КИЇВСЬКИЙ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

О. В. Барабанов
О. С. Баужа

ОСНОВИ ЦИФРОВОЇ СХЕМОТЕХНІКИ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

О. В. Барабанов
О. С. Баужа

ОСНОВИ ЦИФРОВОЇ СХЕМОТЕХНІКИ

Навчальний посібник

КИЇВСЬКИЙ УНІВЕРСИТЕТ 2016

УДК 004.31:621.38(075.8)
ББК 32.973.26я73
Б24

Рецензенти:
д-р техн. наук, проф. С. Д. Погорілий,
д-р пед. наук, проф. Б. А. Сусь

Затверджено вченою радою факультету радіофізики, електроніки та комп'ютерних систем (протокол № 4 від 9 листопада 2015 року)

Ухвалено науково-методичною радою Київського національного університету імені Тараса Шевченка (протокол № 4-15/16 від 21 квітня 2016 року)

Барабанов О. В.

Б24 Основи цифрової схемотехніки : навч. посіб. / О. В. Барабанов, О. С. Баужа – К. : ВПЦ "Київський університет", 2016. – 103 с.

Розглянуто основи булевої логіки та методи мінімізації булевих виразів. Викладено принципи роботи основних функціональних вузлів цифрових електронних систем: мультиплексорів, схем додавання, множення та зсуву, тригерів, регістрів зсуву, лічильників тощо. Значну увагу приділено розгляду скінченних автоматів і часових характеристик цифрових схем.

Для студентів природничих і технічних факультетів вищих навчальних закладів.

УДК 004.31:621.38(075.8)
ББК 32.973.26я73

ПЕРЕДМОВА

Більшість сучасних електронних систем цифрові. Технології, які в них використовуються, постійно стають більш складними. Знайомство з основними принципами побудови таких систем важливе для розробників як апаратного, так і програмного забезпечення. Цей посібник допоможе студентам познайомитися із фундаментальними принципами, на яких ґрунтується робота сучасних цифрових електронних схем.

Навчальний посібник розрахований на студентів вищих навчальних закладів технічного напряму, які прослухали курси лінійної алгебри, математичного аналізу та знайомі з методами аналізу електричних кіл.

На початку посібника розглянуто основні логічні елементи та основи булевої логіки: аксіоми, теореми, методи мінімізації булевих виразів. Велику увагу приділено представленню чисел у цифрових системах та операціям із ними. Далі розглянуто комбінаційні та послідовнісні схеми: дешифратори, мультиплексори, схеми додавання, множення, зсуву, тригери, лічильники. Значне місце посідає розгляд проектування скінченних автоматів. І нарешті, посібник знайомить читача із часовими характеристиками послідовнісних схем.

1. ЦИФРОВА ЕЛЕКТРОНІКА

Більшість реальних фізичних величин (швидкість, густина, напруга, струм, напруженість електричного поля і т. д.) змінюються неперервним чином. З іншого боку, у сучасних цифрових системах інформація представлена у вигляді змінних, які змінюються дискретно і мають обмежену кількість дозволених строго визначених значень. Зазвичай робота комп'ютерів базується на використанні двійкового (бінарного) коду, який володіє лише двома такими дозволеними значеннями. Завдяки існуванню великої кількості бістабільних схем у напівпровідниковій схемотехніці оперувати з двома рівнями напруги досить легко. При використанні двійкового коду висока напруга відповідає одиниці, а низька напруга – нулю (поняття *висока* та *низька напруга* визначаються напівпровідниковою технологією, що використовується).

Обсяг інформації D, який передається (або зберігається) однією дискретною змінною, що може перебувати в N різних станах, вимірюється в одиницях, які називають *бітами*, й обчислюється за формулою $D = \log_2 N$.

Одна двійкова змінна передає $\log_2 2 = 1$ – один біт інформації (абревіатура *bit* (біт, *binary digit*), розшифровується як двійкове число). Неперервний сигнал здатний передати нескінченну кількість інформації, оскільки може набувати необмеженої кількості значень. На практиці обсяг інформації, що передається більшістю неперервних сигналів, становить від 10 до 24 бітів. Це зумовлено такими технічними обмеженнями, як шум та помилки при вимірюванні рівня сигналу. Якщо вимірювання рівня необхідно провести дуже швидко, то обсяг інформації, що передається, буде ще нижчим.

