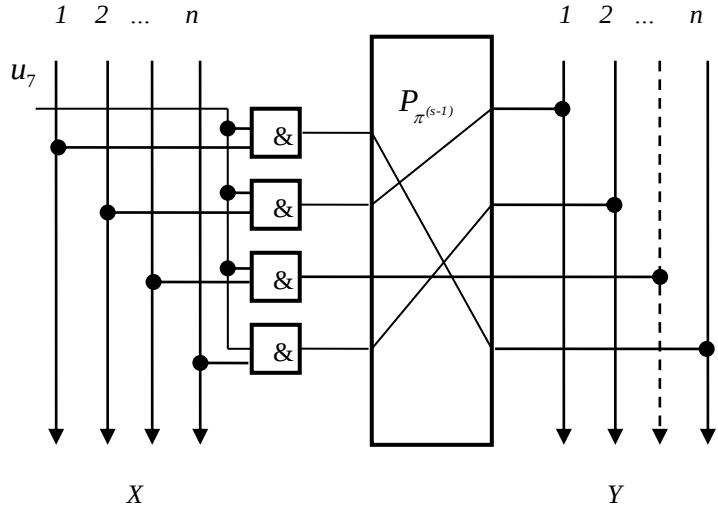


Рис. 10.9. Приклад схеми блоку керованого вибору фіксованої модифікації $\pi_{4/4}$

При синтезі конкретних комбінаційних схем цифрових пристроїв, які реалізують цей алгоритм шифрування, рекомендується наперед забезпечити формування вектора U, час затримки менший $(log_2 m+1)\cdot\tau$, де $\tau = t_{\vee}$. У загальному випадку

$$\tau \le t(P_{n/m}) \le (log_2 m+1)\cdot\tau.$$

Складність реалізації блоку $P_{n/m}$ із матричною структурою

$$C_\Omega(P_{n/m}) = min(n\cdot 2^m, n^2) + C_\Omega(D_m).$$

Формула для $C_\Omega(P_{n/m})$ отримана наступним чином. Значення $n\cdot 2^m$ отримано для кожної з 2^m модифікації $\pi^{(s)}$ (див. рис. 10.9) необхідно *n* вентилів, які реалізують кон'юнкцію - &.

Однак, при $n \ge 2^m$ достатньо застосувати n^2 вентилів $f_{\&}^{(ij)}(u)$, де індекси *i* та *j* позначають конкретні вхід та вихід вентиля. Вентиль $f_{\&}^{(ij)}(u)$ «замикає» ланцюг, якщо виконується умова

$$u = u^{(ij)} = 1,$$

де $u^{(ij)} = \sum_{s=1}^{2^m} u_s \cdot \xi^{(ij)} \cdot (\pi^{(s)});$

$$\xi^{(ij)}(\pi^{(s)}) = \begin{cases} 1, & якщо \quad \pi^{(s)}(i) = j \\ 0, & у\ протилежному\ випадку. \end{cases}$$

Складність реалізації двійкового дешифратора, відповідно до [39,40], знаходиться в межах $2^m + m - 2 \le C_\Omega(P_{n/m}) \le 2^m + (m-2)2^{\lceil m/2 \rceil}$.

Розглянемо приклад реалізації блоку керованих перестановок $P_{2/1}$ в базисі функціональних елементів $\Omega = \{\wedge, \vee, \neg\}$. Принципова схема блоку показана на рис. 10.10. Реалізація блоку показана на рис. 10.11.

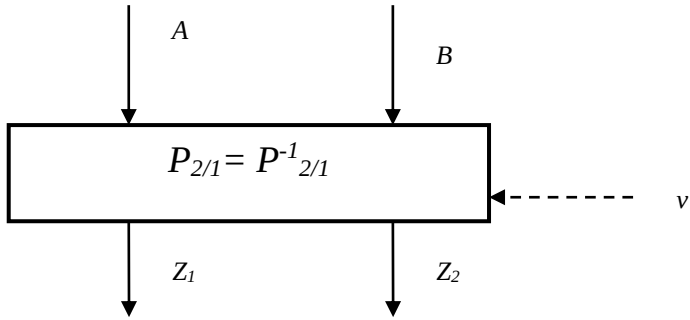


Рис. 10.10. Приклад схеми блоку керованих перестановок $P_{2/1}$

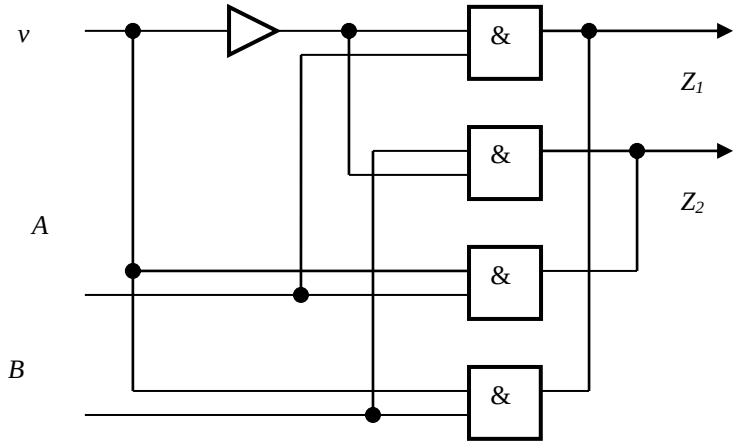


Рис. 10.11. Схема блоку керованих перестановок $P_{2/1}$ в базисі $\Omega = \{\wedge, \vee, \neg\}$

У загальному випадку блок керованих перестановок $P_{n/m}$ задовольняє умові - рівноймовірного зсуву, якщо при рівноймовірному розподілі всіх модифікацій блоку довільно вибраний вхідний біт $x_i, i \in \{1,2,.....,n\}$ в результаті перетворення може виявитися на виході в будь-якому з *n* двійкових розрядів з однаковою вірогідністю.

Стосовно блоку $\pi_{n/m}$ дана умова визначається виразом

$$\Pr_v\{\pi_{n/m}(i) = j\} = \frac{1}{n} \quad \forall i, j \in P\{1,2,...,n\},$$

де Pr_v – вірогідність події $\{\pi_{n/m}(i) = j\}$, яка обчислюється для всіх $V \in \mathbf{V}$.

Блок $P_{n/m}$ називається блоком *h*-го порядку, якщо він дозволяє перемістити будь-які *h* бітів на вході в будь-які *h* розряди на виході [39]. Для блоку *h-го* порядку використовуватимемо позначення $P_{n/m}^{(h)}$.

В термінах блоку $\pi_{n/m}$ дане визначення рівнозначне наступному. Блок $P_{n/m}$ називається блоком *h-го* порядку, якщо для будь-яких відмінних між собою елементів $i_1, i_2 ...., i_h \in \{1,2,....n\}$ існує $\pi_v \in \pi_{n/m}$

$$\pi_v = \begin{pmatrix} i_1 & i_2 & \dots & i_h \dots & \dots & \dots \\ j_1 & j_2 & \dots & j_h & \dots & \dots \end{pmatrix}.$$

З погляду практичної реалізації цифрових комбінаційних схем, інтерес становить умова $h = n$. Як наголошувалося вище, блоки керованих перестановок, що задовольняють даній умові, називаються блоками максимального порядку. Такі блоки реалізують будь-яку перестановку, що рівнозначно виразу $S_n \subseteq \pi_{n/m}$.

Блок має такі властивості:

- є блоком рівномірного зсуву;
- реалізує усі перестановки (наприклад, транспозицію), тобто є блоком максимального порядку;
- усі модифікації унікальні;
- реалізує інволюцію, тобто $P_{2/1}^{-1} = P_{2/1}$.

Розглянемо приклад для блоку $P_{4/m}$. Для *n=4* існують 24 різні перестановки. Тому, наприклад, за допомогою матричної структури блоку $P_{4/4}$ неможливо реалізувати всі перестановки $(2^4 = 16 <!= 24)$, але можна вибрати будь-які 16 унікальних модифікацій. З іншого боку, з

використанням блоку $P_{4/5}$ можна реалізувати всі перестановки (2^5=32>24), проте для виконання властивості «унікальності» можливо забезпечити формування тільки 24-х значень управляючого вектора. Наприклад, для $P_{4/4}$ швидкодія $\tau \le t(P_{4/4}) \le 3\tau$, а складність реалізації матричної структури - $C_\Omega(P_{4/4}) \approx n^2 + 2^m = 32$.

Економічна схема $P_{4/4}$, показана на рис. 10.12, має швидкодію 2τ та $C_\Omega(P_{4/4}) \approx 16$.

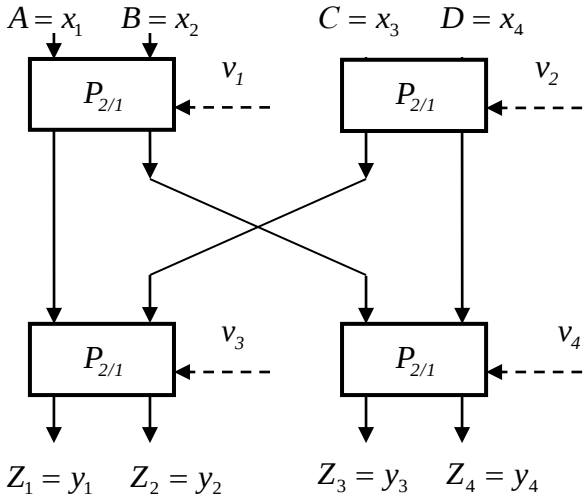


Рис. 10.12. Структура блоку $P_{4/4}$

Перевагою такого методу проектування та реалізації блоків керованих перестановок для криптографічних систем захисту інформації є ефективність та компактність реалізації в апаратному та програмному варіантах.

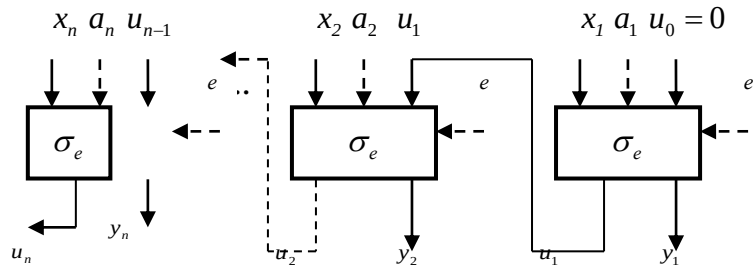
Метод табличних підстановок [39] виконує перестановку біт швидше. Цей метод розподіляє біти на декілька під блоків і виконує перестановку бітів в підблоці шляхом застосування певної таблиці. В такій таблиці якщо стовпці відповідають певному підблоку, то рядки

відповідають певному виду перестановки. Кінцева перестановка знаходиться на перетині певного стовпця з певним рядком. Коли біти всіх підблоків переставлені, вони об'єднуються інструкцією **OR**. Кількість інструкцій **OR** залежить від кількості підблоків.

Так, наприклад, 64-бітна перестановка може бути виконана однією табличною перестановкою, якщо всі результати звести в таблицю розміром 2^{64}X8 байт. Це нереально. Для виконання такої перестановки 64 біти розподіляються на 8 байтів. Кожний байт переставляється окремою табличною перестановкою. Така таблиця має 256 елементів. Для реалізації цього методу необхідно 23 інструкції. Вісім інструкцій **EXTRACT** (для вилучення певного байту), вісім табличних підстановок та сім інструкцій **OR**.

Якщо у якості функцій виходу та перехідних функцій у рекурсивній моделі керованих підстановочних операцій Ω''_{σ,e_f} обрати функції, які не залежать від керуючого вектора V, то можемо отримати цифровий автомат, у якому реалізовано суматор по модулю 2^n (див. рис. 10.13 та 10.14.). Цей суматор здійснює складання операндів X та A та відповідає умові функціональної обертаності:

$$\psi_i^{(e)}(x_i, u_{i-1}, A) = e(a_i \oplus u_{i-1}) \oplus x_i(a_i \oplus u_{i-1}) \oplus a_i \cdot u_{i-1}.$$



e=0 – «складання»; e=1 – «віднімання»

Рис. 10.13. Схема суматора по модулю 2^n

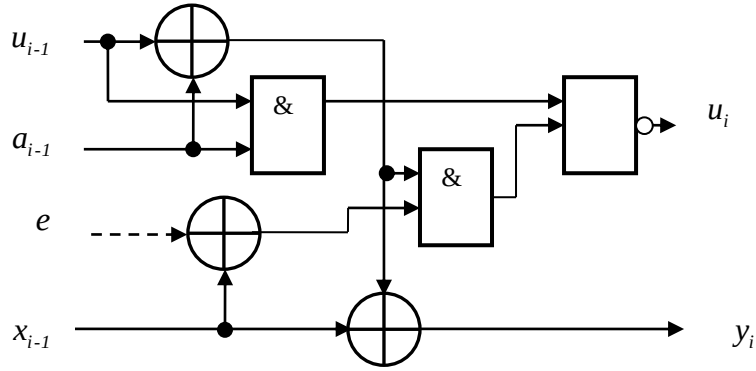


Рис. 10.14. Схема однорозрядного суматора σ_e

Використовуючи схеми 10.9 та 10.10, можна отримати принципову схему елементарного керованого суматора (ЕКС)(див. рис. 10.15.).

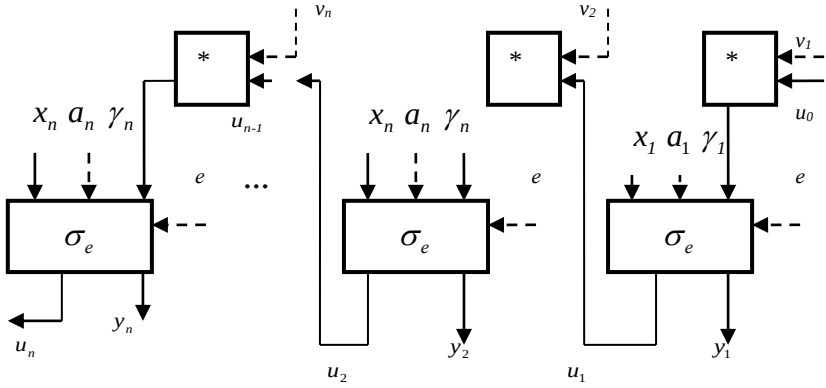


Рис. 10.15. Схема елементарного керованого суматора

Через те що будь-яка булева операція (*) ($u \cdot v$) є відображенням ($GF(2)^2 \to GF(2)$), ці операції поділяються на унарні та бінарні (тобто такі, що залежать від однієї або двох змінних), отже, маємо 4 унарні та 10

бінарних операцій. Таким чином, при реалізації конкретної схеми елементарного керованого суматора у системі криптографічного захисту ми отримаємо близько 20 варіантів ЕКС.

Якщо булева операція (*) є бінарною, то елементарний керований суматор є повним, тобто реалізує 2^n унікальних модифікацій.

Сучасні цифрові автомати орієнтовані на роботу з байтами, їхні інструкції обмежено підтримують роботу з даними меншого розміру. На сучасних процесорах комп`ютерних систем бітові перестановки можуть бути реалізовані з використанням методу логічних операцій чи методу табличних підстановок.

У методі логічних операцій, кожний біт вилучається інструкцією логічного множення (інструкція AND), зсувається на свою нову позицію (інструкція SHIFT), а потім конкатенується з попередньо переставленими бітами (інструкція OR). Метод табличних підстановок виконує перестановку бітів швидше. Цей метод розподіляє біти на декілька підблоків і виконує перестановку бітів у підблоці шляхом застосування певної таблиці. У такій таблиці якщо стовпці відповідають певному підблоку, то рядки відповідають певному виду перестановки. Кінцева перестановка знаходиться на перетині певного стовпця з певним рядком. Коли біти всіх підблоків переставлені, вони об`єднуються інструкцією OR. Кількість інструкцій OR залежить від кількості підблоків.

В таблиці 10.1 наведено кількість інструкцій та об`єм пам`яті, необхідний для реалізацій 64-бітної перестановки описаними вище методами.

Метод табличної перестановки вимагає 16 Кбайт об`єму пам`яті для виконання певного виду перестановки. Кількість інструкцій для реалізації цього методу може бути більше за 23 через помилки кеш-пам`яті процесору.

Таблиця 10.1

Залежність кількості інструкцій та об`єму пам`яті від методу перестановки

Назва методу перестановки	**Кількість інструкцій**	**Об`єм пам`яті**
Таблична перестановка	23	16 Кбайт
Логічні інструкції	256	0

Виконання певних операцій у цифрових автоматах вимагає різної кількості інструкцій. Кількість інструкцій залежить від набору базових інструкцій цифрового автомата. На одному цифровому автоматі, наприклад спеціалізованому процесорі, для виконання певної операції необхідно виконати лише одну інструкцію, тоді як для іншого цифрового автомата необхідно реалізувати деяку сукупність операцій.

У таблиці 10.2. наведено кількість інструкцій для виконання типових для цифрових автоматів операцій.

Кількість тактів, що необхідні для виконання операцій вилучення чи конкатенації, дорівнює кількості інструкцій. Операція циклічного зсуву виконується однією інструкцією, якщо ця інструкція підтримується набором інструкцій цифрового автомата. В протилежному випадку, ця операція виконується інструкціями зсуву *SHIFT* та логічною інструкцією *OR*, кількість яких дорівнює п`яти.

Таблиця 10.2

Кількість інструкцій та тактів для виконання операцій

Операції	**Кількість інструкцій**	**Кількість тактів**
Арифметичні	1	1
Логічні	1	1
Добуток	1 або 2	> 3
Завантаження	1	1…100
Вилучення	2	2
Конкатенація	4	4
Бітові перестановки	> 23	23…1000
Зсуви	1…5	1…5

Більш детально питання, пов’язані із використанням положень алгебри логіки та проектування цифрових систем криптографічного захисту інформації з обмеженим доступом, можна знайти у наступних виданнях [4, 19, 26, 39, 40, 48, 49 та ін.].

Запитання для самоперевірки

1. Яким чином логічні перетворення та функції використовуються у криптографічних системах захисту інформації з обмеженим доступом?
2. Які системи шифрування вам відомі?
3. Які булеві операції використовуються у відомих системах шифрування?
4. У чому полягає суть використання блоків керованих перестановок у блочних шифрах?
5. З яких етапів складається синтез блоку керованих перестановок?
6. Як розрахувати складність реалізації блоку $P_{n/m}$.
7. Яка схема має назву матричної структури?
8. Які логічні операції дозволяють виконувати операцію керованих перестановок?
9. У чому полягають переваги та недоліки метода табличних підстановок?
10. Зберіть схему, наведену на рис. 10.9, у EBW або Multisim та перевірте її працездатність.

Література для самостійної підготовки за темою:

4, 19, 39, 40, 43.

Розділ 11. ПРОГРАМНЕ МОДЕЛЮВАННЯ ЦИФРОВИХ КОМПОНЕНТІВ СИСТЕМ ЗАХИСТУ ІНФОРМАЦІЇ З ОБМЕЖЕНИМ ДОСТУПОМ

Операції, що надежать до процесу розробки цифрових систем, можна поділити на аналітичні й синтетичні. Результатом підпроцесу синтезу є концептуальний проект передбачуваного продукту у формі схеми або креслення, що відображає зв'язки різних компонентів продукту. Аналітичне моделювання відбувається, якщо з проекту видалити несуттєві елементи, редукувати розмірності та ін. На цьому етапі комп'ютери можуть застосовуватися для забезпечення ефективності створення різних концептуальних проектів. Для цього також можна застосовувати різні засоби моделювання, а також програми в системах автоматизованої розробки цифрових систем.

Автоматизоване проектування цифрових систем в останні роки має тенденцію до використання спеціальних мов опису апаратури високого рівня, зокрема для апаратної частини комплексів захисту інформації з обмеженим доступом. При цьому існує певний прогрес у переході від традиційних представлень автоматів за допомогою мов - VHDl, Verilog, Abel та ін. [28, 30] до нових інтегрованих мов, типу System C, що поєднує переваги паралелізму VHDL із семантичними можливостями мови програмування C++, або іншої мови високого рівня. Прогрес у засобах опису цифрових систем пов'язаний зі зростанням ринкового попиту на компілятори та симулятори, що здатні вирішувати завдання введення і верифікації проектів, які містять сотні й тисячі рядків вихідних описів.

У даному розділі навчальний посібника розглядаються приклади програмного моделювання найбільш поширених цифрових компонентів систем захисту інформації з обмеженим доступом, які використовуються при розробці цифрової електронної апаратури.

Основою для моделювання є вибір структур даних, що відображають пам'ять цих пристроїв, і процедури (функції), які реалізовують основні режими експлуатації цих компонентів. Приклади програмної реалізації цифрових компонентів наведені на алгоритмічних мовах Object Pascal та C++ (див. додаток), як найбільш поширених у навчальних програмах молодших курсів вищих навчальних закладів.

11.1. Шифратор двійкового коду

Розглянемо роботу шифратора двійкового коду на наступному прикладі.

Приклад. Розробити процедуру кодування двійкового позиційного восьмирозрядного коду *A* у двійковий трьох розрядний код *B*. Для зберігання двійкового позиційного восьмирозрядного коду використовувати вектор $A=|a_1,a_2,a_3,a_4,a_5,a_6,a_7,a_8|$. Для зберігання двійкового трьохрозрядного коду використовувати вектор $B=|b_1,b_2,b_3|$. Розробити програму для перевірки правильності роботи шифратора. Схема шифратора представлена на рис. 11.1.

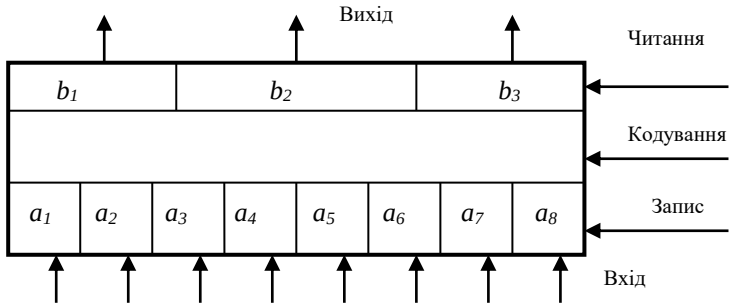


Рис. 11.1. Схема шифратора

Шифратор працює три такти.

Такт перший: на одну з 8 вхідних шин подається потенціал, який означає 1. На решті шин - 0. Імпульс по шині «Запис» формує у регістрі *A* позиційний восьмирозрядний двійковий код.

Такт другий: по шині «Кодування» надходить імпульс, що ініціює роботу алгоритму кодування коду *A* у код *B*.

Такт третій: код *B* з регістра *B* надходить на вихідні шини у формі паралельних імпульсів по команді «Читання».

Для ілюстрації роботи пристрою наведемо декілька прикладів кодування:

- Нехай *A=(10000000)*, тоді для цього коду *B=(000)*;
- Нехай *A=(01000000)*, тоді для цього коду *B=(001)*;
- Нехай *A=(00100000)*, тоді для цього коду *B=(010)*;
- Нехай *A=(00010000)*, тоді для цього коду *B=(011)*;
- Нехай *A=(00001000)*, тоді для цього коду *B=(100)*;
- Нехай *A=(00000100)*, тоді для цього коду *B=(101)*;

- Нехай *A=(00000010)*, тоді для цього коду *B=(110)*;
- Нехай *A=(00000001)*, тоді для цього коду *B=(111).*

На підставі цих варіантів можна зробити кодову таблицю (див. табл. 11.1).

Таблиця 11.1

Кодова таблиця шифратора

Розряди коду *A*		**Розряди коду *B***		
		b_1	b_2	b_3
a_1	10000000			
a_2	01000000			+
a_3	00100000		+	
a_4	00010000		+	+
a_5	00001000	+		
a_6	00000100	+		+
a_7	00000010	+	+	
a_8	00000001	+	+	+

Класичний підхід до розв'язання цієї задачі такий. На базі таблиці 11.1 створюється логічна схема кодування (див. рис. 11.2). У схемі використовуємо три логічних елементи **«АБО»** (саме три розряди у коді *B*).

кодування

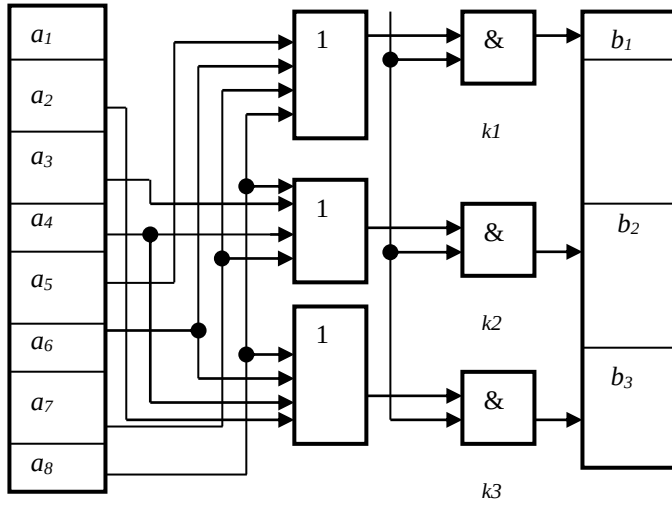


Рис. 11.2. Логічна схема кодування

Аналогом логічного елемента **«АБО»** у мові програмування Pascal є логічна операція **OR**, а елемента **«І»** – операція **AND**. Таким чином, залежно від стану регістра *A* на виходах елементів **«АБО»** формуються потенційні сигнали або *1*, або *0*, які відповідають коду. У регістр *A* код записується імпульсом команди «Кодування», що подається на другий вхід ключів, які реалізовані схемою **«І»**.

На основі рис. 11.2 легко скласти формули для кодування із використанням логічних величин – *k1, k2, k3*.

k1=a5 OR a6 OR a7 OR a8;
k2=a3 OR a4 OR a7 OR a8;
k3=a2 OR a4 OR a6 OR a8.

Лістинг програми *Coder* та вигляд вікна консольного додатку наведено нижче(див. рис.11.3.). Лістинг програми *Coder* на C++ наведено в додатку Б.

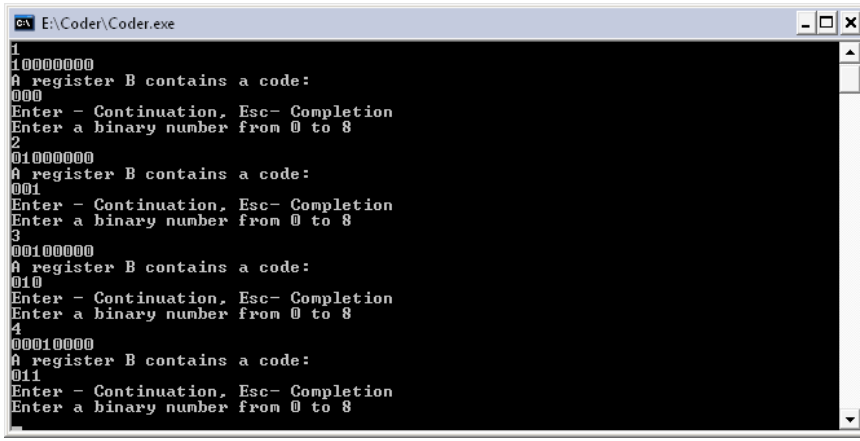


Рис. 11.3. Загальний вигляд вікна консольного додатка, який моделює роботу шифратора

```
program Coder;
{$APPTYPE CONSOLE}
uses
  SysUtils;
//Оголошуємо змінні у програмі
```

```
 VAR  A: ARRAY[1..8] of BYTE;
        B: ARRAY[1..3] of BYTE;
        K1, K2, K3:BOOLEAN;
        i,n:INTEGER;
//Початок виконання програми
    BEGIN
//Початок циклу
    REPEAT
// Запрошення до вводу числа від 0 до 8

    WRITELN('Enter a binary number from 0 to 8');
    READLN(N);
    FOR I:=1 TO 8

// Формуємо двійковий позиційний код числа

    DO IF I = N         THEN A[I]:= 1
      ELSE A[I]:= 0;
    FOR I:=1 TO 8 DO WRITE(A[I]:1, '');
   WRITELN;
//Вивід регістру А
   K1:= (A[5]=1) OR (A[6]=1) OR (A[7]=1) OR (A[8]=1);
   K2:= (A[3]=1) OR (A[6]=1) OR (A[7]=1) OR (A[8]=1);
   K3:= (A[2]=1) OR (A[4]=1) OR (A[6]=1) OR (A[8]=1);
// Запис у регістр В

   IF K1 THEN B[1]:=1 ELSE B[1]:=0;
   IF K2 THEN B[2]:=1 ELSE B[2]:=0;
   IF K3 THEN B[3]:=1 ELSE B[3]:=0;
   WRITELN ('A register B contains a code:');
   FOR I:=1 TO 3 DO WRITE(B[1]:1, ''); WRITELN;

// Завершення роботи або продовження роботи програми
   WRITELN('Enter - Enter a binary number from 0 to 8, 9 -
Completion');

// У разі натискання клавіші 9 – робота  програми припиняється
   UNTIL  n>8 {завершение по клавише «9»}
  { TODO -oUser -cConsole Main : Insert code here }
end.
```

Як бачимо, програмний код не складний, проте, дає можливість побачити механізм роботи шифратора.

11.2. Дешифратор двійкового коду

Як було показано у 9-му розділі, дешифратор виконує по відношенню до шифратора зворотне перетворення. Розглянемо роботу дешифратора двійкового коду на наступному прикладі.

Приклад. Розробити процедуру декодування двійкового трьохрозрядного коду *A* у двійковий позиційний восьмирозрядний код *B*. Для зберігання двійкового трьохрозрядного коду потрібно використовувати вектор $A = | a_1, a_2, a_3 |$. Для зберігання двійкового позиційного восьмирозрядного коду використовувати вектор $B = | b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8 |$. Розробити програму для перевірки правильності роботи дешифратора.

Схематичне зображення дешифратора представлено на рис. 11.4.

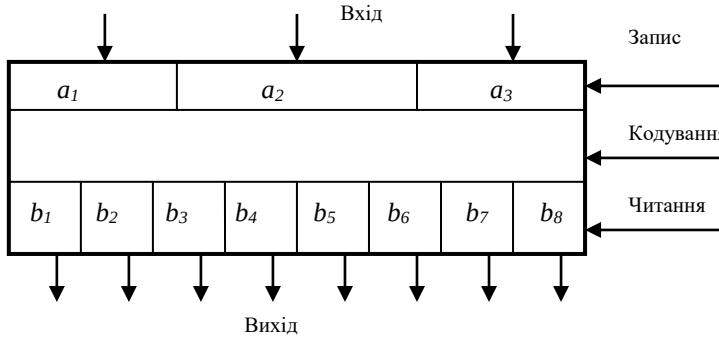


Рис. 11.4. Схема дешифратора

Дешифратор, як і шифратор, працює в три такти.

Такт перший: на вхідні шини подається код у формі потенційних сигналів. Наявність потенціалу означає 1, відсутність – 0. Імпульс по шині «Запис» формує у регістрі *A* трьохрозрядний двійковий код.

Такт другий: по шині «Декодування» надходить імпульс, який ініціює роботу алгоритму декодування коду *A* у код *B*.

Такт третій: код *B* поступає на вхідні шини. При цьому імпульс по команді «Читання» з'являється тільки на одній з восьми шин.

Для ілюстрації роботи дешифратора наведемо декілька прикладів декодування:

- Нехай *A=(000)*, тоді для цього коду *B=(10000000)*;
- Нехай *A=(001)*, тоді для цього коду *B=(01000000)*;
- Нехай *A=(010)*, тоді для цього коду *B=(00100000)*;
- Нехай *A=(011)*, тоді для цього коду *B=(00010000)* і т.д.

На підставі цих варіантів можна зробити кодову таблицю (див. табл. 11.2).

Таблиця 11.2

Кодова таблиця дешифратора

Код A	**Розряди коду *B***							
(a_1, a_2, a_3)	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8
0 0 0	+							
0 0 1		+						
0 1 0			+					
0 1 1				+				
1 0 0					+			
1 0 1						+		
1 1 0							+	
1 1 1								+

Класичний підхід до розв'язання цієї задачі наступний. На базі табл. 11.2 створюється логічна схема декодування (див. рис. 11.5). У схемі використовується вісім елементів **«І»**. Кожен логічний елемент **«І»** містить три входи від трьох розрядів регістру *A*, відповідно a_1, a_2, a_3. На вхід елемента **«І»** подається прямий сигнал або його інверсія. Інверсія сигналу досягається за допомогою логічних елементів **«НЕ»**. Прямий або інверсний вхід визначається кодом (a_1, a_2, a_3), закріпленим за відповідним розрядом b_i регістру *B* (див. табл. 11.2). Наприклад, для коду *A=(000)*, що відповідає першому розряду b_1, усі три розряди a_1, a_2, a_3 повинні подаватися на елемент **«І»** в інверсному вигляді. Для коду *A=(011*) відповідного розряду b_4, перший розряд a_1 потрібно інвертувати, а два інші розряди a_2, a_3, подавати на елемент **«І»** без інвертування.

На підставі рис. 11.5 можна скласти формули для декодування із використанням логічних змінних - *d1,d2,d3,d4,d5,d6,d7,d8.*

$$d1 = \overline{a1} \wedge \overline{a2} \wedge \overline{a3}; \quad d2 = \overline{a1} \wedge \overline{a2} \wedge a3; \quad d3 = \overline{a1} \wedge a2 \wedge \overline{a3};$$
$$d4 = \overline{a1} \wedge a2 \wedge a3; \quad d5 = a1 \wedge \overline{a2} \wedge \overline{a3}; \quad d6 = a1 \wedge \overline{a2} \wedge a3;$$
$$d7 = a1 \wedge a2 \wedge \overline{a3}; \quad d8 = a1 \wedge a2 \wedge a3.$$

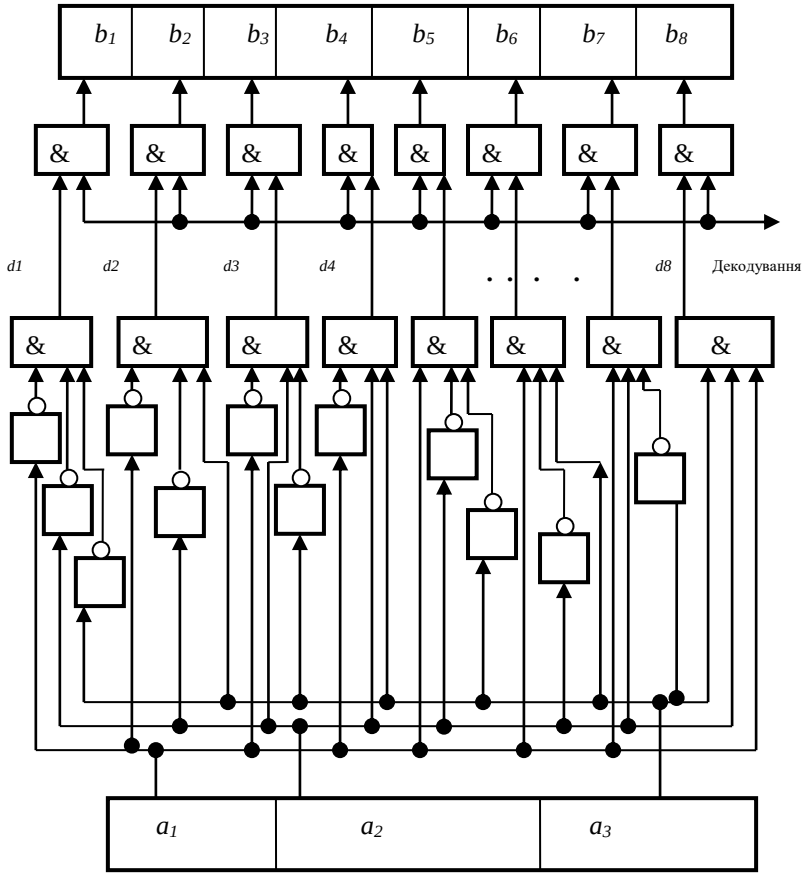


Рис. 11.5. Логічна схема декодування

Лістинг програми та вигляд вікна консольного додатку наведено нижче(див. рис. 11.6.). Лістинг програми *Decoder* на C++ наведено в додатку Б.

```
 program Decoder;
{$APPTYPE CONSOLE}
Uses   SysUtils;
//Оголошуємо змінні у програмі
VAR B: ARRAY[1..8] of BYTE; A: ARRAY[1..3] of BYTE;
          D: ARRAY[1..8] of BOOLEAN;
          i,N:INTEGER;
     BEGIN
     REPEAT
// Введіть через пропуск  у регістр А тризначне двійкове число
     WRITELN('Enter through blanks in a register "A" a three-digit binary
number');
     READLN(A[1],A[2],A[3]);

//Декодування
  D[1]:=(A[1]=0) and (A[2]=0) and (A[3]=0);
  D[2]:=(A[1]=0) and (A[2]=0) and (A[3]=1);
  D[3]:=(A[1]=0) and (A[2]=1) and (A[3]=0);
  D[4]:=(A[1]=0) and (A[2]=1) and (A[3]=1);
  D[5]:=(A[1]=1) and (A[2]=0) and (A[3]=0);
  D[6]:=(A[1]=1) and (A[2]=0) and (A[3]=1);
  D[7]:=(A[1]=1) and (A[2]=1) and (A[3]=0);
  D[8]:=(A[1]=1) and (A[2]=1) and (A[3]=1);

// Формуємо регістр В
   FOR I:=1 TO  8   DO IF D[I]
      THEN B[I]:=1     ELSE B[I]:=0;
    WRITELN ('A register B contains a code:');
    FOR I:=1 TO 8 DO WRITE(B[I]:1, '); WRITELN;
    WRITELN('Enter - Enter a binary number from 000 to 111, n>9
Completion');

 UNTIL  n>9
  { TODO -oUser -cConsole Main : Insert code here }
end.
```

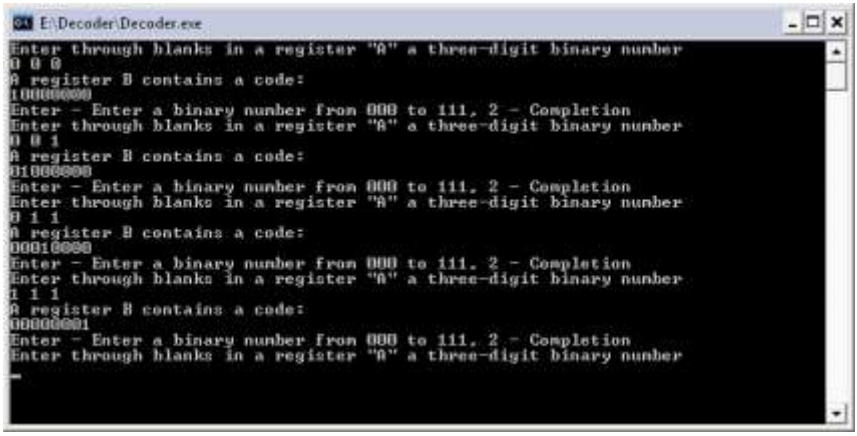


Рис. 11.6. Загальний вигляд вікна консольного додатка, який моделює роботу дешифратора

11.3. Лічильник імпульсів

Розглянемо реалізацію лічильника імпульсів на прикладі розв'язання наступної задачі.

Приклад. Розробити процедуру складання цілого десяткового числа до програмного лічильника за модулем 5, який містить 4 розряди. Як такий лічильник використовує вектор $A=|a_1,a_2,a_3,a_4|$. Розробити програму для перевірки правильності роботи лічильника.

Схематичне зображення лічильника по модулю п'ять (mod 5) подано на рис.11.7.

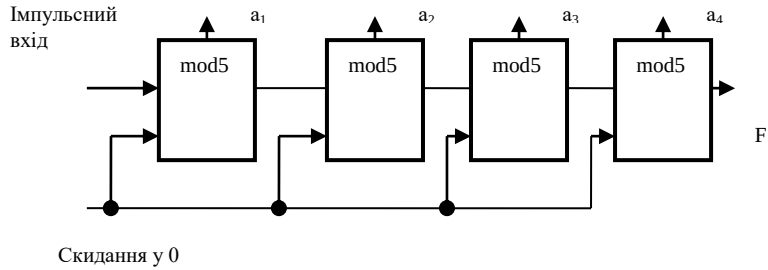


Рис. 11.7. Схема лічильника

Лічильник (рис. 11.7) призначений для підрахунку числа імпульсів, які поступають на його вхід. При надходженні чергового імпульсу

лічильник збільшує ціле число, яке зберігається у ньому в п'ятиричній системі числення на одиницю. Найменше значення цього числа *0000*, найбільше - *4444*. Старший розряд лічильника - *4*, молодший розряд - *1*. Початковий стан лічильника *0000*. У цей стан лічильник приводиться шляхом скидання його поточного значення у *0* за допомогою подачі імпульсу на шину «*Скидання у 0*». Розглянемо алгоритм роботи лічильника.

Нехай лічильник має стан *(*a_4*,*a_3*,*a_2*,*a_1*) = (0000).* Подамо на вхід лічильника послідовність з 12 імпульсів. Стан лічильника змінюється наступним чином: (*0001*), (*0002*), (*0003*) (*0004*). З приходом п'ятого імпульсу відбудеться переповнення першого розряду. Це означає, що перший розряд скинеться в *0*. На виході першого розряду з'явитися імпульс, який поступить на вхід другого розряду. Другий розряд під впливом цього імпульсу перейде із стану *0* у *1*. Таким чином, стан лічильника стане (*0010*). Далі стан лічильника змінюється так: *(0011), (0012), (0013), (0014), (0020), (0021), (0022).* Кінцевий стан лічильника *A(*a_4*,*a_3*,*a_2*,*a_1*)=(0022).* Тобто $12_{10}=0022_5$.

Отже, лічильник перетворює кількість імпульсів, яку людина сприймає у десятиричній системі числення, у п'ятиричну систему. Максимальне число $4444_5=4\cdot5^3+4\cdot5^2+4\cdot5^1+4\cdot5^0=624_{10}$. 625 імпульс веде до скидання усіх чотирьох розрядів лічильника в 0, але при цьому четвертий розряд формує імпульс переповнення – *F*.

Для програмної імітації роботи лічильника будемо використовувати одновимірний цілочисельний масив *COUNT*: *ARRAY[1..K] BYTE*, де *K* - число розрядів лічильника, у нашому випадку *K=4*. Для реалізації режимів роботи лічильника створюємо дві підпрограми: *INIT* і *CHADD*. Процедура *INIT* здійснює скидання лічильнику у 0. Функція *CHADD* формує значення і-го розряду лічильника. Виклик функції імітує значення *i*-го розряду лічильника. Величина *i* є формальним вхідним параметром функції *CHADD*. Виклик функції імітує надходження імпульсу на вхід розряду з номером *i*, тобто складання одиниці до цього розряду. Функція *CHADD* повертає логічне значення **TRUE** при переповнюванні відповідного розряду лічильника. Інакше - функція *CHADD* повертає значення **FALSE**.

Лістинг програми та вигляд вікна консольного додатку наведено нижче(див. рис.11.8.).

```
program Counter;
 {$APPTYPE CONSOLE}
uses
  SysUtils;
// Число розрядів лічильника
```

```
CONST K=4;
//Оголошуємо змінні у програмі
    VAR COUNT: ARRAY [1..K] of BYTE;
    I,N:INTEGER;
    //Процедура скидання у 0 всіх розрядів лічильника
    PROCEDURE INIT;
    BEGIN
    FOR I:=1 TO K DO COUNT[I]:=0
    END;
    //Функція складання  1 до J розряду лічильника
    FUNCTION CHADD(J:INTEGER):BOOLEAN;
    BEGIN
    CHADD:= FALSE;
     COUNT[J]:=COUNT[J]+1;
     IF  COUNT[J]=K
     THEN BEGIN
     //Перенесення 1 у старший розряд
    CHADD:= TRUE;
    //Скидання  J-го розряду у 0
     COUNT[J]:=0  {Сброс }
   END
END;
BEGIN
//Ініціалізація лічильника
INIT;
REPEAT
WRITELN('Enter an integer from 0 to 10'); READLN(N);
FOR I:=1 TO N
//Складання 1 та молодшого розряду лічильника
DO IF CHADD(1)

//Додавання 1 до другого розряду лічильника
  THEN IF CHADD(2)

 //Додавання 1 до третього розряду лічильника
    THEN IF CHADD(3)
    //Складання  1 та старшого розряду лічильника
 THEN IF CHADD(4)
//Скидання значення у лічильнику
 THEN WRITELN('Repletion of counter. Up cast of value counter in 0');
FOR I:=K DOWNTO 1 DO WRITE(COUNT[I]:1, '');
WRITELN;
```

```
WRITELN('Enter - Enter an integer from 0 to 9, N=10 - Completion');
UNTIL N>9
  { TODO -oUser -cConsole Main : Insert code here }
end.
```

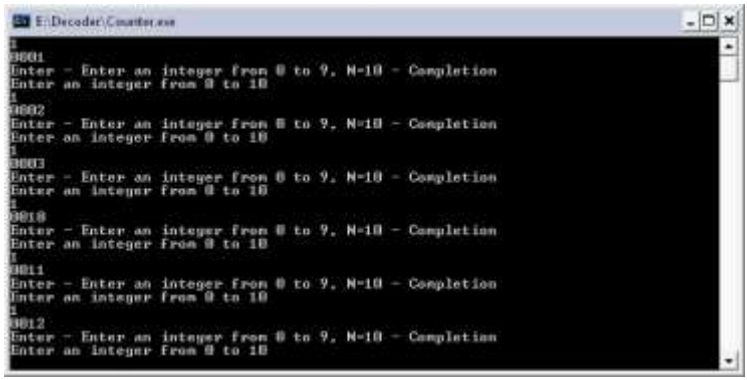


Рис. 11.8. Загальний вигляд вікна консольного додатка, який моделює роботу лічильника

11.4. Суматор по модулю два

У багатьох розглянутих у попередніх розділах цифрових схемах використовується логічний елемент – суматор по модулю два. У наступному прикладі ми розглянемо програмне моделювання цього компоненту комбінаційних схем.