Розглянемо цифрові схеми, які використовують двійкові змінні, а саме: 0 та 1. Англійський математик Дж. Буль (1815–1864) у 1854 р. розробив систему логіки, що використовує двійкові змінні, і тому її називають його ім'ям – *булева логіка*. Булеві змінні можуть набувати двох значень: 1, (ІСТИНА, TRUE) або 0, (НЕПРАВДА, FALSE). У цифрових електронних пристроях позитивна напруга зазвичай (але не завжди) представляє 1, а нульова напруга – 0. Конкретні значення напруг 0 та 1 різні для різних технологій виготовлення мікросхем. Наприклад, при використанні традиційної комплементарної метал-оксид-напівпровідникової технології (КМОН-технології) напруга на вході елемента більша ніж 3,15 В відповідає логічній 1, а менша 1,35 В – логічному 0. Діапазон вхідних напруг 1,35–3,15 В заборонений, оскільки для таких значень коректна робота елемента не гарантується. Для інших технологій порогові значення будуть відрізнятися від наведених вище.

Розробник цифрової системи може повністю зосередитися на обробці абстрактних булевих значень 0 та 1 та ігнорувати, яким чином булеві змінні будуть представлені на фізичному рівні. Програміст може працювати, не володіючи детальною інформацією про апаратне забезпечення комп'ютера. Проте розуміння того, як саме працює це апаратне забезпечення, дозволяє йому набагато краще оптимізувати програму для конкретного комп'ютера. Один біт не може передати великої кількості інформації. На практиці декілька бітів об'єднуються в групи для представлення чисел з різною точністю, символів і т. д.

1.1. Системи числення

У повсякденному житті, науковій та інженерній діяльності традиційно застосовуються десяткові числа. Проте в цифрових системах використовуються тільки два значення: 0 та 1. У таких системах зручніше працювати з двійковими або шістнадцятковими числами. У цьому розділі розглянемо всі поширені на сьогодні системи числення.

1.1.1. Десяткова система числення

Десяткова система числення використовує десять арабських цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 – саме стільки, скільки у нас пальців на руках. Числа більше 9 записуються у вигляді рядка цифр. Причому, цифра, що стоїть в кожній подальшій (лівішій) позиції такого рядка має "вагу", що вдесятеро перевищує "вагу" цифри, що стоїть у попередній (правішій) позиції. Отже, десяткова система є системою числення за основою 10. При переміщенні справа наліво "вага" кожної позиції збільшується: 1, 10, 100, 1000 і т. д. Позицію, яку цифра займає в рядку десяткового числа, називають *розрядом*, або *декадою*. Кожне десяткове число може бути записане у вигляді суми всіх його цифр, помножених на "вагу" (відповідний ступінь 10) розряду кожної цифри:

Тисячі Сотні Десятки Одиниці

$$1\ 9\ 8\ 4 = 1\times10^3+9\times10^2+8\times10^1+4\times10^0=1000+900+80+4.$$

Щоб уникнути непорозумінь при одночасній роботі з більш ніж однією системою числення, основа системи часто вказується позаду і трохи нижче від основного числа: 1984_{10}. *N*-розрядне десяткове число може представляти одну із 10^N цифрових комбінацій: 0, 1, 2, 3,..., 10^N–1. Таким чином, за допомогою *N* розрядів можна представити будь-яке число в діапазоні від 0 до 10^N–1. Наприклад, десяткове число, що складається із чотирьох цифр (розрядів), представляє одну із 10000 можливих цифрових комбінацій, тобто чисел, у діапазоні від 0 до 9999.

1.1.2. Двійкова система числення

Один біт може набувати одного з двох значень: 0 або 1. Декілька бітів, сполучених в одному рядку, утворюють двійкове (*binary*) число. Отже, спостерігаємо повну аналогію зі створенням багаторозрядного десяткового числа з декількох арабських цифр. Кожна подальша позиція у двійковому рядку має вдвічі більшу "вагу", ніж попередня позиція, а отже, двійкова система числення – це система з основою 2. У двійковому числі "вага"

кожної позиції збільшується (так само, як і в десятковому – справа наліво) таким чином: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536 і т. д. Довільне N-розрядне двійкове число може представляти одну з 2^N цифрових комбінацій: 0, 1, 2, ..., 3, 2^N-1.