Приклад. Розробити процедуру кодування двійкової *m*-розрядної кодової комбінації, яка зберігається у регістрі *A*, шляхом складання одного з *m+1* – го контрольного розряду із перевіркою на непарність, див. рис. 11.9.

Алгоритм роботи схеми, яка зображена на рис. 11.9, наступний.

Такт перший: у перші *m*-розряди регістру *A* записуються інформаційні позиції кодової комбінації. Число таких позицій – *m*.

Такт другий: обчислюється сума $Z = \mathrm{mod}_2 \sum_{i=1}^{m} a_i$.

Такт третій: за результатом обчислення значення *Z*, встановлюється 1 або 0 у *m+1-y* позицію у регістр *A*. Кодова комбінація сформована з перевіркою на непарність та готова до передачі.

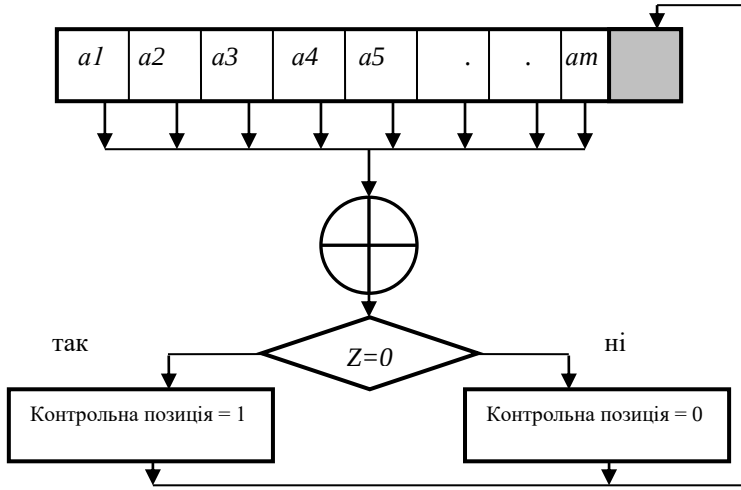


Рис. 11.9. Кодування контрольної позиції

Лістинг програми та вигляд вікна консольного додатку наведено нижче(див. рис. 11.10.).

```
program Summatot_mod2;
{$APPTYPE CONSOLE}

Uses   SysUtils;
//Число інформаційних розрядів
CONST MRI=10;
VAR A: ARRAY[1..MRI+1] of BYTE;
I, PZ,X:INTEGER;
begin
REPEAT
WRITELN('Enter positions of binary code combination with 1-th for 10-th', MRI);
FOR I:=1 TO MRI DO READ(A[I]);
PZ:=0;
//Знаходження суми по модулю два
FOR I:=1 TO MRI DO PZ:=PZ+A[I];
PZ:=PZ mod 2;
//Запис контрольної позиції у регістр A
IF ODD(PZ)
THEN A[MRI+1]:=0
```

```
ELSE A[MRI+1]:=1;
// Регістр А містить код
WRITELN('Register A contains a code:');
FOR I:=1 TO MRI+1 DO WRITE(A[1]:1, ''); WRITELN;
WRITELN(' Enter, M<10 -  Completion');
UNTIL MRI<10
  { TODO -oUser -cConsole Main : Insert code here }
end.
```

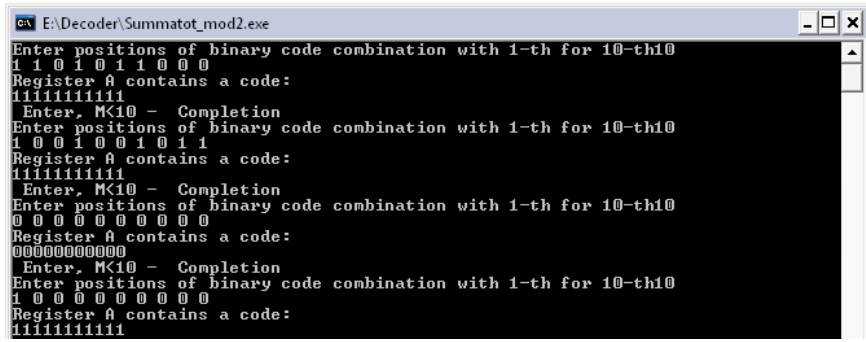


Рис. 11.10. Загальний вигляд вікна консольного додатка, який моделює роботу суматору по модулю два

11.5. Регістри

11.5.1. Циклічний регістр двійкового коду

На рис. 11.11 представлена схема регістру A, що містить $m+1$ бітовий розряд. Спочатку виконується запис паралельного коду за вхідними шинами в розряди регістру A. Наступним кроком є здійснення циклічного зсуву кодової комбінації.

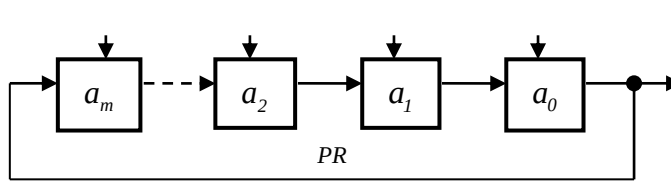


Рис. 11.11. Циклічний регістр

Приклад. Розробити процедуру циклічного зсуву вправо на *N* тактів кодової комбінації $A=(K_6,K_5,K_4,K_3,K_2,K_1,K_0)$. Наприклад, для кодової комбінації $A=(1\,0\,1\,1\,1\,0\,1)$ та $N=8$ ми повинні отримати такірезультати(див. табл. 11.3.).

Таблиця 11.3

Кодова таблиця регістру *A*

Етапи	*Розряди регістру A*
Початковий стан	1 0 1 1 1 0 1
Такт 1	1 1 0 1 1 1 0
Такт 2	0 1 1 0 1 1 1
Такт 3	1 0 1 1 0 1 1
Такт 4	1 1 0 1 1 0 1
Такт 5	1 1 1 0 1 1 0
Такт 6	0 1 1 1 0 1 1
Такт 7	1 0 1 1 1 0 1
Такт 8	1 1 0 1 1 1 0

Лістинг програми та вигляд вікна консольного додатку наведено нижче(див. рис. 11.12.).

```
program Cicle_register;

{$APPTYPE CONSOLE}

Uses   SysUtils;
//MR - Число розрядів кодової комбінації
CONST MR=6;
//Оголошуємо змінні у програмі
VAR A: ARRAY[0..MR] of BYTE;
J,I,N,PZ: INTEGER;
begin
REPEAT
//Введіть через клавішу «Enter» позиції двійкової
//кодової комбінації з 6-ої по 0-ву
WRITELN('Enter (through Enter) positions of binary code combination with 6-th for 0- th');
FOR I:=MR DOWNTO 0 DO READ(A[I]);
// Ведіть число зсувів
WRITELN('Conduct the number of changes to 8');
```

```
READLN(N);
// Регістр А містить код
WRITELN('Register A contains a code');
FOR I:=1 TO N
DO BEGIN
// Запам'ятовуємо правий крайній розряд
PZ:=A[0];
// Зсуваємо  вправо всі інші розряди
FOR J:=0 TO MR-1
DO A[J]:=A[J+1];
A[MR]:=PZ;
//Вивід кодової комбінації на кожному такті роботи програми
WRITE('Time', I:1,': ');
FOR J:=6 DOWNTO 0 DO WRITE(A[J]:1, '');
WRITELN;
END;
WRITELN('Enter (through Enter) positions of binary code combination with 6-th for 0- th, N<8 - Completion');
//Завершення роботи програми
UNTIL N<8

  { TODO -oUser -cConsole Main : Insert code here }
end.
```

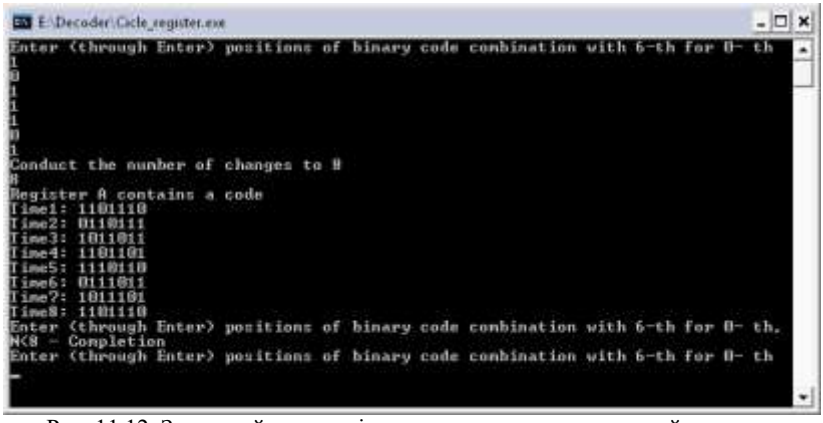


Рис. 11.12. Загальний вигляд вікна консольного додатка, який моделює роботу циклічного регістру двійкового коду

11.5.2. Циклічний регістр із зворотними зв'язками по модулю два

На рис. 11.13 представлена схема регістру *A*, який містить m - бітовий розряд. Гнізда бітової пам'яті із номерами $m,m-1,m-2,...,2,1$ спочатку використовуються для запису інформаційною частини коду, яка поступає з вхідних шин. На наступному етапі зміст лівого гнізда (0 або 1) перезаписується у праве гніздо, тобто $m \to m-1,\ m-1 \to m-2,$ … , $2 \to 1$. Гніздо із номером 1 надсилає інформацію в канал зв'язку. Контрольні позиції коду формуються на кожному такті за допомогою зворотних зв'язків.

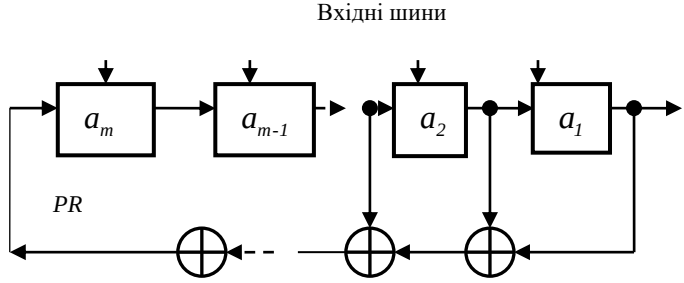


Рис. 11.13. Кодер циклічного коду (*n, m*)

Приклад. Розглянемо приклад кодера циклічного коду (*7, 3*), для якого регістр зсуву показано на рис. 11.14. Потрібно запрограмувати роботи цього цифрового пристрою для наступної комбінації інформаційних розрядів циклічного коду – 0 1 1. Формула зворотного зв'язку має такий вигляд - $PR = a_1 \oplus a_3$.

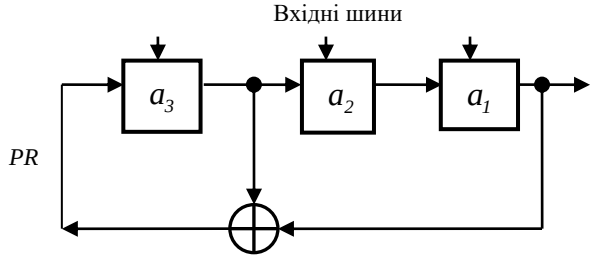


Рис. 11.14. Кодер циклічного коду (*7, 3*)

Алгоритм роботи схеми, що зображена на рис. 11.14, наступний.

Такт перший: здійснюється зсув позиції коду вправо на один розряд. Одиниця із крайнього правого розряду із номером 1 надходить до каналу зв'язку. Суматор по модулю два формує одиницю, яка записується у гніздо пам'яті із номером три.

Такт другий: здійснюється другий зсув позицій коду вправо на один розряд. Одиниця із крайнього правого розряду із номером 1 надходить до каналу зв'язку. Суматор по модулю два формує нуль, який записується у гніздо пам'яті з номером три.

Робота пристрою на 3,4,5,6,7 тактах аналогічна. Після сьомого такту (останнього) первинна кодова комбінація повинна бути тією ж самою - 1 0 1 1 1 0 1.

Отримана кодова таблиця інформаційних позицій регістру *A* показана у таблиці 11.4.

Лістинг програми та вигляд вікна консольного додатку наведено нижче(див. рис. 11.15.).

```
program Coder_7_3;
{$APPTYPE CONSOLE}
Uses   SysUtils;
//MR - Число розрядів кодової комбінації
CONST MR=3;
VAR A: ARRAY[1..3] of BYTE;
J,I,N,PZ,K: INTEGER;
begin
REPEAT
//Введіть через клавішу "Enter" позиції двійкової
//кодової комбінації з 1-ої по 3-тю
WRITELN('Enter positions of binary code combination with 1-th for 3-th');
FOR I:=3 DOWNTO 1 DO READ(A[I]);

// Ведіть число зсувів
WRITELN('Conduct the number of changes to 7');   READLN(N);

// Регістр А містить код
WRITELN('Register A contains a code');
FOR I:=1 TO N DO BEGIN
//Формула зворотного зв'язку по модулю два  (8-2)
  PZ:= (A[1]+A[3]) mod 2;
//У канал зв'язку
  K:=A[1];
```

```
// Зсуваємо  вправо всі інші розряди
  FOR J:=1 TO MR-1
  DO A[J]:=A[J+1];
  A[MR]:=PZ;
  WRITE('Time', I:1,': ');
  FOR J:=3 DOWNTO 1 DO WRITE(A[J]:1, '');
  WRITE(', Code entering communication channel   ->', K:1);
  WRITELN;
  END;
WRITELN('Enter positions of binary code combination
with 1-th for 3-th, N<7 - Completion');
UNTIL N<7
  {TODO -oUser -cConsole Main : Insert code here }
end.
```

Таблиця 11.4

Кодова таблиця інформаційних позицій регістру *A*

Такт	**Гнізда регістру *A***			Канал зв'язку
	a_3	a_2	a_1	
0	0	1	1	-
1	1	0	1	1
2	0	1	0	1
3	0	0	1	0
4	1	0	0	1
5	1	1	0	0
6	1	1	1	0
7	0	1	1	1

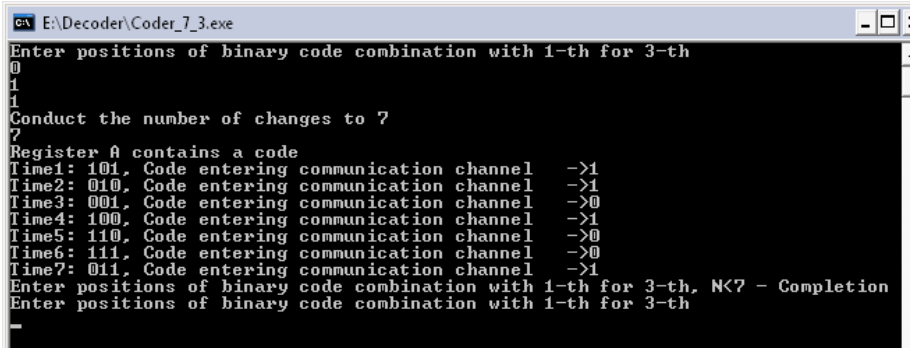


Рис. 11.15. Загальний вигляд вікна консольного додатка, який моделює роботу кодера циклічного коду зі зворотним зв'язком

11.5.3. Декодуючий регістр двійкового циклічного коду

В останньому прикладі розглянемо декодер двійкового циклічного коду, схема якого подана на рис. 11.16.

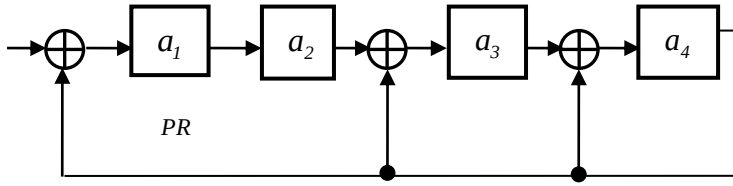


Рис. 11.16. Декодуючий регістр циклічного зсуву коду (*7, 3*)

Потрібно запрограмувати роботи цього цифрового пристрою для перевірки коду, який надходить з каналу зв'язку К=1 0 0 1 0 1 1.

Гнізда бітової пам'яті із номерами 1,2,3,4 використовуються для запису перших чотирьох розрядів коду К, який надходить із каналу зв'язку, тобто початковий стан регістру А=(0 0 0 0). Надалі на кожному такті зміст правого гнізда a_4 (0 або 1) формує зворотний зв'язок $PZ = a_4$. Решта гнізд, із урахуванням роботи алгоритму суматора по модулю два, а також значення PZ, послідовно змінюють своє значення відповідно до наступних формул:

$$a_4 = a_3 \oplus PZ;$$
$$a_3 = a_2 \oplus PZ;$$
$$a_2 = a_1 \oplus PZ;$$
$$a_1 = K_i \oplus PZ,$$

де K_i - елемент коду, який приймається пристроєм із каналу зв'язку та такті із номером – *i.*

Отримана кодова таблиця інформаційних позицій регістру *A* для коду, який надходить із каналу зв'язку (1 1 0 1 0 0 1), показана в таблиці 11.5.

Лістинг програми та вигляд вікна консольного додатку наведено нижче(див. рис. 11.17.).

Таблиця 11.5

Кодова таблиця інформаційних позицій регістру *A*

Такт	Канал зв'язку	Гнізда декодуючого регістру *A*			
		a_1	a_2	a_3	a_4
0	-	0	0	0	0
1	1	1	0	0	0
2	1	1	1	0	0
3	0	0	1	1	0
4	1	1	0	1	1
5	0	1	1	1	0
6	0	0	1	1	1
7	1	0	0	0	0

```
Program Decoder_7_3;
{$APPTYPE CONSOLE}
uses  SysUtils;
//N - Число розрядів кодової комбінації, число зсувів
CONST N=7;
//MR - Число розрядів регістру декодера
MR=4;
VAR A: ARRAY[1..MR] of BYTE; K: ARRAY[1..N] of BYTE;
    J,I,PZ: INTEGER;
begin
REPEAT
//Скидання змісту регістру А у нуль
FOR J:=1 TO 4 DO A[J]:=0;
//Введіть через клавішу "Пропуск" двійковий код
//з 1-ої по 7-му позиції
WRITELN('Enter through blanks  positions of binary code combination
with 1-th for 7-th');
FOR I:=N DOWNTO 1 DO READ(K[I]);
// Ведіть число зсувів
WRITELN('Conduct the number of changes to', N);
// Регістр А містить код
WRITELN('Register A contains a code');
FOR I:=1 TO N
DO BEGIN
//Значення зворотного зв'язку
```

```
    PZ:= A[4];
  //Формула зворотного зв'язку по модулю два  для A4
    A[4]:=(A[3]+PZ)mod 2;
  //Формула зворотного зв'язку по модулю два  для A3
    A[3]:=(A[2]+PZ)mod 2;
  //Формула зворотного зв'язку по модулю два  для A2
    A[2]:=A[1];
  //Формула зворотного зв'язку по модулю два  для A1
    A[1]:=(K[I]+PZ)mod 2;
    WRITE('Time',I:1,',Code entering communication channel ->',
K[I]:1,");
    FOR J:=1 DOWNTO 4 DO WRITE(A[J]:1,");
    WRITELN;
    END;
  WRITELN('Enter positions of binary code combination with 1-th for 4-
th, N<7 - Completion');
  UNTIL N<7
   { TODO -oUser -cConsole Main : Insert code here }
  end.
```

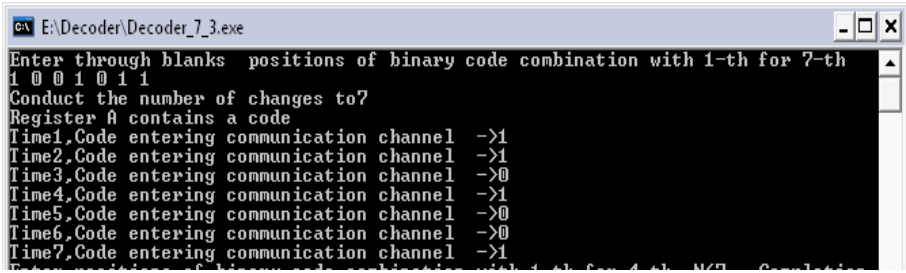


Рис. 11.17. Загальний вигляд вікна консольного додатка, який моделює роботу декодуючого регістру циклічного зсуву

Запитання для самоперевірки

1. Які засоби програмування дозволяють моделювати режими експлуатації цифрових елементів комбінаційних схем систем захисту інформації з обмеженим доступом?
2. З яких етапів складається синтез програмної моделі шифратора?
3. З яких етапів складається синтез програмної моделі дешифратора?

4. Складіть програми для моделювання роботи лічильника імпульсів, суматора по модулю два, циклічного регістру двійкового коду, декодуючого регістру циклічного зсуву та кодера циклічного коду зі зворотним зв’язком на мовах об’єктного проектування C++ або C#.

Література для самостійної підготовки за темою:

29, 30, 33.

Розділ 12. ПЕРСПЕКТИВИ РОЗВИТКУ ЦИФРОВИХ СИСТЕМ ЗАХИСТУ ІНФОРМАЦІЇ З ОБМЕЖЕНИМ ДОСТУПОМ

Забезпечення інформаційної безпеки здійснюється використанням різних способів, засобів і прийомів, зокрема засобами технічного захисту із застосуванням цифрових автоматів.

Вибір методів аналізу стану захисту інформації з обмеженим доступом залежить від конкретного рівня і сфери організації захисту. На програмно-технічному та технологічному рівнях, які передбачають використовування цифрових автоматів, здійснюється ідентифікація і перевірка дійсності користувачів, управління доступом, протоколювання і аудит, криптографія, екранування та ін.

Нині важливою умовою забезпечення захисту інформації з обмеженим доступом є не стільки секретність, конфіденційність інформації, скільки її доступність, цілісність, захист від різних загроз. Отже, система захисту інформації має відповідно реагувати і гарантувати ефективну діяльність у цьому напрямі.

Іншим завданням захисту є забезпечення незмінності інформації під час її зберігання або передачі, тобто забезпечення її цілісності. Таким чином конфіденційність інформації, яка забезпечується за допомогою криптографічних методів не є головною вимогою при проектуванні систем захисту інформації. Виконання процедур криптокодування і декодування може уповільнити передачу даних та зменшити доступ до них через те, що користувач буде позбавлений можливості своєчасного і швидкого доступу до цих даних та інформації. Саме тому забезпечення конфіденційності інформації має відповідати можливості доступу до неї.

Сучасні методи впливу на інформацію умовно можна поділити на електронні та неелектронні. Електронні методи впливу застосовуються у тих випадках, коли повідомлення закріплюються на електромагнітних носіях, котрі призначені для оброблення за допомогою засобів обчислювальної техніки. Вони полягають у знищенні, викривленні, копіюванні повідомлень, які зберігаються на цих пристроях. Такі дії можуть бути вчинені лише за допомогою технічного і програмного забезпечення. Неелектронні методи за своєю суттю мають той самий зміст, але реалізуються без використання засобів обчислювальної техніки для

впливу на повідомлення, закріплення на інших, передусім паперових, носіях інформації.

При зміні способу зберігання інформації з паперового виду на цифровий, з'явився головне питання - як цю інформацію захистити, адже дуже велика кількість факторів впливає на збереження конфіденційних даних. Для того щоб організувати безпечне зберігання даних, насамперед потрібно провести аналіз загроз, для правильного проектування схем інформаційної безпеки.

З розвитком процесу автоматизації та інформатизації суспільства напрямок інформаційної безпеки та контролю доступу стає стратегічно важливим навіть щодо тих підприємств, чия діяльність безпосередньо не пов'язана з комп'ютерними або Інтернет-технологіями. Це може відноситися як до контролю доступу до приміщення, так і доступу до певної інформації.

З метою забезпечення захищеності інформаційних мереж були створені цифрові системи, які класифікують мережеву активність різних програм. У разі виникнення загрозливої ситуації такі системи пропонують користувачу припинити дії несанкціонованого програмного забезпечення (ПЗ) і повернутися у попередній стан системи до якого у ній були проведені зміни. Проте більшість сучасних технічних систем мережевої безпеки не має можливості самонавчання і оперує тільки закладеними розробниками в них правилами. Поява небажаного нових поколінь програмного забезпечення, яке використає досі не вивчені уразливості інформаційної безпеки, ставить нові вимоги до систем мережної безпеки.

В даному розділі коротко розглянуті новітні методи побудови систем захисту інформації з обмеженим доступом у мережах із використовуванням останніх розробок у галузі цифрових систем, зокрема - багатоагентних систем і методів штучного інтелекту (ШІ).

12.1. Багатоагентні системи

Оскільки захист мережі - задача комплексна, то крім засобів персонального захисту (антивіруси, мережні екрани і т. д.) ведуться розробки багатоагентних систем. Кожний агент, див. див. рис. 12.1, відповідає за певну частину завдання і загальне рішення приймається в результаті їх скоординованих дій. Агенти можуть мати реалізовані як цифрові елементи або елементи штучного інтелекту. В процесі роботи агенти обмінюються повідомленнями із допомогою спеціальних протоколів. Звичайно існує агент, який управляє діями інших. Багатоагентний підхід детально розглянутий у [4,44,45]. Перерахуємо особливості багатоагентних систем, які дозволяють ефективно використовувати їх в системах мережної безпеки.

Гнучкість. Агенти можуть створювати собі подібних і розміщувати їх на нових вузлах мережі, тому багатоагентні системи легко адаптуються до будь-якої мережної архітектури і адекватно відповідають на зміни в конфігурації мережного устаткування.

Можливість централізованого адміністрування. Внесення змін в роботу агентів можливо виконувати централізовано і по протоколах взаємодії агентів передаватися на всі точки забезпечення безпеки.

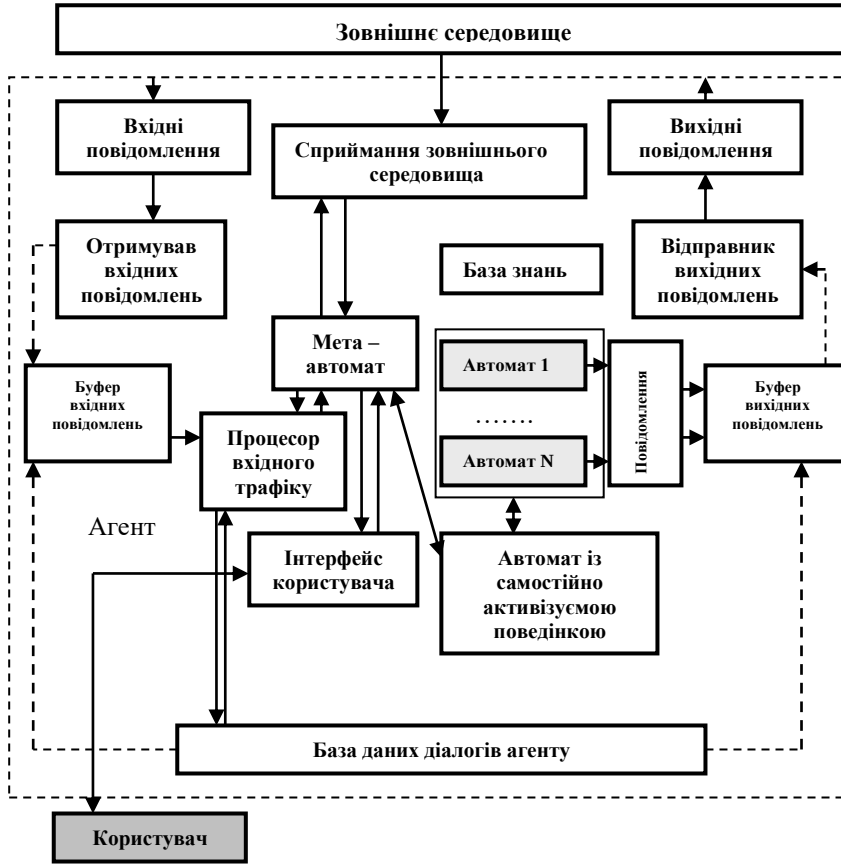


Рис. 12.1. Схема типового агенту

Економічність. Система рівномірно розподілена по всьому периметру захисту. Ця особливість забезпечує оптимальний розподіл обчислювальних ресурсів мережі.

Підвищена надійність. Оскільки агенти можуть існувати самостійно, і вони розподілені на всіх вузлах мережі, тобто система захисту не має центру, та атакувати її буде складніше, ніж мережу із централізованим сервером захисту. Розподілена по мережі інформація і розподілений захист вимагають від зловмисника проводити атаку багатьох вузлів одночасно.

В [4, 19, 44] розглянуті багатоагентні системи захисту мережі від зовнішніх загроз. Особливості цього підходу забезпечують захист від складних загроз і дозволяють наочно представити поточний стан всіх агентів і мережі в цілому. Агенти розділені по роду діяльності і з'єднані у команди. Наприклад, у [45] виділені наступні класи агентів команд захисту:

- обробки інформації (семплери);
- виявлення атаки (детектори);
- фільтрації (фільтри);
- агенти розслідування.

Семплери здійснюють збір даних для подальшого виявлення мережних аномалій або зловживань детектором. Фільтри відповідальні за фільтрацію трафіку за правилами, представленими детекторами. Агент розслідування намагається знешкоджувати агентів атаки. Команда агентів захисту спільно реалізує механізм захисту і може взаємодіяти по різних схемах. В одній з схем при виявленні початку атаки діє детектор тієї команди, на чию мережу направлена атака. Він посилає запит агентам-семплерам інших команд з метою отримання інформації, яка може бути застосована для пошуку аналогів вказаній атаці. Семплери інших команд відповідають на запит, посилаючи необхідні дані. Ця інформація істотно підвищує шанси на виявлення атаки. У разі виявлення вірогідного джерела атаки детектор мережі-жертви посилає інформацію про адресу агента атаки детектору команди, в мережі якій може знаходитися цей агент, з метою його деактивації.

Розглянемо невеличкий приклад використання положень теорії автоматів для розробки агенту.

Як відомо однією з загроз для інформаційної безпеки WEB додатку є недостатня фільтрація даних, наприклад випадок коли відвідувач має можливість використовувати символи < та >. Вихід з цієї ситуації може бути наступним:

- видалення символів < та >, але у цьому випадку зміст повідомлення, див. рис. 12.1 може змінитися;

- блокування повідомлення із символами < та >, але у цьому випадку можуть бути заблоковані важливі повідомлення;
- заміна символів < та > на безпечні - *&lt* та *&gt*.

Абстрактний автомат, який використовується у машині Тюрінга [32], сам по собі не є універсальним і не зручний для практичного застосування. Другий з вказаних недоліків знімається при переході від абстрактного автомата до структурного, головна відмінність якого полягає в паралелізмі по входах і виходах. Розглянемо модель (рис. 12.2), що базується на структурному автоматі, в яку для забезпечення універсальності і можливості практичного застосування введені формувачі вхідних дій 1, а найголовніше - об'єкти управління 2.

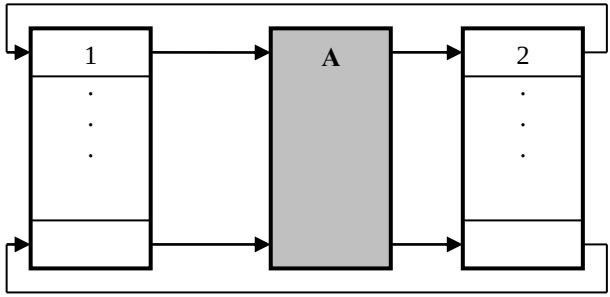


Рис. 12.2. Схема зв'язків автомата

Цю модель будемо називати - "схема зв'язків автомату". В ній формувач 1 перетворить інформацію яка поступає від джерела, наприклад із каналу зв’язку, у вхідні дії. Наприклад, шляхом порівняння значення лічильника з нулем. В цій схемі об'єктами управління можуть бути пристрої, в яких по командах автомата можливе виконання скільки завгодно складних операцій (у тому числі і над пам'яттю).

В даній моделі вихідні дії формуються в певних "точках" простору станів, що задається графом переходів автомата. Тому процес управління об'єктами повністю задається цим графом в наочній і зрозумілій формі.

Ефективність використовування цієї моделі розглянемо на прикладі задачі розпізнавання ланцюжків символів – < та > , тобто тегів.

Як основний спосіб завдання автоматів використовуються графи переходів [6, 8, 14, 25, 27 та ін.].

При цьому, замість двійкових входів і виходів використовуються вхідні і вихідні дії. Вхідні дії можуть бути подіями і вхідними змінними, а вихідні дії - діями, які можуть виконуватися у вершинах, а також на дугах і петлях графа переходів. Для забезпечення універсальності вхідні змінні і

дії реалізуються функціями, число і вид яких не фіксовано, а визначається схемою зв'язків, яка містить також автомат і об'єкти управління.

В задачі як об'єкти управління виступають - лічильник, пристрій для заміни символів і два індикатори "Пропустити" і "Відкинути". Значення лічильника може бути:

- скинуто у нуль;
- збільшено або зменшено на одиницю.

З виходу лічильника на вхід автомата наджодить інформація про те, чи рівне значення лічильника нулю.

При цьому граф переходів автомата Милі реалізується одним оператором *switch*. Якщо на всіх вхідних дугах деякої вершини цього графа присутні однакові дії, то вони можуть бути перенесені в неї. Таким чином, автомат Милі перетвориться в більш компактний змішаний автомат, який, як і автомат Мура, реалізується за шаблоном, що містить два оператори - *switch*.

На рис. 12.3 та 12.4 відповідно, показані схеми зв'язків і граф переходів автомата Милі.

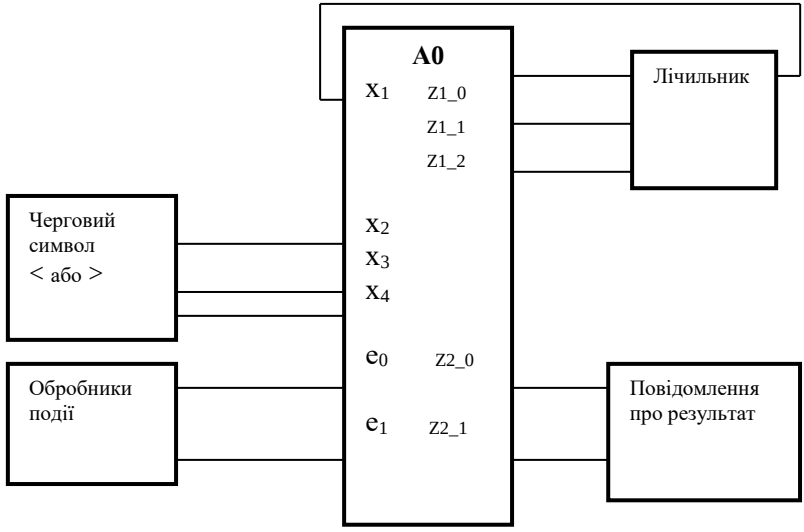


Рис. 12.3. Розпізнавання послідовностей символів < та > довільної глибини. Схема зв'язків автомата

Відповідно до запропонованої програмної реалізації автоматів, номер події передається функції, що реалізує автомат, як перший параметр. Другим параметром передається код чергового символу.

Подія *e0* призначена для ініціалізації автомата перед початком обробки нового виразу, і тому петлі в графах переходів, помічені цією подією, мають перший пріоритет. Подія *e1*, вказана на рис. 12.3 не використовується в графах переходів і призначена для введення чергового символу. В лістингу приведена програма для розпізнавання послідовностей символів < та > довільної глибини, що реалізовує автомат Милі.

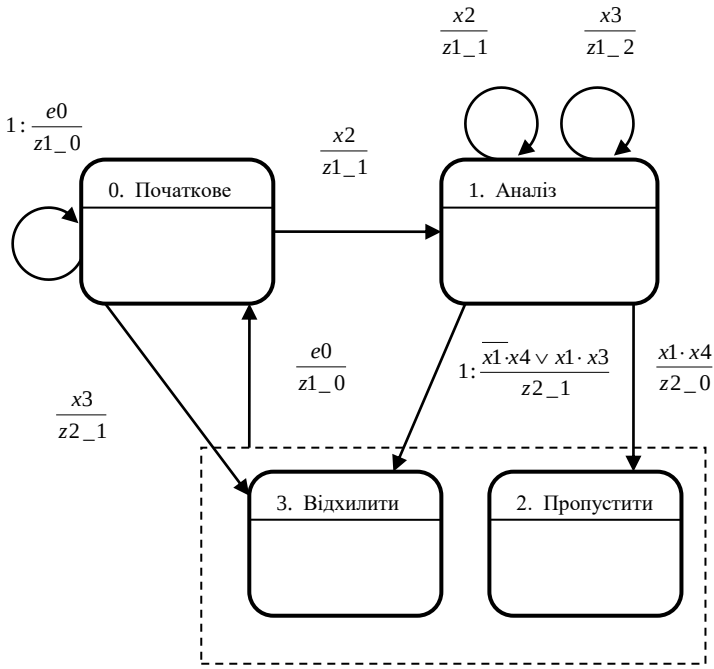


Рис. 12.4. Розпізнавання послідовностей символів < та > довільної глибини. Граф переходів автомата Милі

На рис 12.3. та 12.4. відповідно прийняти наступні позначення:

x1– значення лічильника - 0;
x2 – символ ‘<’;

x3 – символ ‘>’;
x4 – кінець виразу;
e0 – початок обробки нового виразу (ініціалізація);
e1 – обробка наступного символа;
z1_0 – обнуління лічильника;
z1_1 – збільшення значення лічильника на одиницю;
z1_2 – зменшення значення лічильника на одиницю;
z2_0 – пропустити вираз;
z2_1 – відкинути вираз.

Листінг програми (реалізація автомата Мілі). (Компілятор - C++ Builder 2009).

```
#include <vcl.h>
#include <stdio.h>
#include <string.h>
#pragma hdrstop
#include <tchar.h>
//---------------------------------------------------------------------------
#pragma argsused
// Стани автомата
int y0=0;

// Лічильник символів
int i=0;

int x1()
{return i==0;}
int x2(char c)
{return c=='<';
c=='&lt';}
int x3(char c)
{return c=='>';
c=='&gt';}
int x4(char c)
{return c==0; }
void z1_0()
{i=0;}
void z1_1()
{i++;}
void z1_2()
{i--;}
void z2_0()
{printf( "\n DOPUSTIT.\n" ) ; }
```

```
void z2_1()
{printf( "\n OTVERGNUT.\n" ) ; }
void A0( int e, char c )
{
  switch( y0 )
  {
   case 0:
     if(x2(c))
    {z1_1();  y0 = 1;}
     else    if(x3(c))
    {z2_1();  y0=3;}
     else
     if(e==0)  {z1_0();}
   break ;
   case 1:
     if(!x1() && x4(c) || x1() && x3(c))
     {z2_1(); y0=3;}
     else
     if(x1() && x4(c))
     {z2_0();  y0=2;}
     else
     if(x2(c))
     {z1_1();}
     else
     if(x3(c))
     {z1_2();}
   break ;
   case 2:
     if(e==0)
     {z1_0();  y0=0;}
   break ;
   case 3:
     if(e==0)
     {z1_0();  y0=0;}
   break ;
  }
}
void main()
{
  char  str[100] = "" ;
  int   j ;
  for(;;)
```

```
  {
   printf("\n VVEDITE STROKY: ") ;
    scanf( "%s", str ) ;
    printf( " VVEDENA STROKA: %s", str ) ;
   A0(0, 0) ;
   for(j=0 ; j<=strlen(str); j++)
     A0(1, str[j]) ;
  }
}
```

Вигляд вікна консольного додатку наведено нижче, див. рис. 12.5.

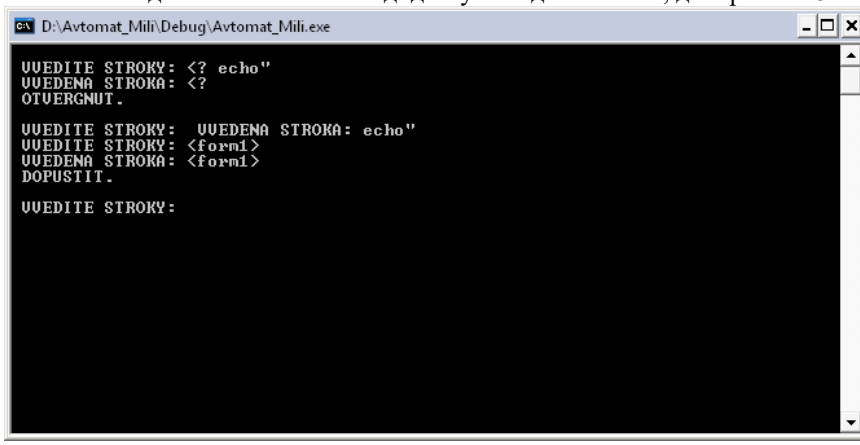


Рис. 12.5. Загальний вигляд вікна консольного додатку який моделює роботу автомату Мілі для розпізнавання послідовностей символів < та > довільної глибини

У зв'язку з простотою навденого прикладу коментар для лістингу програми ненаведений.

Таким чином, використовування багатоагентного підходу та інтелектуальних алгоритмів обробки даних при розробці систем забезпечення безпеки інформаційних мереж значно підвищує їх якісні характеристики.

12.2.Системи штучного інтелекту

Сучасні апаратні та програмні засоби забезпечення інформаційної безпеки, що реалізують методи сигнатурного аналізу для виявлення втручань (порушень, загроз, атак, відмов), або методи статистичного

аналізу для виявлення аномалій не можуть забезпечити гарантованого або навіть прийнятного захисту кібернетичного простору інформаційних систем і мереж від наростаючої загрози все нових і різноманітних атак з боку швидко прогресуючих спеціалістів та аматорів, у тому числі терористів.

Увага дослідників і розробників спеціалізованих технічних систем та програмного забезпечення для захисту інформаційних систем і мереж все частіше звертається до евристичних методів, нейромережевих технологій і до інших еволюційних механізмів, що довели свою ефективність у біологічних системах, забезпечуючи їхнє виживання в найскладніших умовах зовнішнього середовища, яке швидко змінюється. До них належать генетичні алгоритми й відповідні еволюційні механізми: селекції, схрещування і мутації, еволюційні обчислення, еволюційне моделювання та механізми штучного імунітету. Значна кількість початкових розробок у цих галузях показали серйозні науково-технічні проблеми, що існують у розвитку та реалізації еволюційних підходів у завданнях виявлення вторгнень у комп'ютерні системи та мережі.

Використання у комп'ютерних мережах штучного інтелекту дозволяє ввести у системи захисту властивість самонавчання та забезпечити швидке виявлення нових загроз.

Штучна нейронна мережа (НМ) є спрощеною моделлю мозку і представляє набір нейронів, з'єднаних між собою певним чином [16], наприклад, як показано на рис. 12.6. Нейронні мережі дозволяють вирішувати різні практичні завдання, пов'язані, в основному, з розпізнаванням і класифікацією образів. Безперечні переваги нейронних мереж полягають в тому, що вони можуть в автоматичному режимі накопичувати нові знання в процесі навчання і приймати відповідні рішення.

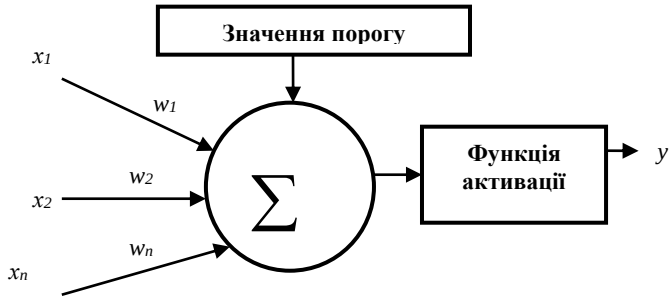


Рис. 12.6. Структурна схема моделі нейрону МакКалока-Пітса

У запропонованій у 1943 році моделі МакКалока-Пітса вхідні сигнали $x_j(\ j=1,2,...N\ )$ складаються з урахуванням відповідної ваги w_{ij} *у суматорі* $-\sum$. Після суматора результат порівнюється із пороговим значенням w_{i0}. Вихідний сигнал нейрона y_i визначається такою залежністю:

$$y_i = f(\sum w_{ij} \cdot x_j(t) + w_{i0}),$$

де $f(u_i)$ - функція активації

$$f(u) = \begin{cases} 1 & для \quad u > 0; \\ 0 & для \quad u \le 0. \end{cases}$$

Зараз центр досліджень у галузі інформаційної безпеки із застосуванням нейронної мережі змістився до систем виявлення втручань (див. рис. 12.7.).

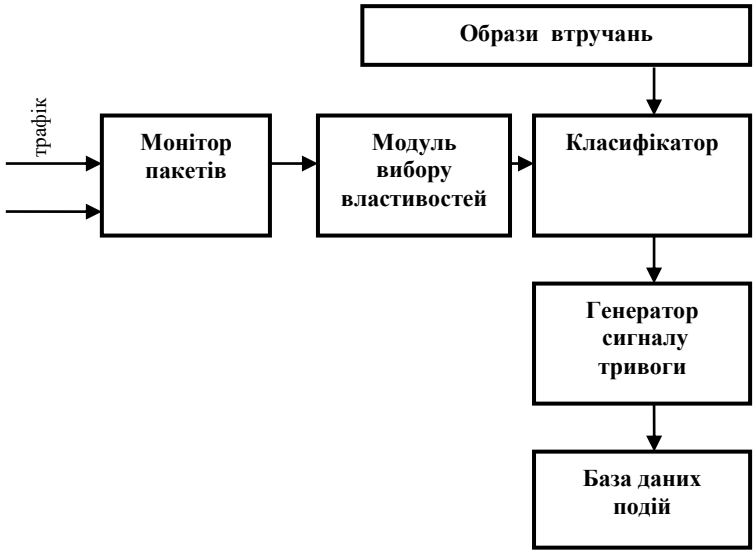


Рис. 12.7. Система виявлення втручань із застосуванням нейронної мережі

Дослідження відомих методів та алгоритмів виявлення втручань, у тому числі алгоритму негативної селекції в концепції штучного імунітету,

показали необхідність вирішення і подолання ряду складних і актуальних науково-технічних проблем, пов'язаних з безперервним збільшенням обсягів пам'яті для збереження інформації про втручання, складністю створення або вибору необхідних детекторів для виявлення, значними складностями і витратами часу на обчислення, що є перешкодою для виявлення втручань у реальному часі, високим рівнем позитивних помилок і низьким рівнем виявлення.