Кожне двійкове число (подібно десятковому) можна записати у вигляді суми цифр (кожна з яких дорівнює 0 або 1), що складають це число, помножених на "вагу" розряду кожної цифри (відповідний ступінь 2):

$$1010_2 = 1\times2^3+0\times2^2+1\times2^1+0\times2^0=8_{10}+2_{10}=10_{10}.$$

Оскільки у двійковій системі використовуються тільки дві цифри 0 та 1, то множення в наведеному вище виразі має дуже простий зміст: відповідний ступінь 2 або входить до суми (якщо цифра розряду дорівнює 1), або ні (якщо цифра розряду дорівнює 0).

У табл. 1 наведено одно-, дво-, три- і чотирибітові двійкові числа та їхні десяткові і шістнадцяткові (див. далі) еквіваленти.

Таблиця 1

Одно-бітове двійкове число	Двобітове двійкове число	Трибітове двійкове число	Чотирибітове двійкове число	Однорозрядне шістнад-цяткове число	Десятковий еквівалент
0	00	000	0000	0	0
1	01	001	0001	1	1
	10	010	0010	2	2
	11	011	0011	3	3
		100	0100	4	4
		101	0101	5	5
		110	0110	6	6
		111	0111	7	7
			1000	8	8
			1001	9	9
			1010	A	10
			1011	B	11
			1100	C	12
			1101	D	13
			1110	E	14
			1111	F	15

1.1.3. Шістнадцяткова система числення

Використання довгих двійкових чисел (кількість розрядів 16, 32, 64, 128 і більше) при проектуванні цифрових систем дуже незручне і збільшує ймовірність виникнення помилок. Однак довге двійкове число можна розбити на групи по чотири біти, кожна з яких представляє одну із $2^4 = 16$ цифрових комбінацій. Зазвичай при проектуванні цифрових систем і розробці програмного забезпечення зручно використовувати систему числення за основою 16, яка називається *шістнадцятковою* (*hexadecimal*). Як показано в табл. 1 для запису шістнадцяткових чисел використовуються арабські цифри від 0 до 9 і букви від A до F. У шістнадцятковому числі "вага" кожної позиції змінюється таким чином: 1, 16, $16^2 = 256$, $16^3 = 4096$ і т. д. Наприклад:

$$3BA_{16} = 3\times16^2+11\times16^1+10 = 954_{10}.$$

1.1.4. Байти та слова

Група із восьми бітів називається *байтом* (*byte*). Байт представляє $2^8 = 256$ цифрових комбінацій. Обсяг пам'яті комп'ютера зазвичай вимірюється в байтах (Б) або кілобайтах (КБ), мегабайтах (МБ) чи гігабайтах (ГБ). Група із чотирьох бітів (половина байта) називається півбайтом (*nibble*). Півбайт становить $2^4 = 16$ цифрових комбінацій, тобто відповідає одному розряду шістнадцяткового числа. Дворозрядне шістнадцяткове число займає 1 Б.

Процесори комп'ютерів звертаються до пам'яті й обробляють дані блоками, які називають словами (*word*). Розмір слова визначається архітектурою конкретного процесора. Більшість сучасних процесорів використовують блоки (слова) розміром 32 біти (4 Б) чи 64 біти (8 Б).

У межах однієї групи бітів довільної довжини (можливо в байті чи в слові) кінцевий біт, розташований на одному кінці цієї групи (зазвичай правому), називається *найменш значимим бітом* (*least significant bit, lsb*), або *молодшим бітом* (він має найменшу "вагу"), а біт на іншому кінці групи називається *найбільш значимим бітом* (*most significant bit, msb*), або *старшим бітом* (він має найбільшу "вагу").

1.1.5. Додавання двійкових чисел

Додавання багаторозрядних двійкових чисел проводиться так само, як і додавання десяткових чисел. Як і при додавання десяткових чисел, якщо сума двох відповідних розрядів доданків перевищує значення, яке вміщується в одному розряді суми, виникає перенесення 1 у наступний розряд:

Перенесення		1 1
Доданок 1		1 0 1 1
Доданок 2	+	0 0 1 1
Сума		1 1 1 0

Біт, доданий до старшого розряду, називається *бітом перенесення* (*carry bit*). У комп'ютерах для збереження чисел та операцій з ними зазвичай зарезервовано певну фіксовану кількість розрядів. Випадок, коли результат складання перевищує виділену для нього кількість розрядів, називають *переповненням* (*overflow*). Чотирибітовий елемент пам'яті, наприклад, може зберігати значення в діапазоні [0, 15]. Переповнення виникає, якщо результат складання перевищує 15. У цьому разі додатковий п'ятий біт відкидається (оскільки немає місця для його зберігання), а результат, що залишився в 4 бітах, буде помилковим. Наприклад:

Перенесення		1 1 1
Доданок 1		1 1 1 0
Доданок 2	+	0 1 1 1
Сума		1 0 1 0 1

Переповнення можна виявити, якщо стежити за перенесенням біта з найбільш значимого розряду двійкового числа (у нашому випадку з четвертого розряду).