В нових апаратних засобах та програмних продуктах у галузі захисту інформації з обмеженим доступом в останні роки спостерігається тенденція збільшення масштабу використання новітніх технологій, зокрема - штучного інтелекту, нанотехнологій, продукційних і експертних систему та ін., тобто так званих інтелектуальних систем. Цьому сприяє наявність у них можливості навчання, а також збільшення числа й ускладнення мережних загроз інформаційній безпеці.

Іншою тенденцією є спрямованість на інтеграцію засобів захисту різних рівнів (наприклад, персональний антивірус і мережевий екран рівня підприємства) з використанням засобів інтелектуальних систем.

Більш детальну інформацію стосовно останніх тенденцій застосування новітніх цифрових розробок у системах захисту інформації можна одержати в наступних джерелах [4,16,19, 39, 42-46 та ін].

Таким чином, можна зробити висновок, що розглянуті в останньому розділі навчальний посібника підходи до проектування цифрових компонентів систем захисту інформації з обмеженим доступом на сьогоднішній день далеко не вичерпали свій потенціал. Існує висока вірогідність того, що подальші дослідження відкриють нові шляхи розробки та впровадження методів інтелектуального захисту у сферу інформаційної безпеки.

12.3. Програмовані логічні інтегральні схеми

Програмована логіка є однією з найбільш динамічних галузей ринку електроніки, зокрема цифрових систем захисту інформації.

Програмована логічна інтегральна схема, ПЛІС (*programmable logic device, PLD*) - електронний компонент, що використовується для створення цифрових інтегральних схем. На відміну від звичайних цифрових мікросхем, логіка роботи ПЛІС не визначається при виготовленні, а задається за допомогою програмування. Для програмування можуть бути використані спеціальні пристрої - програматори, або налагоджувальні середовища, наприклад, системи автоматизованого проектування MAX+PLUS II, Quartus II, Xilinx Foundation Series та ін.

САПР дозволяють задати бажану структуру цифрового пристрою у вигляді принципової електричної схеми або програми на спеціальних мовах опису апаратури Verilog, VHDL, AHDL та ін.

За статистичними даними провідних фірм Aldec Inc., США та Xilinx Inc., США обсяг виробництва програмованих логічних інтегральних мікросхем (ІМС) з 1990 по 2010 роки зріс у понад 20 разів і продовжує зростати.

Через невелику швидкодію й малу кількість еквівалентних логічних вентилів ПЛІС довго займали досить скромну нішу на ринку електронних компонентів. З появою сучасних швидкодіючих ПЛІС надвисокої інтеграції, що працюють на високих тактових частотах, їх вплив на світовий ринок значно розширився.

Сфера застосування ПЛІС постійно розширюється. Діаграма розподілу обсягів споживання ПЛІС в окремих галузях наведена на рис. 12.8. Найбільшим споживачем ПЛІС є галузь телекомунікацій і зв'язку (41 % від всього обсягу виробництва). Друге місце посідає галузь комп'ютерних мереж, що використовує 26 % обсягу виробництва ПЛІС. Крім того, програмовані логічні ІМС застосовуються в галузі цифрової обробки даних (19 %) та в індустріальному виробництві (14 %). Серед споживачів цих мікросхем можна назвати такі відомі фірми і концерни, як Alcatel, Aser, Asus, IBM, Lockheed, Hewlett Packard, Fujitsu, Hitachi, Silicon Graphics, Texas Instruments, Motorola, Rockwell, Kodak та багато інших.

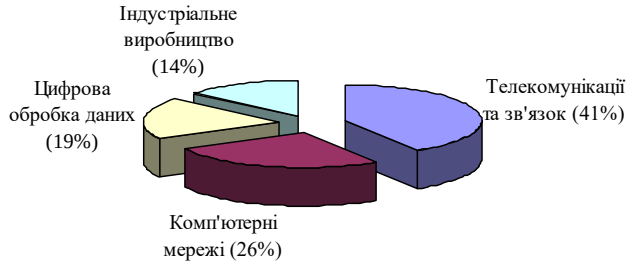


Рис. 12.8. Розподіл обсягів споживання ПЛІС в окремих галузях

ПЛІС діляться на наступні класи, див. рис. 12.9:

- стандартні (*Standard Programmable Logic Devices - SPLD*) або класичні;
- складні (*Complex PLD*);
- решітки програмованих елементів (*Field Programmable Gate Array -FPGA*);
- програмовані матриці (PLM).

У схемах PLA (*Programmable Logic Array*) обидві логічні матриці є програмованими, що робить їх найбільш гнучкими серед інших ПЛІС.

В PAL (*Programmable Array Logic*) програмованою є логічна матриця «І», а матриця «АБО» є фіксованою. Ці пристрої поєднують гнучкість, властиву PLA, із швидкодією PROM.

У PROM (*Programmable Read Only Memory*) логічна матриця «І» є фіксованою, а матриця «АБО» може бути програмованою.

До основних характеристик ПЛІС належать такі [30, 50]:

- кількість системних вентилів;
- швидкодія;
- системна тактова частота;
- максимальне число повторів вводу-виводу;
- енергоспоживання.

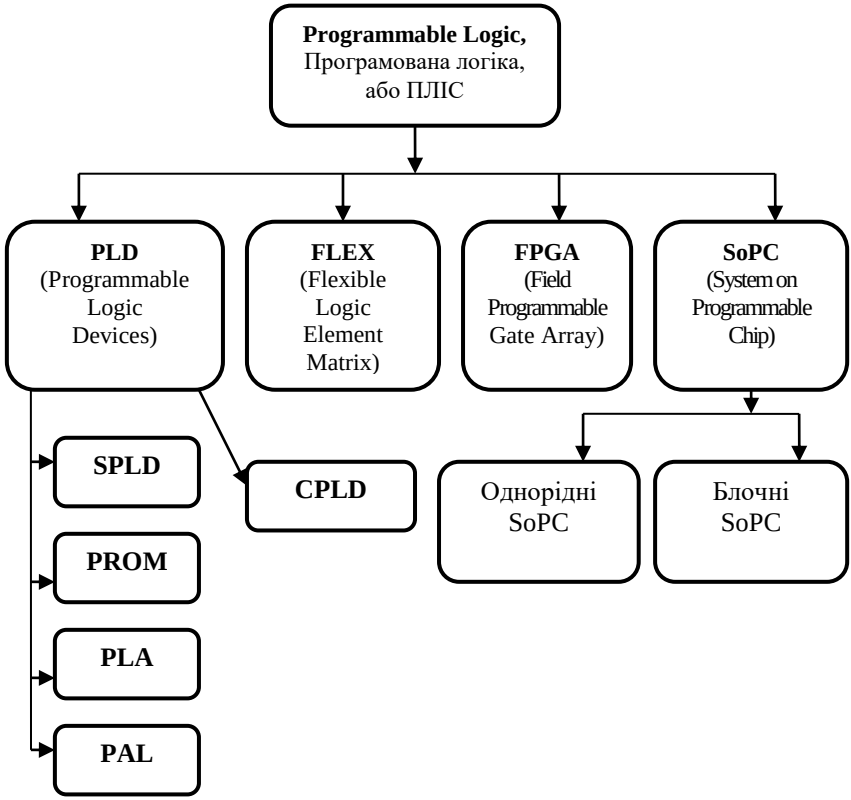


Рис. 12.9. Класифікація пристроїв із програмованою логікою

На рис. 12.10 показані умовні позначення стандартного логічного елементу «І» та його еквівалент у *PLD*.

Одиночна лінія на вході «І» використовується для представлення декількох входів. Вертикальні лінії відповідають сигналам (аргументами) *A*, *B* і *C*. Точка на перетині ліній позначає програмоване з'єднання між вхідними сигналами *A*, *B* і *C* із входом елемента «І». Програмування з'єднання відбувається за допомогою спеціальної технології.

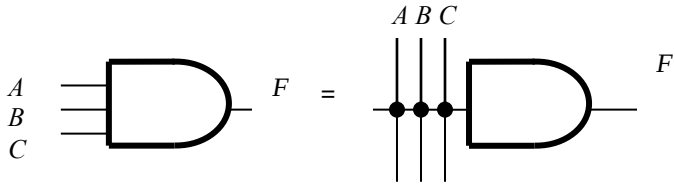


Рис. 12.10 Умовні позначення стандартного логічного елемента «І» та його еквівалент у *PLD*

Ще один приклад (рис. 12.11) ілюструє реалізацію логічної функції трьох аргументів у ПЛІС (*PLD*).

$$F = A \cdot \overline{B} + \overline{B} \cdot C + \overline{A} \cdot B \cdot \overline{C}.$$

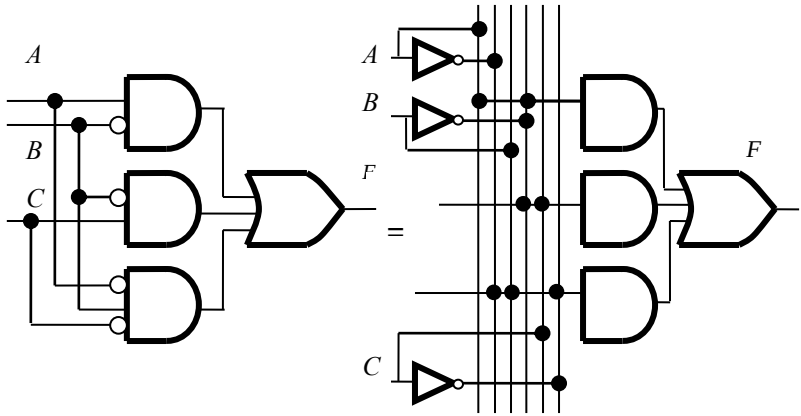


Рис. 12.11. Приклад реалізації логічної функції у *PLD*

Сьогодні ПЛІС проектують і виробляють кілька десятків провідних фірм. Лідерами у цій галузі є фірми Xilinx (створена у США у 1984 р., займає приблизно 60% ринку), Altera (створена у США у 1983 р., 19%) та Actel (створена у США у 1988 р., 15%).

До перших ПЛІС, які з'явилися на початку 70-х років, відносяться програмовані постійні запам'ятовуючі пристрої (ППЗП - Programmable Read Only Memory - *PROM*). Спочатку PROM використовували винятково для зберігання інформації, пізніше їх стали використовувати для реалізації логічних функцій.

Структура *PROM* містить дві матриці: матрицю DC, яка реалізує функції повного дешифратора, і програмовану матриця «АБО» («OR»). Конструкція *PROM* дозволяє реалізувати логічні функції, представлені в довершеній диз'юнктивній нормальній формі.

З 1971 р. стали випускатися програмовані логічні матриці (ПЛМ – Programmable Logic Array – PLA), які містять дві програмовані матриці (рис. 12.12), одна з яких реалізує функції «І» («AND»), а інша - функції «АБО» («OR»).

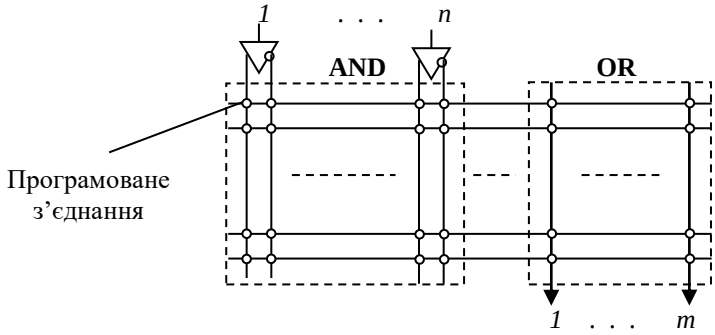


Рис. 12.12. Структура ПЛМ – Programmable Logic Array (PLA)

Приклад реалізації структури *PLA* для логічної функції чотирьох аргументів наведено на рис. 12.13.

Удосконалювання структури *PLA* призвело до створення програмованих матриць логіки (ПМЛ - Programmable Array Logic - *PAL*), у яких, на відміну від *PLA*, програмується тільки матриця «І», а матриця «АБО» має попередньо налагоджену та фіксовану структуру, при якій q проміжних шин зв'язується з одним виходом (рис. 12.14). Це дозволяє реалізувати матрицю «АБО» у вигляді сукупності q-входових диз'юнкторів. Вихідні буфери, що визначають архітектуру *PAL*, є

програмованими макроосередками, які можуть включати інвертор із трьома станами, тригери різних типів, елементи «виключне АБО» тощо..

Удосконалювання технології виробництва ПЛІС призвело до можливості реалізації на одному кристалі декількох *PAL*, що поєднуються у єдину конструкцію програмованими з'єднаннями. Такі ПЛІС одержали назву складних ПЛП (*Complex Programmable Logic Devices, CPLD*).

Загальна структура *CPLD* містить матрицю функціональних блоків *FB* і програмовану матрицю перемикань (*Switch Matrix, SM*). Основні логічні перетворення виконуються в *PAL*-блоках, а матриця перемикань служить лише для передачі сигналів між ними. Також у структурі *CPLD* присутні спеціалізовані входи, зв'язані як з матрицею перемикань, так і з усіма *PAL*-блоками. Ці входи звичайно використовують для передачі глобальних сигналів синхронізації і керування пристроєм.

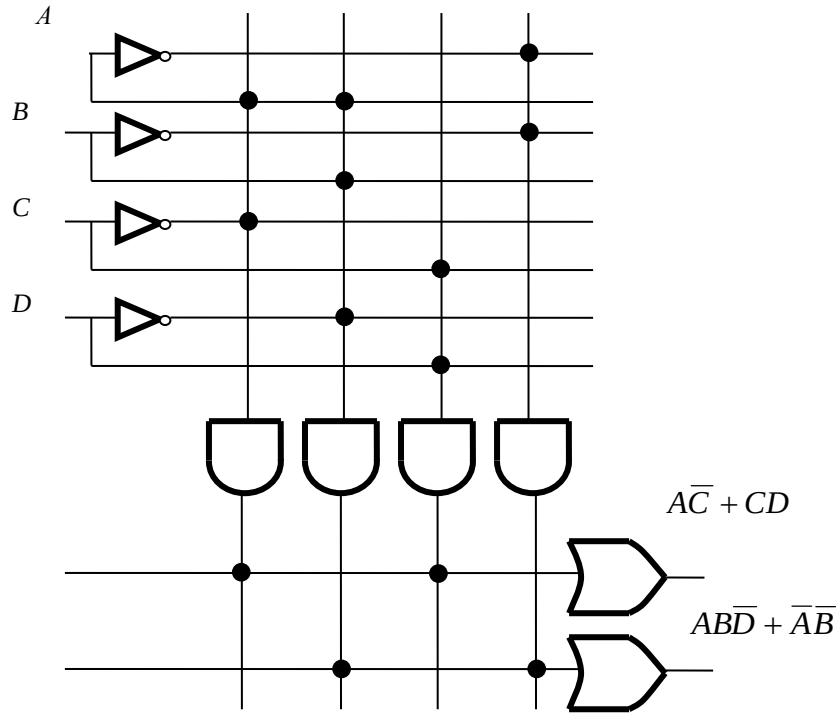


Рис. 12.13. Приклад структури ПЛМ (*PLA*)

Приклад реалізації структури PAL для функції чотирьох аргументів наведено на рис. 12.15.

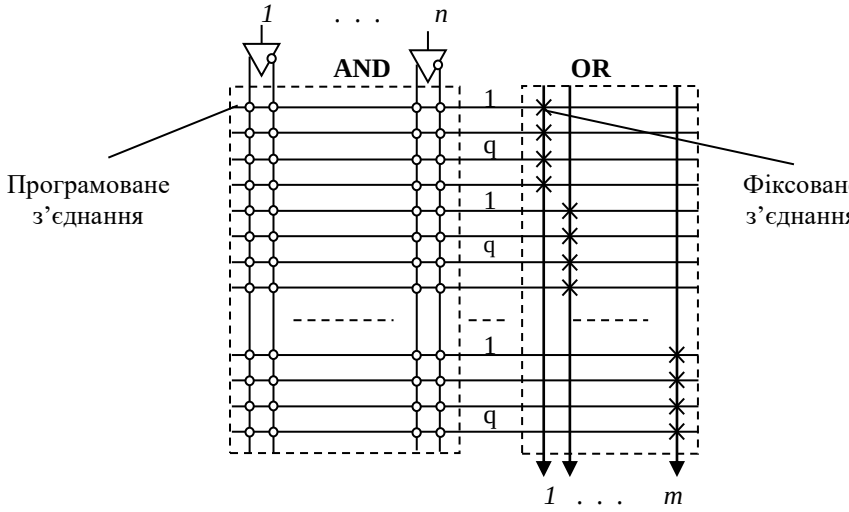


Рис. 12.14. Структура ПМЛ – Programmable Array Logic (PAL)

Схеми PAL дуже зручні для проектування комбінаційної логіки. Але вони не можуть бути використані для проектування схем без застосування зовнішніх тригерів. Тому в конструкції *PLD* (простих *PLD - simple PLD*) додають тригери, як це показано на рис. 12.16. За допомогою мультиплексорів здійснюється вибір виходу.

Суть проектування ПЛІС полягає в розробці спеціальної програми (у вигляді двійкового потоку "bit-stream"), при завантаженні якої в ПЛІС остання виконує функції спроектованого електронного пристрою.

Перший підхід до розробки ПЛІС ґрунтувався на проектуванні за допомогою булевих рівнянь. Такий спосіб проектування передбачає формування булевих рівнянь чи таблиць істинності для кожного тригера або блоку вентилів.

Опис проектних схем за допомогою булевих рівнянь є прийнятним для малих проектів, що включають десятки або навіть сотні вентилів, але для схем, що базуються на тисячах вентилів, такий опис стає надто громіздким і неефективним [30, 50].

Крім того, схема, що формалізована за допомогою булевих рівнянь, важко піддається декомпозиції.

У традиційній формі схемне проектування є модифікацією методу проектування булевими рівняннями. Цей підхід ґрунтується на застосуванні попередньо синтезованих логічних блоків, таких, як регістри,

лічильники, дешифратори та ін. Довгий час метод схемного проектування вважався найкращим вибором, однак із розвитком елементної бази мікросхемотехніки кількість елементарних схем у нових ПЛІС постійно зростала.

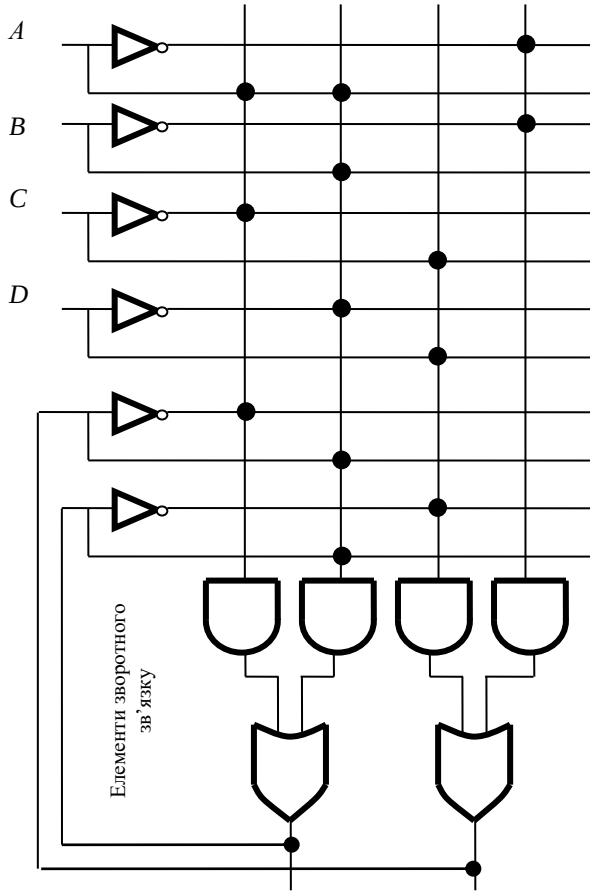


Рис. 12.15. Приклад реалізації структури *PAL* для функції чотирьох аргументів

Сучасний підхід до проектування ПЛІС ґрунтується на застосуванні спеціалізованих мов опису обладнання (*HDL - Hardware Description Language*). Ці мови, крім звичних конструкцій, таких як розгалуження, цикли, підпрограми та ін., мають також спеціалізовані засоби - робота з

портами, забезпечення паралельності виконання процесів тощо. Найбільш поширеними серед HDL стали мови ABEL, Verilog та VHDL [30, 50].

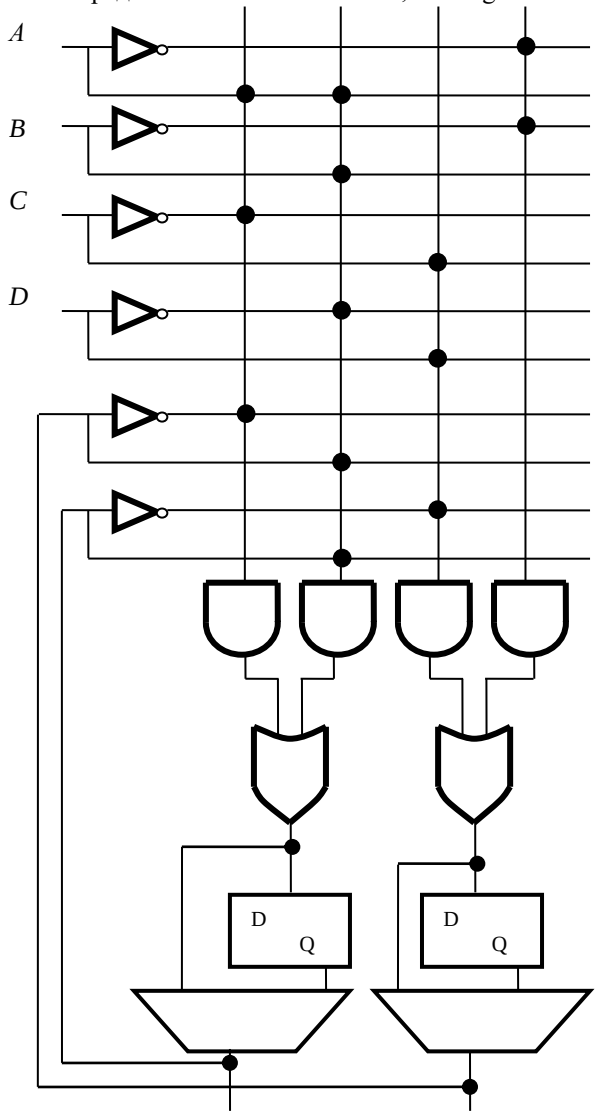


Рис. 12.15. Структура *SPLD*

ABEL (*Advanced Boolean Equation Language* - розширена мова логічних рівнянь) є промисловим стандартом, розробленим Data I/O Corp.

для програмованих логічних пристроїв. ABEL може застосовуватися для опису поведінки систем (за допомогою С-подібних операторів) у різних формах: на основі логічних рівнянь, таблиць істинності, діаграм стану. Порівняно з ABEL мови VHDL та Verilog є складнішими, але більш придатними для опису великих систем. Вони майже рівні за своїми технічними можливостями. В Європі та США ширше застосовується VHDL, в країнах Азії - Verilog. VHDL базується на мові високого рівня Ada. З цієї мови розробниками VHDL було запозичені синтаксис та основні структури.

Наприклад, синтаксис операторів у Verilog подібний до синтаксису мови програмування С++. Існують арифметичні (+, -, *, /), логічні (&& - logical **AND**; || - logical **OR**; = - logical **EQUALITY**; != - logical **INEQUALITY**) оператори, корисні для моделювання цифрових схем. При однаковому синтаксисі (\ ~| ^ ~^ & ~&) дані оператори можуть бути бітовими (*bitwize*) і працювати з двома операндами або операторами згортки (*reduction*) і працювати з одним операндом. Тип операції визначається по місту оператора у виразі.

У якості приклада в таблиці 12.1 показані варіанти опису стандартного **D** – тригера на мовах VHDL та Verilog. Більш детальному опису цих мов присвячено багато спеціальної літератури [50, 51, 52].

Таблиця 12.1

Приклади опису D – тригера на мовах VHDL та Verilog

VHDL	Verilog

library IEEE;
use IEEE.std_logic_1164.all;
entity dff is
port (data, clk : in std_logic;
q :out std_logic);
end dff;
architecture behav of dff is
begin
process (clk) begin
if (clk'event and clk = '1') then
q <= data;
end if;
end process;
end behav;
	module dff (data, clk, q);
	input data, clk;
	output q;
	reg q;
	always @(posedge clk)
	q = data;
	endmodule

Відповідно до сучасних вимог системи автоматизованого проектування ПЛІС повинні забезпечувати:

• реалізацію однієї чи більше HDL-мов з можливістю введення, редагування та відлагодження початкового тексту програм;

• реалізацію засобів графічного введення проектної схеми, наприклад, за допомогою редактора скінчених автоматів та засобів компіляції графічного представлення в HDL-код;

• реалізацію засобів моделювання поведінки описаного об'єкта;

• реалізацію засобів синтезу бітового потоку з підтримкою широкого класу серій ІМС;

• реалізацію засобів моделювання об'єкта на рівні вентилів;

• реалізацію засобів програмування ІМС.

Розробкою такого програмного забезпечення займається значна кількість фірм, серед яких можна назвати Aldec, Altera, Xilinx, Viewlogic, Mentor, Synopsis, Vantage та ін.

На рис. 12.16, 12.17 та 12.18 відповідно, показані вікна програмних пакетів для синтезу мікросхем ПЛІС - Active-HDL 8.2 (фірма Aldec), Quartus II 9.1 (Altera) та Xilinx ISE 10.1 (Xilinx).

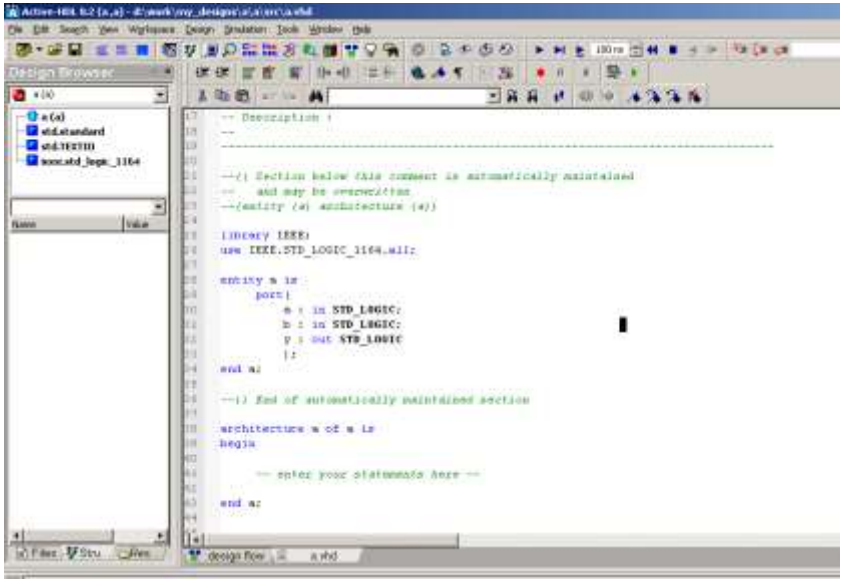


Рис. 12.18. Загальний вигляд пакета Active-HDL 8.2 (фірма Aldec)

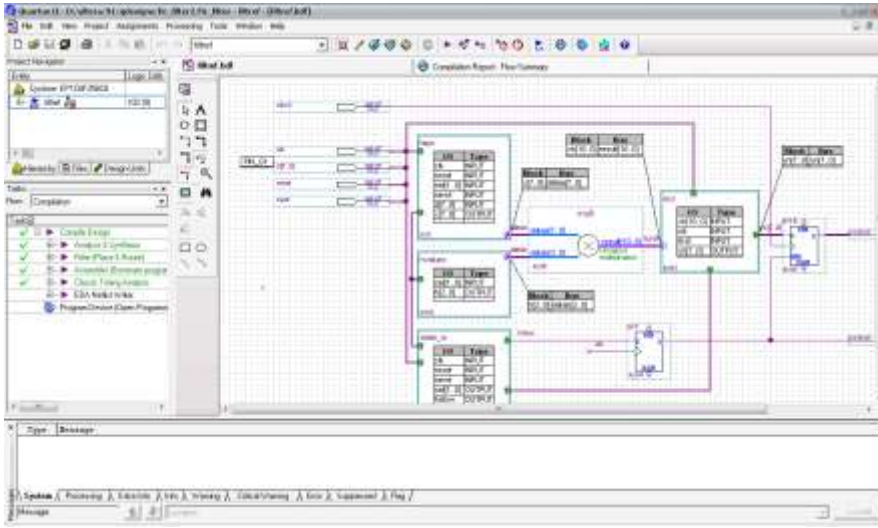


Рис. 12.18. Загальний вигляд пакета Quartus II 9.1 (Altera)

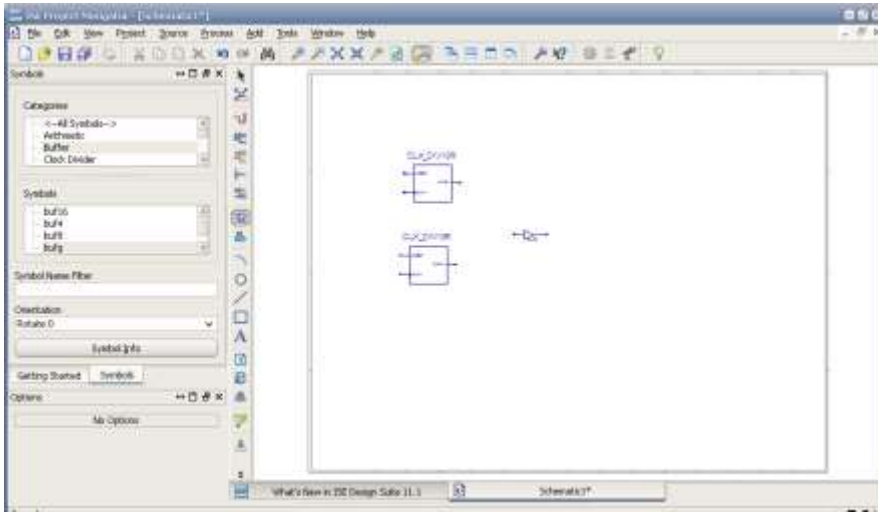


Рис. 12.19. Загальний вигляд пакета Xilinx ISE 10.1 (Xilinx)

Основним обмеженням цих програмних пакетів є те, що вони не мають засобів синтезу для мікросхем інших виробників ПЛІС.

Крім того, розроблено ряд програмних продуктів, що реалізують ту чи іншу частину загальних вимог до САПР ПЛІС. Застосування цих продуктів вимагає в кожному конкретному випадку розв'язання проблем сумісності між пакетами, тому перевагу слід надавати пакетам, що реалізують процес проектування ПЛІС.

Серед номенклатури ПЛІС найбільшого поширення набули мікросхеми, що виготовлені за технологіями *FPGA* та *CPLD*. Абревіатура FPGA розшифровується як *Field Programmable Gate Array* - програмована користувачем вентильна матриця. Загальну структуру кристала FPGA-мікросхеми наведено на рис. 12.20.

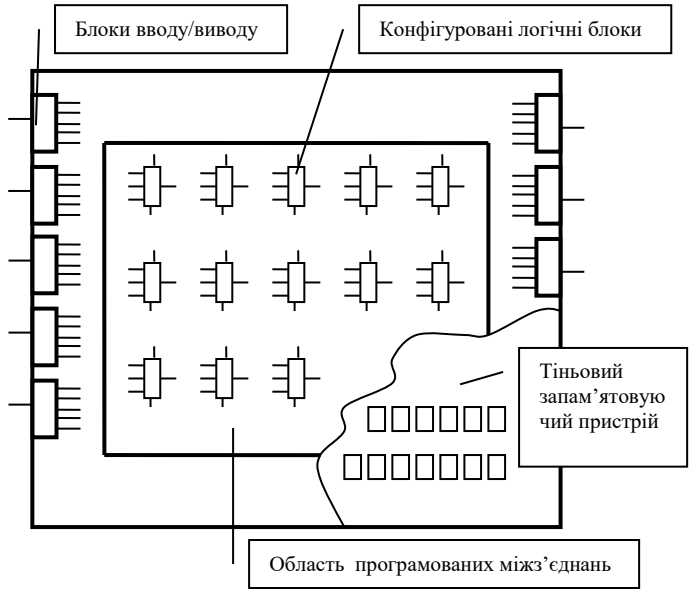


Рис. 12.20. Загальна структура кристала *FPGA*-мікросхеми

По периферії верхнього шару кристала розміщуються блоки вводу/виводу, що можуть бути запрограмовані для виконання функцій буферів:

- вхідного;
- вихідного;

- запам'ятовуванням;
- інші.

У деяких серіях FPGA-IMC рівень напруги на блоках вводу/виводу може різнитися, що дає змогу зв'язувати різні за живленням інтерфейси FPGA. У центрі кристала у вигляді матриці розміщено конфігуровані логічні блоки. Швидкодія мікросхем визначається часовою затримкою "вхід-вихід" одного конфігурованого логічного блоку.

Конфігуровані логічні блоки FPGA-IMC можуть генерувати будь-яку логічну функцію чотирьох аргументів або дві логічні функції трьох аргументів. Аргументи для логічних функцій можуть надходити у вигляді сигналів з чотирьох входів та виходу запам'ятовуючого елемента. Область між конфігурованими логічними блоками називається областю програмованих з'єднань і представляє собою розвинену ієрархію ліній зв'язку, в місцях перетину яких розміщено спеціальні швидкодіючі транзистори.

Функція області міжз'єднань полягає в забезпеченні зв'язку між будь-якими виводами конфігурованих логічних блоків та блоками вводу/виводу.

Нижній шар кристала займає тіньовий запам'ятовувальний пристрій, інформація в елементах якого і визначає логічні функції конфігурованих логічних блоків, конфігурацію блоку вводу/виводу та маршрути міжз'єднань.

Широкий діапазон FPGA-технології дозволяє проектувати на їх основі широкий спектр електронних пристроїв, серед яких: мікропрограмні пристрої керування, скінченні автомати, універсальні та спеціалізовані процесори, пристрої цифрової обробки сигналів, засоби поєднання різних за живленням інтерфейсів, перетворювачі кодів, периферійні контролери, тощо.

Архітектура ПЛІС типу *CPLD*, зображена на рис. 12.17., ПЛІС типу *CPLD* - XC9500 має три групи виводів:

- виводи JTAG-порта для програмування та периферійного сканування;
- порти вводу/виводу;
- керуючі виводи: сигнал тактування - GCK, установлення/скидання - GSR, керування третім станом - GTS.

Блоки вводу/виводу забезпечують буферизацію всіх входів та виходів ПЛІС. Кожен функціональний блок (ФБ) містить 18 макрокомірок зі структурою «36 входів - 1 вихід» і дозволяє формувати 18 логічних функцій для будь-якої комбінації з 36 аргументів. Матриця перемикань забезпечує подавання будь-яких вхідних сигналів та вихідних сигналів ФБ на входи ФБ, а також подавання вихідних сигналів ФБ на блоки вводу/виводу.

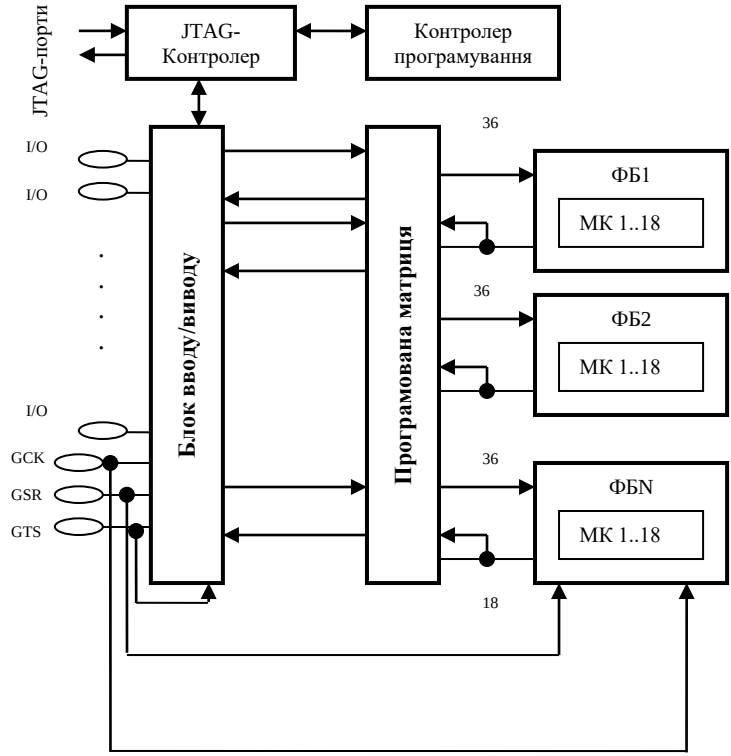


Рис. 12.17. Архітектура ПЛІС типу *CPLD* - XC9500

CPLD-технології широко застосовуються для проектування нестандартних арифметико-логічних пристроїв, дешифраторів, мультиплексорів та ін.

У порівнянні з *FPGA*, до недоліків *CPLD* слід віднести: малу кількість системних вентилів та значне енергоспоживання. Переваги технології *CPLD* полягають у вищій швидкодії та забезпеченні можливості встановлення захисту від копіювання. Важливою перевагою також є те, що програмні засоби для розробки та синтезу систем на базі *CPLD* поширюються вільно.

Основу функціонування сучасних цифрових систем технічних комплексів захисту інформації з обмеженим доступом складає принцип мікропрограмного керування, відповідно до якого координування взаємодії всіх блоків системи виконує керуючий автомат. На практиці широко застосовується реалізація керуючих автоматів у вигляді автомата з «жорсткою» логікою, тобто автомати Мілі і Мура. Вихідними даними для синтезу схеми керуючого автомата служать алгоритми керування операційним пристроєм, задані, як правило, у вигляді граф-схем. Зараз для реалізації схем керуючих автоматів використовуються різноманітні програмовані логічні пристрої, які дозволяють суттєво підвищити швидкодію, надійність і компактність цифрових систем.

Сьогодні для реалізації логічних схем цифрових автоматів широко використовуються програмувальні логічні пристрої - *PLA, PAL, PLD, PROM*, і схеми середнього ступеня інтеграції – дешифратори та мультиплексори. Одним із важливих завдань, що виникають при синтезі керуючих автоматів, є оптимізація апаратурних витрат у логічній схемі.

Під областю ефективної реалізації структури керуючого автомата розуміється таке сполучення характеристик ГСА, обумовлених її розмірністю і конфігурацією, при якому реалізація автомата з даною структурою забезпечує найменші апаратурні витрати в його логічній схемі.

Сучасні мікросхеми ПЛІС, виконані за 0,22-мікронної технології, здатні працювати на частотах до 300 Мгц і реалізують до 3 млн. еквівалентних логічних вентилів [30, 44]. Різке збільшення потужності сучасних ПЛІС створило передумови для реалізації не тільки простих контролерів та інтерфейсних вузлів, але й систем цифрової обробки сигналів, пристроїв управління в реальному часі найскладнішими технологічними процесами, інтелектуальних контролерів і нейронних мереж.

Поява ПЛІС із наднизьким рівнем енергоспоживання відкриває широкі можливості для їх використання в мобільних мережах та портативних комп'ютерах.

Недоліком ПЛІС є їх енергозалежність. Програма зазвичай зберігається в енергозалежній пам'яті, при кожному включенні живлення мікросхеми необхідно заново конфігурувати її. Такі мікросхеми виробляють фірми Xilinx й Altera.

Завантаження конфігурації відбувається за допомогою завантажувача, що може бути вбудований і в саму *FPGA*. Фірми-виробники пропонують конфігураційні ПЗУ, що при включенні живлення завантажує конфігурацію в ПЛІС. Фірми Actel й Lattice Semiconductor пропонують мікросхеми *FPGA*, в яких конфігурація зберігається в

енергонезалежній Flash-пам'яті і в яких ПЛІС програма зберігається при зникненні електроживлення.

Альтернативою ПЛІС є: програмовані логічні контролери; базові матричні кристали, що вимагають заводського виробничого процесу для програмування; ASIC - спеціалізовані замовні великі інтегральні схеми, які при багатосерійному та одиничному виробництві істотно дорожчі; спеціалізовані комп'ютери або спеціалізовані процесори (наприклад, цифровий сигнальний процесор).

Сучасний рівень розвитку ПЛІС, їх швидкодія та ємність, дозволяють реалізувати системи на одному кристалі, що дає більший виграш у продуктивності, ніж за використання класичних мікроконтролерів та ВІС, а також дозволяє прискорити швидкість процесів передачі даних, що, зокрема, є критичним для різноманітних сучасних цифрових систем захисту інформації. Застосування ПЛІС обумовлює простоту розробки та налагодження системи, а також дозволяє адаптувати систему під визначений об'єкт управління з максимальною продуктивністю та швидкодією.

Запитання для самоперевірки

1. Які останні тенденції розвитку та використовування цифрових технологій у системах інформаційної безпеки?
2. Які завдання виконують типові агенти у багатоагентних системах захисту інформації з обмеженим доступом?
3. Які переваги мають системи штучного інтелекту в системі захисту інформації з обмеженим доступом?
4. Яким чином можна використовувати положення теорії автоматів при розробці типових агентів систем інформаційної безпеки?
5. На яких постулатах базується поняття штучного інтелекту для систем захисту інформації з обмеженим доступом?
6. Дайте визначення ПЛІС.
7. На які класи поділяють ПЛІС?
8. У яких галузях використовуються ПЛІС?
9. Які програмні пакети використовують для розробки ПЛІС?
10. Які мови програмування використовують для розробки пристроїв із програмованою логікою?
11. Які фірми є лідерами на ринку програмованих логічних пристроїв?

12. Які переваги та недоліки мають програмовані логічні пристрої?

13. Яким чином виконується програмування з’єднань у матрицях ПЛІС.

14. Як розшифровуються абревіатури *PLA*, *PAL*, *PLD*, *PROM*?

Література для самостійної підготовки за темою:

4, 16, 41, 44, 48, 50-52.

В И С Н О В К И

Розвиток цифрової технології протягом останніх десятиліть відбувався феноменальними темпами. Кількість галузей застосування цифрових систем продовжує неухильно зростати. Так, наприклад, за останні декілька років цифрові автомати активно впроваджуються у системи інформаційної безпеки державних і комерційних організацій. Сьогодні можна використовувати голосові команди для відсилки й отримання електронної пошти, відправки факсу, переключення радіостанції, зміни компакт-дисків, виклику інформації про технічний стан автомобіля тощо. Уже через кілька років ваші кишенькові інформаційні системи стануть більш продуктивними, ніж сучасні офісні інформаційні системи. Мобільні телефони зможуть отримувати й сортувати вхідні дзвінки та відповідати на них, а також дадуть можливість переглядати телевізійні передачі й відео, коли абоненти перебувають на величезних відстанях один від одного.

Інакше кажучи, цифрова техніка й технологія не лише продовжать свою тріумфальну ходу планетою, але й дозволять реалізувати такі технічні нововведення, про які ми лише мріємо. Тому зараз наше завдання – дізнатися якомога більше про основні принципи побудови цифрових схем, розібратися з теоремами й правилами алгебри логіки, системами числення, логічними елементами, для того, щоб бути готовими реалізувати нові технології в реальних цифрових автоматах, зокрема для телекомунікаційних систем та комплексів систем захисту інформації з обмеженим доступом.

ЛІТЕРАТУРА

1. Брауэр В. Введение в теорию конечных автоматов / Брауэр В.; [Пер. с нем.]. – М.: Радио и связь, 1987. – 392 с.
2. Будинский Я. Логические цепи в цифровой технике / Будинский Я. – М.: Связь, 1977. – 164 с.
3. Вавилов Е.Н. Синтез схем электронных цифровых машин / Вавилов Е.Н., Портной Г.П. – М.: Советское радио, 1963. – 176 с.
4. Вєртузаєв М.С. Захист інформації в комп'ютерних системах від несанкціонованого доступу : навч. посібник [за ред. С.Г. Лаптева] / Вєртузаєв М.С., Юрченко О.М. – К.: Видавництво Європейського університету, 2001. – 201 с.
5. Гилмор Ч. Введение в микропроцессорную технику / Гилмор Ч.; [Пер. с анг.]. – М.: Мир, 2001. – 380 с.
6. Глушков В.М. Синтез цифровых автоматов / Глушков В.М. – М.: Физматгиз, 1964. – 240 с.
7. Горбатов В.А. Теория автоматов: Учебник для студентов втузов / Горбатов В.А., Горбатова М.В., Горбатов А.В. – М.: Астрель АСТ, 2008. – 560 с.
8. Комп'ютерна логіка: [навч. посібник] / Жабін В.І., Жуков І.А., Клименко І.А., Ткаченко В.В. – К.: Книжкове вид-во НАУ, 2007. – 364 с.
9. Жабін В.І. Цифрові автомати. Практикум / Жабін В.І. – К.: ВЕК +, 2004. – 160 с.
10. Жмакин А.П. Архитектура ЭВМ / Жмакин А.П. – СПб.: БХВ - Петербург, 2006. – 320 с.
11. Про захист інформації в інформаційно-телекомунікаційних системах / Відомості Верховної Ради України (ВВР), 1994, N 31, ст. 286. Із змінами, внесеними згідно із Законами N 879-VI від 15.01.2009, N 1180-VI від 19.03.2009 – (Закон України).
12. Ильин В.А. Телеуправление и телеизмерение / Ильин В.А. – М.: Энергоиздат, 1982. – 186 с.
13. Каган Б.М. Электронные вычислительные машины и системы / Каган Б.М. – М.: Энергоатомиздат, 1991. – 212 с.
14. Карпов Ю.Г. Теория автоматов / Карпов Ю.Г. – СПб.: Питер, 2002. – 224 с.
15. Карцев М.А. Арифметика цифровых машин / Карцев М.А. – М.: Наука, 1969. – 134 с.
16. Комашинский Д.А. Нейронные сети и их применение в системах управления и святи / Комашинский Д.А. Смирнов В.И. – М.: Издательство: Горячая Линия - Телеком, 2002. – 96 с.

17. Концепція розвитку телекомунікацій в Україні до 2010 року / Розпорядження Кабінету Міністрів України від 7 червня 2006 р. № 316-р.
18. Лазарев В.Г., Пийль Е.И. Синтез управляющих автоматов / Лазарев В.Г., Пийль Е.И. – М.: Энергия, 1978. – 246 с.
19. Ленков С.В. Методы и средства защиты информации / [Ленков С.В., Перегудов Д.А., Хорошко В.А.] ; под ред. В.А.Хорошко. – [в 2-х томах]. – К.: Арий, 2008. – 464 с. – (Несанкционированное получение информации; т. 1).
20. Майоров С.А. Электронные вычислительные машины (справочник по конструированию) / Майоров С.А., Крутовских С.А. – М.: Связь, 1975. – 488 с.
21. Майоров С.А. Введение в микроЭВМ / Майоров С.А., Кириллов В.В., Приблуда А.А. – М.:Машиностроение, 1988. – 304 с.
22. Поспелов Д.А. Логические методы анализа и синтеза схем / Поспелов Д.А. – [2-е издание переработанное и дополненное]. – М.: Энергия, 1974. – 198 с.
23. Поспелов Д.А. Арифметические основы вычислительных машин дискретного действия / Поспелов Д.А. – М.: Высшая школа, 1970. – 210 с.
24. Савельев А.Я. Арифметические и логические основы цифровых автоматов / Савельев А.Я. – М.: Высш. шк., 1980. – 312 с.
25. Савельев А.Я. Прикладная теория цифровых автоматов / Савельев А.Я. – М.: Высш. шк., 1987. – 272 с.
26. Вдосконалення генераторів ПВП та їх застосування в системах скремблер-дескремблер телекомунікаційних пристроїв / Савченко Ю.Г., Малогулко Л.В. – 2008. – с. 34-38. – Наукові записки УНДІЗ, №6 (8).
27. Прикладная теория цифровых автоматов / [Самофалов К.Г., Романкевич А.М. и др.]. – К.: Вища шк., 1987. – 369 с.
28. Семенець В.В. Проектування цифрових систем з використанням мови VHDL / Семенець В.В., Хаханова І.В., Хаханов В.І. – Харків: ХНУРЕ, 2003. – 492 с.
29. Темников Ф.Е. Теоретические основы информационной техники / Темников Ф.Е., Афонин В.А., Дмитриев В.И. – М.: Энергия, 1971. – 198 с.
30. Точи Р. Цифровые системы. Теория и практика / Точи Р., Уидмер Дж., Нил С.; [Пер. с анг.].– М.: Вильямс, 2004. – 8-е изд. – 1024 с.
31. Трахтенброт Б.А. Конечные автоматы / Трахтенброт Б.А., Барздинь Я.М. – М., «Наука», 1970. – 124 с.
32. Тьюринг А. Может ли машина мыслить? / Тьюринг А. – М.:Физматгиз, 1960. – 112 с.

33. Филиппов А.Г. Проектирование логических узлов ЭВМ / Филиппов А.Г., Белкин О.С. – М.: Советское радио, 1974. – 144 с.
34. Чу Я. Организация ЭВМ и микропрограммирование / Чу Я. – М.: Мир. 1975. – 234 с.
35. Шауман А.М. Основы машинной арифметики / Шауман А.М. – Л.: Изд-во Ленингр. ун-та. 1979. – 146 с.
36. Ярлыков М.С. Применение марковских теорий нелинейной фильтрации в радиотехнике / Ярлыков М.С. – М.: Сов. радио, 1980. – 360 с.
37. Emmanuel C. Feather Digital Signal Processing / Emmanuel C. Feather, Barrie W. Jervis. – М.: Вильямс, 2007. – 1222 с.
38. Floyd L. Digital Fundamentals / Floyd L., Thomas. – Sixth Editional., 1997 by Prentice-Hall, Inc.
39. Moldovyan A.A. Flexible Block Ciphers with Provably Inequivalent Cryptalgorithm Modifications / Moldovyan A.A., Moldovyan N.A. – Criptologia. – 1998. – V.XXII. – N2. P.134-140.
40. Shannon C.E. Communication Theory of Secret System / Shannon C.E. – 1949.- V. 28.-P.656-715. – (Bell System Technical Journal).
41. Short K. Microprocessor And Programmed Logic. – 2-nd Ed.- Edlewnod Cliffc: Prentice-Hall, 1987. - 515 p.
42. Tarannikov Y. On Resilient Boolean Functions with Maximal Possible Nonlinearity. Departament Moscow State University.
43. Zeng Y., Mihaljevic M. Imai H. Cellular Automaton Based Fast One-way Hash Function Suitable for Hardware Implementation.Japan. Jokohama. Pre-Proceedings.-1200 p.
44. Zou C.C., Duffield N., Towsley D., Gong W. Adaptive Defense against Various Network Attacks // IEEE Journal on Selected Areas in Communications: High-Speed Network Security (J-SAC). 2006. Vol.24, №.10. pp. 44 -51.
45. Urchin V., Peng T., Leckie C., and Ramamohanarao K. Survey of Network- Based Defense Mechanisms Countering the DoS and DDoS Problems // ACM Computing Surveys. - April 2007. - Vol. 39, N 1. pp. 31-42.
46. Urgen M. W. Pan and S. Stolfo. Ensemble-based adaptive intrusion detection. In Proceeding of 2002 SIAM International Conference on Data Mining, Arlington, VA, 2002.
47. Niederreiter H. Knapsack-Type Cryptosystems and Algebraic Coding Theory / H. Niederreiter // Probl. Control and Inform. Theoty. – 1986. – V. 15. – P. 19-34.
48. McEliece R.J. A Public-Key Criptosystem Based on Algebraic Theory / R.J. McEliece // DGN Progres Report 42- 44, Jet Propulsi on Lab. Pasadena, CA. January – February, 1978. – P. 114-116.

49. Криптографічний захист інформації. Цифровий підпис, що ґрунтується на еліптичних кривих : ДСТУ 4145-2002. – [Чинний від 2003 – 07 – 01]. – К. : Державний комітет України з питань технічного регулювання та споживчої політики, 2003. – IV. 30 с. (Національний стандарт України).
50. Стандарт мови VHDL'99. [Електронний ресурс]. Режим доступа: http://vhdl-ams.org.
51. Приклади ПЛІС пристроїв на мові Verilog. [Електронний ресурс]. Режим доступа: http://vhdl-ams.org.
52. Мови VHDL і VERILOG у проектуванні цифрової апаратури. [Електронний ресурс]. Режим доступа: http://bukomp.if.ua/ samouchitel_wml_i_wmlscript_-aaa/yazyki_vhdl_ i_ verilog_v.php
53. NESSIE Call for Cryptographic Primitives, Version 2.2, 8th March 2000. [Електронний ресурс]. Режим доступа: http://cryptonessie.org
54. CRYPTREC. Technical report of Cryptography Research and Evaluation Committees. [Електронний ресурс]. Режим доступа: http://cryptrec.jp

ГЛОСАРІЙ

„АБО”, що виключає (XOR) – функція нерівнозначності, яка фактично реалізує процедуру сумування за модулем 2.

Абсолютна похибка представлення – різниця між істинним значенням вхідної величини та її значенням, отриманим з машинного зображення.

Автомат Мілі – синхронний автомат, у якого вихідні сигнали залежать і від стану автомата, і від значення вхідного сигналу.

Автомат Мура – синхронний автомат, вихідні сигнали якого в момент часу *t* однозначно визначаються станом автомата в цей же момент часу й у явному вигляді не залежать від значень вхідних сигналів.

Алгебра логіки – алгебра висловлювань (є логічною основою комп’ютерів).

Алгоритм – скінченна сукупність точно сформульованих правил розв’язання якоїсь задачі або ж точно описана послідовність виконання деяких елементарних процедур, необхідна для отримання певного результату.

Асинхронний автомат – автомат, у якого інтервал часу, протягом якого залишається незмінним стан вхідних сигналів, є величиною перемінною й визначається часом, необхідним для установки відповідних вихідних сигналів і завершення переходу в новий стан.

Базис – функціонально повна система елементарних функцій, за допомогою якої будь-яка логічна функція може бути представлена суперпозицією вихідних елементарних функцій.

Байт – вісім бітів.

Біт – один розряд (одна позиція) у двійковій системі числення.

Біт парності – контрольний біт, що додається до масиву двійкових цифр так, щоб сума всіх розрядів масиву, включаючи біт парності, була завжди парною (або непарною).

Висловлювання – усяке твердження, яке може бути істинним або хибним. Істинному висловлюванню приписується символ 1, а хибному – 0.

Від’ємна логіка роботи логічної схеми – варіант роботи, при якому високому рівню сигналу в ЦА ставиться у відповідність логічний нуль, а низькому рівню – логічна одиниця.

Власна частина логічного добутку – результат виключення з певного добутку одного або кількох співмножників.

Геометричний (графічний) спосіб представлення логічної функції – спосіб представлення логічної функції, при якому, наприклад, функція двох аргументів представляється у вигляді квадрата, вершини якого відповідають *n* комбінаціям аргументів; трьох аргументів – у вигляді тривимірного куба; чотирьох аргументів – у вигляді чотиривимірного куба і т. д.

Граф автомата – графічна схема, що складається з вузлів, з'єднаних гілками. Вузли ототожнюють внутрішні стани автомата. Кожна гілка відзначається вхідним сигналом, який викликає в автоматі відповідний цій гілці перехід, і вихідним сигналом, який виникає при цьому переході.

Граф-схема алгоритму – алгоритм, описаний спеціальними графічними символами.

Диз'юнкція (логічне додавання, чи функція **„АБО" (OR)**) – функція, яка є істинною тоді, коли істинний хоча б один з її аргументів.

Діапазон представлення чисел у заданій системі числення – інтервал числової осі, уміщений між максимальним і мінімальним числами, значення яких залежить від довжини розрядної сітки, виділеної в машині для представлення чисел.

Довжина розрядної сітки – кількість розрядів (позицій), виділена в комп'ютері для представлення числа в позиційній системі числення (в основному в двійковій системі).

Довжина числа (слова) – кількість позицій (або розрядів) у записі числа.

Додатна логіка роботи логічної схеми – варіант роботи, при якому високому рівню сигналу ставиться у відповідність логічна одиниця, а низькому рівню – логічний нуль.

Досконала (стандартна або канонічна) форма – форма аналітичного представлення ЛФ.

Елементарна сума – диз'юнкція аргументів, частина яких може мати заперечення, тобто фактично макстерм.

Елементарний добуток – кон'юнкція кількох аргументів або їх заперечень, тобто фактично мінтерм.

Імпліканта – деяка логічна функція, що входить у певну логічну функцію й обертається в нуль при наборі аргументів, на якому сама ЛФ також дорівнює нулю.

Імпліканта диз'юнктивна – будь-який мінтерм чи група мінтермів початкової НДФ.

Імпліканта кон'юнктивна – будь-який макстерм чи група макстермів початкової НКФ.

Імплікація, або функція „ЯКЩО-ТО" – функція, яка є хибною тоді й лише тоді коли x_1 істинний і x_2 хибний. Аргумент x_1 називається посилкою, а x_2 – наслідком.

Інверсія (заперечення) деякого логічного аргументу – логічна перемінна, яка приймає зворотне значення.

Інтернет – мережа, що поєднує по лініях зв'язку інформаційні системи в усьому світі.

Інтерфейс – сукупність апаратних і програмних засобів для організації інформаційного зв'язку між інформаційними пристроями, тобто пристроями, що здійснюють прийом, передачу, зберігання інформації та її обробку.

Інформація – відомості про певні властивості або параметри тих чи інших явищ або об'єктів і про залежності між цими властивостями.

Інформація з обмеженим доступом – інформація, доступ до якої обмежується у відповідності з законом або рішенням власника такої інформації чи уповноваженої ним особи. За своїм правовим режимом інформація з обмеженим доступом поділяється на конфіденційну та таємну.

Карти Карно – один із способів графічного представлення логічної функції. Використовуються в процедурах мінімізації ЛФ.

Код числа – це запис числа в деякій системі числення.

Комбінаційні схеми – логічні схеми, вихідний сигнал яких залежить лише від стану вхідних сигналів у кожен момент часу.

Комплексна система захисту інформації з обмеженим доступом – взаємопов'язана сукупність організаційних та інженерно-технічних заходів, засобів і методів захист інформації з обмеженим доступом.

Кон'юнкція (логічне множення або функція **„І" (AND)**) – це функція, яка є істинною тоді, коли всі її аргументи одночасно істинні.

Конституанта нуля – тотожна макстерму.

Конституанта одиниці – тотожна мінтерму.

Криптографічний захист інформації – вид захисту інформації, що реалізується шляхом перетворення інформації з використанням спеціальних (ключових) даних з метою приховування/відновлення змісту інформації, підтвердження її спроможності, цілісності, авторства тощо.

Криптографічне перетворення інформації – перетворення інформації з використанням одного з криптографічних алгоритмів, обумовлене цільовим призначенням криптографічної системи.

Літерал – логічна перемінна або її інверсія (заперечення).

Логічна перемінна – логічна (булева) перемінна, або простий висліз, це така величина, яка може приймати лише два значення – 0 або 1.

Логічна функція – функція, яка так само, як і її аргументи, може приймати лише два значення – 0 або 1.

Логічне заперечення – функція **„НЕ" (NOT)** – функція, що є інверсією або запереченням аргументу.

Логічний елемент (вентиль) – комбінаційна логічна схема, яка реалізує одну з елементарних логічних функцій: **„НЕ”, „І”, „АБО”, „І-НЕ”** і т. д.

Логічний оператор – елементарна логічна функція, яка реалізована відповідним комбінаційним логічним елементом, тобто вентилем.

Логічні зв’язки – об’єднання простих висловлювань у логічні функції.

Логічні схеми – електронні схеми, які реалізують певні логічні функції.

Макстерм, або **диз’юнктивний терм** – логічна функція, яка зв’язує всі перемінні в прямій або інверсній формі, тобто літерали, знаком диз’юнкції.

Машинне зображення числа – це представлення числа в розрядній сітці цифрового автомата.

Мережа – кілька інформаційних систем, зв’язаних між собою для обміну інформацією і ресурсами.

Мережа може охоплювати одну кімнату або всю будівлю яку займає організація. Щоб розширити мережу, підключивши до неї інформаційні системи, розташовані на іншій вулиці або навіть в іншій країні, можна скористатися телефонними лініями.

Мережі інформаційних систем – це сукупність інформаційних систем і різних пристроїв, що забезпечують інформаційний обмін між інформаційними системами в мережі без використання будь-яких проміжних носіїв інформації.

Все різноманіття мереж інформаційних систем можна класифікувати за групою ознак:

– територіальна поширеність;

– відомча приналежність;

– швидкість передачі інформації;

– тип середовища передачі.

За територіальною поширеністю мережі можуть бути локальними, регіональними.

Термін “корпоративна мережа” також використовується в літературі для позначення об’єднання декількох мереж, кожна з яких може бути побудована на різних технічних, програмних і інформаційних принципах.

За приналежністю розрізняють відомчі і державні мережі. Відомчі належать одній організації і розташовуються на її території. Державні мережі – мережі, які використовуються в державних структурах.

За швидкістю передачі інформації мережі діляться на низько-, середньо- і високошвидкісні.

За типом середовища передачі розділяються на мережі коаксіальні, на витій парі, оптоволоконні, з передачею інформації по радіоканалах в інфрачервоному діапазоні тощо.

Інформаційні системи можуть з'єднуватися кабелями, утворюючи різну топологію мережі (радіальна, лінійна, кільцева тощо).

Слід розрізняти мережі інформаційних систем і універсальні інформаційні мережі. Мережі інформаційних систем зв'язують інформаційні системи, кожна з яких може працювати і автономно. Універсальні інформаційні мережі (термінальні мережі) зазвичай зв'язують потужні інформаційні системи (майнфрейми), а в окремих випадках і інформаційні системи з пристроями (терміналами), які можуть бути достатньо складні, але поза мережею їх робота або неможлива, або взагалі втрачає сенс. Наприклад, мережа банкоматів. Будуються вони на абсолютно інших, ніж мережі інформаційних систем, принципах і навіть на іншій обчислювальній техніці. У класифікації мереж існує два основні терміни: LAN і WAN.

LAN (Local Area Network) – локальні мережі, що мають замкнуту інфраструктуру до виходу на постачальників послуг. Термін "LAN" може описувати і маленьку офісну мережу, і мережу рівня великої організації, що займає декілька сотень гектарів. Зарубіжні джерела дають навіть близьку оцінку – близько 10 км в радіусі; використання високошвидкісних каналів.

WAN (Wide Area Network) – глобальна мережа, що покриває великі географічні регіони, що включають як локальні мережі, так і інші телекомунікаційні мережі і пристрої. Приклад WAN – мережі з комутацією пакетів (Frame Relay), через яку можуть "розмовляти" між собою різні мережі інформаційних систем.

Розглянуті вище види мереж є мережами закритого типу, доступ до них дозволений тільки обмеженому колу користувачів, для яких робота в такій мережі безпосередньо пов'язана з їх професійною діяльністю. Глобальні мережі орієнтовані на обслуговування будь-яких користувачів.

Структура локальних мереж інформаційних систем – спосіб з'єднання інформаційних систем називається структурою або топологією мережі. Мережі можуть мати топологію "лінійна" "радіальна" "кільцева". У першому випадку (лінійна) рис. Г. 1 (а), всі інформаційні системи підключені до одного загального кабелю (шині), в другому (радіальна) рис. 1 (в) – є спеціальний центральний пристрій, від якого йдуть кабелі до кожної інформаційної системи, тобто кожна інформаційна система підключений до свого кабелю.

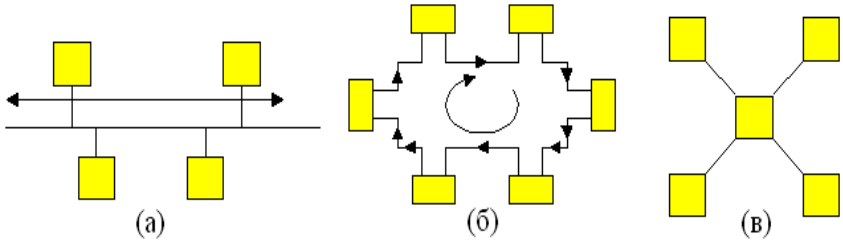


Рис. Г.1. Структура побудови (а) лінійна, (б) кільцева, (в) радіальна

Структура типу “лінійна”, рис. Г.1 (а), простіша й економічна, оскільки для неї не потрібний додатковий пристрій і витрачається менше кабелю. Але вона дуже чутлива до несправностей кабельної системи. Якщо кабель пошкоджений хоч би в одному місці, то виникають проблеми для всієї мережі.

Перевага кільцевої структури рис. Г.1 (б) – простота реалізації пристроїв, а недолік – низька надійність.

Всі розглянуті структури – ієрархічні. Проте, завдяки використанню мостів, спеціальних пристроїв, об’єднуючих локальні мережі з різною структурою, з вище перелічених типів структур можуть бути побудовані мережі з складною ієрархічною структурою.

Локальна інформаційна мережа – з’єднання декількох інформаційних систем між собою лініями зв’язку для передачі інформації між підрозділами організації з метою її спільної обробки.

Локальна мережа – мережа, системи якої розташовані на невеликій відстані друг від друга.

Локальна мережа, іменована також локальною мережею інформаційних систем, охоплює невеликий простір, як правило, будинок. Вона характеризується великою швидкостями передачі даних. Важливо те, що в локальній мережі канали мають високу якість і належать одній організації. Застосовуються дві архітектури локальних мереж. Архітектура клієнт-сервер дозволяє ефективно використовувати ресурси серверів. Однорангова архітектура припускає взаємодію рівноправних абонентських систем.

Глобальна мережа – мережа, абонентські системи якої розташовані в багатьох країнах.

Прагнення до надання мережних служб і ресурсів великій кількості користувачів привело до об'єднання територіальних мереж і створенню глобальних мереж. Завдяки свої великим розмірам кожна з них

надає своїм користувачам тисячі баз даних, міжконтинентальну електронну пошту, можливість навчання практично будь-яким спеціальностям. Крім цього, глобальна мережа є сполучною ланкою великої кількості невеликих мереж. Прикладом такої мережі є мережа Internet. Спочатку створювалися глобальні телефонні мережі. На зміну їм прийшли мережі комутації пакетів і цифрові мережі. Виділилися два класи мереж: кабельна мережа й бездротова мережа. Створення глобальних мереж привело до появи архітектури мережа інформаційної системи, у якій прості й високоефективні мережні інформаційних систем стали компонентами цих мереж і призначені для використання їхніх великих можливостей. Абонентські системи, побудовані на цих інформаційних системах, дозволили їхнім власникам інтегруватися у світову інформаційну інфраструктуру.

Комунікаційна мережа – мережа, основним завданням якої є передача даних.

Стосовно типам переданих сигналів розрізняють аналогові мережі й дискретні мережі. Вимоги підвищення надійності й збільшення пропускної здатності комунікаційної мережі привели до необхідності переходу на дискретні принципи. **У** результаті, замість аналогових мереж майже повсюдно стали використовуватися дискретні мережі. Залежно від простору, що покривається, комунікаційні мережі утворять локальні мережі, територіальні мережі та глобальні мережі. Крім цього, виділяють кабельні мережі й бездротові мережі.

Віртуальна мережа – мережа, характеристики якої, в основному, визначаються її програмним забезпеченням.

Основними причинами, що приводять до створення віртуальної мережі, є:

– необхідність створення оперативних ізольованих від інших користувачів робочих груп;

– бажання полегшити процедури переміщення, додавання й видалення об’єктів мережі;

– прагнення надати оперативну можливість зміни ролей, щоб клієнт, коли це необхідно, міг виступати в ролі сервера;

– можливість забезпечення безпеки даних шляхом локалізації трафіку в рамках ізольованої групи.

Програмне забезпечення дозволяє не тільки видозмінювати характеристики реальної мережі, але й створювати так звані мережі. Ці мережі з її абонентам виглядають зовсім інакше, чим їхня фізична сутність і структура. Тут поняття віртуальний, що використовувалося раніше в каналах, системах і їхніх компонентах, тепер поширилося й на мережі. Другий випадок пов’язаний з створенням замкнутої групи абонентських

систем. Для цього, в одній комунікаційній мережі встановлюються інтелектуальні пристрої (вузли комутації, концентратори, мости тощо), які відповідно до вказівок адміністративної системи з'єднують один з одним логічні канали, що створює закриту для інших абонентів віртуальну мережу. В одній великій асоціації фізичних мереж може бути створене значна кількість віртуальних мереж, що функціонують незалежно друг від друга. Віртуальна технологія має велику гнучкість, що дозволяє динамічно змінювати число й склад віртуальних мереж скільки завгодно раз. Воно дозволяє адміністраторові мережі переміщати абонентські системи з однієї віртуальної мережі в іншу за допомогою простих операцій буксирування об'єктів на екрані монітора.

Віртуальний – умовний, за допомогою спеціальних методів сприймається інакше, чим реалізований фізично.

Віртуальний – термін, використовується для вказівки того, що фізична реалізація відрізняється від тої, котра надається користувачеві, прикладному процесу або функціональному блоку. Відповідно до цього, у мережах широко використовуються такі поняття, як: віртуальна машина, віртуальна реальність, віртуальна мережа, віртуальна форма подання даних, віртуальний ланцюг, віртуальне завдання, віртуальна адреса, віртуальний виклик, віртуальний канал, віртуальний клас, віртуальний маршрут, віртуальний принтер, віртуальний термінал.

Віртуальними можуть також бути дані, керування мережею, завдання, виклик, файл, адреса та інші об'єкти мережі. Віртуальною пам'яттю є реальна пам'ять, надавана спільно працюючим оперативним запам'ятовувальним пристроєм і зовнішнім запам'ятовувальним пристроєм. Вони за допомогою спеціальних програм так поєднуються разом, що користувач звертається до них так, як до однієї загальної оперативної пам'яті. Таке об'єднання дає можливість прикладним процесам працювати з пам'яттю, розмір якої значно більше оперативної. При цьому, природно, швидкість звертання до пам'яті зменшується. Механізм створення віртуальної пам'яті полягає в тім, що в міру заповнення оперативної пам'яті частина її даних операційна система скидає на зовнішній запам'ятовувальний пристрій. Потім, у міру необхідності, ці дані вертаються в оперативну пам'ять.

Корпоративна мережа – локальна мережа великої організації.

Корпоративна мережа, іменована також мережею масштабу організації, призначена для автоматизації всіх робіт, виконуваних організацією: від проектування нових виробів до їхньої реклами й продажу. Спочатку ця мережа складалася з однієї або групи базових інформаційних систем, до яких були підключені термінали. Потім, термінали були замінені міні-інформаційні системи. Пізніше замість міні-

інформаційні системи стали використовуватися робочі станції і інформаційні системи.

Сучасна корпоративна мережа характеризується ієрархічною організацією утворюючих її систем. Класична структура цієї мережі є двухшарова, що поєднує високопродуктивну центральну підмережу організації із групою підмереж підрозділів організації. До центральної підмережі, крім підмереж підрозділів, підключається комплекс базових інформаційних систем. У великого числа шарів може збільшуватися, а в малих – скорочуватися. Аналогічно цьому, змінюється число базових інформаційних систем або вони можуть зовсім відсутні. На комплекс базових інформаційних систем покладають завдання створення великих баз даних організації, керування ресурсами мережі, забезпечений небезпеки даних. Природно, що корпоративна мережа повинна взаємодіяти з територіальною мережею. Концепція корпоративної мережі, що сформувалася на базі синтезу моделей централізованої й розподіленої обробки даних характеризується:

– широким застосуванням робочих станцій і інформаційних систем;

– використанням устаткування та програмного забезпечення різних виробників;

– забезпеченням максимальної ефективності роботи;

– високою гнучкістю й динамізмом організації;

– оптимізацією архітектури мережі та використовуваних у ній прикладних програм.

У зв’язку з розвитком і успіхами мережі Internet, практично всі корпоративні мережі з’єднуються із цією мережею. Більше того, з’явилися мережі Internet. Їхня сутність полягає у використанні інфраструктури Internet.

Нейронна мережа. Нейронна мережа являє собою сукупність нейроподібних елементів, певним чином зв’язаних один з одним і зовнішнім середовищем за допомогою зв’язків, обумовлених ваговими коефіцієнтами. У процесі функціонування мережі здійснюється перетворення вхідного вектора у вихідний, деяка переробка інформації. Конкретний вид виконуваного мережею перетворення даних обумовлює не тільки характеристиками нейроподібних елементів, але й особливостями її архітектури, а саме топологією міжнейронних зв’язків, вибором певних підмножин нейроподібних елементів для введення і виводу інформації, способами навчання мережі, наявністю або відсутністю конкуренції між нейронами, напрямком і способами керування та синхронізації передачі інформації між нейронами. Найбільше часто нейронні мережі використовуються для вирішення наступних проблем:

– класифікація образів – вказівка приналежності вхідного образу, представленого вектором ознак, одному або декільком попередньо певним класам;

– кластерізація – класифікація образів при відсутності навчальної вибірки з мітками класів;

– прогнозування – пророкування значення $y(tn+1)$ при відомій послідовності $y(t\ 1), y(t2)... y(tn)$;

– оптимізація – знаходження рішення, що задовольняє системі обмежень і максимізує або мінімізує цільову функцію. Пам'ять, яка адресується по змісту (асоціативна пам'ять) – пам'ять, доступна при вказівці заданого змісту;

– керування – розрахунок такого вхідного впливу на систему, при якому система слідує по бажаній траєкторії.

Територіальна мережа – мережа, системи якої знаходяться в різних географічних місцях.

Територіальна мережа, що іменується також регіональною мережею охоплює великий простір – район, область, регіон, країну, групу країн. У тому випадку, коли вона охоплює континенти, використовується визначення глобальної мережі. Характерною особливістю територіальної і глобальної мереж є застосування великої кількості вузлів комутації або супутників зв'язку. Мережа використовує різноманітні типи каналів. Територіальна мережа повинна задовольняти наступним вимогам:

– включати велику кількість абонентських систем (до декількох тисяч);

– покривати великий географічний район;

– забезпечувати широкомовлення і доставку повідомлень групам і окремим адресатам;

– мати високу пропускну спроможність, що доходить до десятків Гбіт/с;

– володіти надійністю в роботі; гарантувати безпеку даних;

– передавати різноманітні види даних: тексти, звук, нерухомі та рухомі зображення.

Територіальні мережі діляться на два класи: мережі з маршрутизацією даних і мережі з селекцією даних. Окрім цього, виділяють кабельні мережі і бездротові мережі. Представниками територіальних мереж є: цифрова мережа з інтегральним обслуговуванням, мережа комутації пакетів, мережа комутації каналів, телефонна мережа.

Мінімальна нормальна диз'юнктивна форма – нормальна диз'юнктивна форма з мінімальною кількістю мінтермів, які мають мінімальні ранги, жоден з яких виключити не можна, або НДФ, яка містить найменшу кількість букв (літералів).

Мінімальна нормальна кон'юнктивна форма – нормальна кон'юнктивна форма з мінімальною кількістю макстермів, що мають мінімальні ранги, жоден з яких виключити не можна, або НКФ, що містить найменшу кількість букв (літералів).

Мінімізація логічної функції – отримання її МНДФ або МНКФ для того, щоб у подальшому отримати мінімальну кількість логічних елементів в електронній схемі, призначеній для реалізації цієї логічної функції.

Мінтерм, або **кон'юнктивний терм** – логічна функція, яка зв'язує перемінні в прямій або інверсній формі, тобто літерали, знаком кон'юнкції.

Набір – сукупність значень аргументів логічної функції.

Накопичувальні схеми – логічні схеми, вихідний сигнал яких залежить і від вхідних сигналів, і від стану схеми в попередні моменти часу.

Непозиційна система числення – система, для якої значення символу, тобто цифри, не залежить від його положення в числі.

Нормальна диз'юнктивна форма – форма, яка становить собою суми елементарних добутків.

Обчислювальна машина – інструментальний засіб для рахування, обчислення і перетворення інформації.

Обчислювальні машини у різний час мали різні можливості та назви: рахункові дошки, рахункові інструменти, прибори, а з середини нашого століття – (комп'ютер) електронна обчислювальна машина, персональна електронна обчислювальна машина, автоматизована система (Закон України "Про захист інформації в автоматизованих системах") і в теперішній час інформаційна (автоматизована) система, інформаційно-телекомунікаційна система, телекомунікаційна система (Закон України "Про захист інформації в інформаційно-телекомунікаційних системах").

Найбільш відомими засобами рахувати й обраховувати у далекому минулому були механічні пристрої, яки дозволяли виконувати арифметичні дії, зберігати простий поточний результат. Для довгострокового зберігання результатів застосовувався, як правило, любий письмовий матеріал. У перших обчислювальних машин процес рахунку був послідовний, поопераційний. У 19 столітті англійський учений Ч. Бєббидж сформулював ідею зберігання програми, яка складається в тому, що машина повинна спочатку отримати від людини всю послідовність необхідних обчислень, а потім вже робити обчислення і видавати результат без втручання людини, тобто автоматично. У широкому значенні автоматичні обчислювальні машини по принципу представлення інформації можна класифікувати як аналогові та цифрові. У аналогових обчислювальних машин використовується аналогія між значеннями, що приймають деяку фізичну величину (наприклад швидкість обертання вала, електрична напруга) і змінною у задачі, яка вирішується. Цифрові

обчислювальні машини безпосередньо оперують з числами, одначе і тут для представлення такої абстракції, якої є число, використовується фізична величина. У цьому смислі цифрові обчислювальні машини також володіють аналоговими властивостями. Поділ обчислювальних машин на аналогові і цифрові можливо при любому принципу їх побудови. Наприклад, логарифмічна лінійка по суті є аналоговою обчислювальною машиною, побудованою на основі механічного принципу, що складається з складанні довжин відрізків (є аналоговим представленням чисел) при виконання операцій множення, ділення, находження корнів чисел тощо.

Комп'ютер – (англ. computer від лат. computo – рахую, вираховую) – запозичений з англійської літератури назву електронної обчислювальної машини.

Електронна обчислювальна машина – обчислювальна машина, основним елементом якої являються електронні прибори.

Перша електронна цифрова обчислювальна машина була створена в 1946 (США), з 1948 почалось серійне виробництво електронних машин. Перша вітчизняна електронна цифрова обчислювальна машина була розроблена в Інституті електротехніки АН УРСР в 1950 році. Електронна обчислювальна машина принципово відрізняється від механічних, електричних, електромеханічних та інших машин компонентами і формою створення і представлення сигналу. Вони більш компактні, економічні, надійні, володіють великою швидкодією, зручно стикуються з зовнішнім джерелом інформації, в зручному вигляді видають результати обробки інформації. Електронні обчислювальні машини можливо об'єднувати у обчислювальні комплекси для переробки інформації на різних рівнях або в обчислювальних системах для переробки великих масивів інформації при сумісній праці. По способу обробки представленої в електронній обчислювальній машині інформації розрізняють цифрові обчислювальні машини, які оперують з інформацією, представленою в дискретній формі (у вигляді цифрових кодів), аналогові обчислювальні машини, які обробляють дані, представлені в безперервній (аналоговій) формі, та гібридні обчислювальні системи, в яких перероблювана інформація представляється частково в дискретній, частково у безперервній формах.

У 60-х роках у електронно-обчислювальних машинах, що займали цілі зали, для обчислювальних операцій застосовувалися друковані плати на транзисторах.

Персональна електронна обчислювальна машина – електронна обчислювальна машина, яку можливо експлуатувати без допомоги професійного програміста, тобто самостійно, персонально.

Персональна електронна обчислювальна машина складається із базового комплекту, периферійних пристроїв, інших технічних і програмних засобів, які забезпечують виконання функціональних

характеристик і завдань користувача. Базовий комплект включає: центральний процесор на основі інтегральної схеми 8-, 16- або 32-разрядного мікропроцесора, запам'ятовуючий пристрій оперативний, постійний запам'ятовуючий пристрій, клавіатуру, пристрій відображення інформації (алфавітно-графічний дисплей), засоби підключення периферійних пристроїв (адаптери, контролери), засоби підключення пристроїв з'єднання з мережею та розширення функціональних можливостей персональної електронної обчислювальної машин. До периферійних пристроїв персональної електронної обчислювальної машини відносяться: запам'ятовуючий пристрій зовнішній (накопичувачі на гнучких і жорстких дисках, на оптичних дисках, а також такі, що використовують інші фізичні принципи); пристрої вводу інформації (речової, вводу текстової і графічної інформації тощо); пристрої виводу інформації (друкуючі, графічні, речові тощо); додаткові пристрої управління процесом обробки вводу і виводу інформації (планшети, маніпулятори тощо). Всі вказані компоненти персональної електронної обчислювальної машини зазвичай об'єднуються у систему за допомогою уніфікованої загальної шини, по якій центральний процесор обмінюється даними із своїми зовнішніми пристроями.

Перші персональні електронні обчислювальні машини, за сучасними стандартами мали дуже обмежені можливості. Працювали повільно, інформацію в них доводилося зберігати на звичайній компакт-касеті, а обсяг внутрішньої пам'яті становив лише один кілобайт.

Дискета використовувалася для зберігання інформації і для перенесення її на іншу персональну електронну обчислювальну машину.

Автоматизована система – система, що забезпечує автоматизоване оброблення даних і до складу якої входять технічні засоби їхнього оброблення (засоби обчислювальної техніки і зв'язку), а також методи і процедури, програмне забезпечення. *(Закон України "Про захист інформації в автоматизованих системах")*

Інформаційно-телекомунікаційна система – сукупність інформаційних та телекомунікаційних систем, які у процесі обробки інформації діють як єдине ціле.

Телекомунікаційна система – сукупність технічних і програмних засобів, призначених для обміну інформацією шляхом передавання, випромінення або приймання її у вигляді сигналів, знаків, звуків, рухомих або нерухомих зображень чи в інший спосіб програмних засобів. *(Закон України "Про захист інформації в інформаційно-телекомунікаційних системах").*

Інформаційна технологія − сукупність методів, виробничих процесів і програмно-технічних засобів, об'єднаних у технологічний

ланцюжок, що забезпечує збір, зберігання, обробку, вивід і поширення інформації для зниження трудомісткості процесів і використання інформаційних ресурсів, підвищення надійності й оперативності.

Основа (базис) позиційної системи числення – кількість знаків або символів, використовуваних для зображення числа в певній системі.

Плаваюча кома – машинна форма представлення речовинних чисел.

Побічне електромагнітне випромінення і навід – електромагнітне випромінення і навід, що є побічним результатом функціонування технічних засобів і можуть бути носіями інформації.

Повідомлення – інформація, втілена й зафіксована в деякій матеріальній формі.

Позиційна система числення – система, у якій значення кожної цифри залежить не лише від її числового еквівалента, але й від її місця (позиції) в числі, тобто той самий символ (цифра) може приймати різні значення.

Похибка – різниця між розрахованим, спостережуваним або виміряним значенням величини чи параметру та дійсним, установленим або теоретично правильним значенням величини чи параметру.

Прості, або **елементарні імпліканти** – найкоротші добутки або найкоротші суми, що входять у певну ЛФ, або імпліканта типу елементарного добутку (мінтерма) чи елементарної суми (макстерма), ніяка власна частина якого вже не є імплікантою цієї ЛФ.

Ранг терма – кількість перемінних та їхніх інверсій, тобто кількість літералів, які входять до цього терму.

Сигнал – фізичний засіб передачі повідомлення. Типи сигналів: аналоговий, дискретний, квантований, кодований, модульований, імпульсний та ін.

Синхронний автомат – автомат, який функціонує під управлінням тактових (або синхронізуючих) сигналів, що мають постійну тривалість і частоту, якщо квантування часу обрано рівномірним.

Система числення – сукупність прийомів і правил для запису чисел.

Систематичний код – код, який містить, крім інформаційних, також контрольні розряди, у які записано деяку інформацію про початкове число, необхідну для виявлення можливих викривлень у записі числа, що виникли з тих або інших причин.

Скорочена форма аналітичного представлення ЛФ – диз'юнкція всіх її простих імплікант.

Суміщений автомат – автомат, який відрізняється від автоматів Мілі й Мура тим, що одночасно реалізує дві функції виходів, кожна з яких характерна для цих автоматів окремо.

Суперпозиція – підстановка в логічну функцію замість її аргументів інших логічних функцій.

Сусідні терми – терми в НДФ або НКФ, які відрізняються лише одною перемінною: в одному термі перемінна без заперечення, а в іншому – із запереченням.

Таблиця істинності – таблиця, у якій наведено всі можливі набори аргументів деякої логічної функції й відповідні їм значення самої функції.

Терм – група логічних перемінних у прямій або інверсійній формі, тобто група літералів деякої логічної функції, об'єднаних тим самим знаком логічної зв'язки – логічного додавання або ж логічного множення. У термі кожна перемінна або її заперечення трапляється лише один раз, тобто в терм може входити або перемінна певної функції, або її заперечення.

Технічний захист інформації – вид захисту інформації, спрямований на забезпечення за допомогою інженерно-технічних заходів та/або програмних і технічних засобів унеможливлення витоку, знищення та блокування інформації, порушення цілісності та режиму доступу до інформації.

Тригер – елементарний автомат Мура який має два внутрішніх стійких стани, що відповідають логічним 1 і 0, тобто логічний елемент запам'ятовування.

Тупикова форма аналітичного представлення ЛФ – диз'юнкція простих імплікант, жодну з яких виключити не можна.

Фіксована кома (точка) – машинна форма представлення цілих чисел або правильного дробу. Розрядну сітку, виділену для представлення числа в цій формі, розбито на дві частини: старший розряд виділено під знак числа, а в іншій частині (у полі числа) представлено значення числа. Для цілих чисел кома фіксується правіше молодшого розряду, а для правильного дробу – лівіше старшого розряду поля числа.

Функціонально повна система логічних функцій – система, за допомогою логічних функцій якої, застосовуючи операції суперпозиції й підстановки, можна отримати будь-яку скільки завгодно складну логічну функцію.

Функція „АБО-НЕ" (стрілка Пірса) – функція, яка істинною є лише тоді, коли всі перемінні хибні.

Функція „І-НЕ" (штрих Шеффера) – функція, яка є хибною тоді, коли всі перемінні одночасно істинні.

Функція виходів цифрового автомату – залежність, яка визначає вихідний сигнал автомата від стану цифрового автомату й вхідного сигналу в певний момент часу.

Функція переходів цифрового автомату – залежність, яка визначає стан автомата в момент дискретного часу залежно від стану автомата й значення вхідного сигналу.

Цифровий автомат – дискретний перетворювач інформації, здатний приймати різні стани, переходити під впливом вхідних сигналів чи команд програми розв'язання задачі з одного стану в інший і видавати вихідні сигнали.

Числовий спосіб представлення логічної функції – спосіб, при якому у випадку ДНДФ під знаком суми перераховано взяті в дужки номери наборів, на яких функція дорівнює одиниці. У випадку ДНКФ – під знаком добутку перераховано взяті в дужки номери наборів, на яких функція дорівнює нулю.

Шина – один або кілька провідників, які слугують для з'єднання групи пристроїв.

Додаток

ЛІСТИНГИ ПРОГРАМ МОДЕЛЮВАННЯ ЦИФРОВИХ КОМПОНЕНТІВ СИСТЕМ ЗАХИСТУ ІНФОРМАЦІЇ

Шифратор двійкового коду

```
#include <iostream>
using namespace std;
int main()
{
	int a[8];
	int b[3];
	bool k1,k2,k3;
	int i,n;
	cout<<"Enter a binary number from 0 to 8\n";
	cin>>n;
	while(n!=9)
	{
	for(int i=0;i<8;i++)
	{
		if(i+1==n)
		{
			a[i]=1;
		}
		else
		{
			a[i]=0;
		}
	}
	cout<<"\n";
	for(int i=0;i<8;i++)
	{
	cout<<a[i];
	}
	cout<<"\n";
	k1=(a[4]==1)|(a[5]==1)|(a[6]==1)|(a[7]==1);
	k2=(a[2]==1)|(a[3]==1)|(a[6]==1)|(a[7]==1);
	k3=(a[1]==1)|(a[3]==1)|(a[5]==1)|(a[7]==1);

	if(k1)
```

```
{
        b[0]=1;
}
else
{
        b[0]=0;
}
if(k2)
{
        b[1]=1;
}
else
{
        b[1]=0;
}
if(k3)
{
        b[2]=1;
}
else
{
        b[2]=0;
}
cout<<"A register B contains a code\n";
for(int i=0;i<3;i++)
{
        cout<<b[i];
}
cout<<"\n";
cout<<"Enter - Enter a binary number from 0 to 8, 9 - Completion\n";
cin>>n;

}
}
```

Дешифратор двійкового коду

```
#include<iostream>
#include "stdio.h"
using namespace std;
void main()
{
	int c,n;
	int b[8];
	int a[3];
	bool D[8];
	char str[6];
	char *px1,*px2,*px3;
	cout<<"Enter through blanks in a register 'A' a three-digit binary number\n";
	gets(str);
	while(str[0]!='9')
	{
	px1=&str[0];
	px2=&str[2];
	px3=&str[4];
	a[0]=atoi(px1);
	a[1]=atoi(px2);
	a[2]=atoi(px3);
	D[0]=(a[0]==0)&(a[1]==0)&(a[2]==0);
	D[1]=(a[0]==0)&(a[1]==0)&(a[2]==1);
	D[2]=(a[0]==0)&(a[1]==1)&(a[2]==0);
	D[3]=(a[0]==0)&(a[1]==1)&(a[2]==1);
	D[4]=(a[0]==1)&(a[1]==0)&(a[2]==0);
	D[5]=(a[0]==1)&(a[1]==0)&(a[2]==1);
	D[6]=(a[0]==1)&(a[1]==1)&(a[2]==0);
	D[7]=(a[0]==1)&(a[1]==1)&(a[2]==1);
	for(int i=0;i<8;i++)
	{
		if(D[i])
		{
			b[i]=1;
		}
		else
		{
			b[i]=0;
		}
```

```
        }
        cout<<"A register B contains a code:";
        for(int i=0;i<8;i++)
        {
                cout<<b[i]<<' ';
        }
        cout<<endl;
        cout<<"Enter - Enter a binary number from 000 to 111, 9 - exit\n";
        gets(str);
        }

}
```

Навчальне видання

ЛАХНО Валерій Анатолійович,
ГУСЄВ Борис Семенович
КАСАТКІН Дмитро Юрійович

Компютерна логіка

Загальна редакція: Лахно В.А.

Видавництво КОМПРІНТ
Свідоцтво про реєстрацію: Серія ДК № 1620 від 18.12.03
Адреса видавництва: 91034, м.Луганськ, кв.Молодіжний, 20а.
Тел. (0642) 41-34-12. Факс (0642) 41-31-60
E-mail: uni@snu.edu.ua