1.1.6. Двійкові числа зі знаком

Ми розглянули двійкові числа без знака (*unsigned*), тобто тільки додатні числа. Проте зазвичай використовуються як додатні, так і від'ємні числа. Для роботи з такими двійковими чис-

лами знадобиться ще один розряд для зберігання знака. Існують декілька способів представлення двійкових чисел зі знаком (*signed*). Найбільш поширене застосування мають два з них: прямий (*Sign/Magnitude*) і доповняльний (*Two's Complement*) коди.

Прямий код. Представлення від'ємних двійкових чисел з використанням прямого коду дуже просте і знайоме, оскільки воно збігається з традиційним способом запису від'ємних чисел у математиці: спочатку записуються знак "мінус" а далі – абсолютне значення числа. Якщо двійкове число в прямому коді складається із N бітів, то його найбільш значимий біт використовується для знака, а інші N–1 біти для запису абсолютного значення цього числа. Якщо найбільш значимий біт дорівнює 0, то число додатне. Якщо найбільш значимий біт дорівнює 1, то число від'ємне. Двійкова змінна в прямому коді, яка складається із N бітів, може зберігати число в діапазоні $[-2^{N-1}+1, 2^{N-1}-1]$. Особливістю прямого коду є наявність двох різних наборів бітів (напр., 1000_2 та 0000_2), що відповідають одному числу 0. Це ускладнює роботу з такими числами і може призвести до помилок.

Недоліком прямого коду є те, що описаний у п. 1.1.5 стандартний спосіб додавання двійкових чисел не працює для двійкових чисел зі знаком, записаних у прямому коді. Наприклад, сума $1110_2 + 0110_2$ має дорівнювати 0, однак стандартний алгоритм додавання дає абсурдний результат $1110_2 + 0110_2 = 10100_2$.

Доповняльний код. Представлення додатних і від'ємних чисел у доповняльному коді аналогічне представленню додатних двійкових чисел без знака (див. п. 1.1.2). Єдина, але дуже суттєва, відмінність полягає в тому, що для доповняльного коду вага найбільш значимого біта від'ємна і дорівнює -2^{N-1} (а не 2^{N-1}, як було для двійкового числа без знака). Використання доповняльного коду, яке забезпечує однозначне представлення нуля і допускає додавання чисел за звичною схемою, розглянутою в п. 1.1.5. Таким чином, доповняльний код позбавлений усіх недоліків прямого коду.

Число 0 у доповняльному коді представлено нулями в усіх розрядах двійкового числа: $00\ldots000_2$. Найбільше додатне число, яке можна представити при використанні N розрядів у доповняльного коду дорівнює $2^{N-1}-1$. Воно представлено нулем у най-

більш значимому розряді та одиницями в усіх інших розрядах двійкового числа: $01\ldots111_2$. Максимальне (за модулем) від'ємне значення має одиницю в найбільш значимому розряді і нулі в усіх інших розрядах: $10\ldots000_2 = -2^{N-1}$. Число -1_{10} представлено одиницями в усіх розрядах двійкового числа: $11\ldots111_2$.

Найбільш значимий розряд для всіх додатних чисел дорівнює 0, а для всіх від'ємних – 1, тобто найбільш значимий біт доповняльного коду можна розглядати як аналог знакового біта прямого коду. Однак всі інші біти доповняльного коду інтерпретуються не так, як біти прямого коду.

Для зміни знака числа на протилежний при використанні доповняльного коду необхідно виконати спеціальну операцію, яка називається *обчисленням доповнення до двох* (*taking the two's complement*). Її суть полягає в тому, що всі біти числа інвертуються, а потім до найменш значимого біта додається 1. Ця операція дозволяє знайти двійкове представлення негативного числа або визначити його абсолютне значення.

Доповнення числа 0 до двох також проводиться за загальним правилом: інвертування всіх бітів (це дає $11\ldots111_2$) і подальше збільшення на 1, що робить значення всіх бітів такими, що дорівнюють 0. При цьому перенесення найбільш значимого біта ігнорується. У результаті число 0 завжди представлене набором тільки нульових бітів. На відміну від прямого коду, доповняльний код не має від'ємного нуля. Нуль вважається додатним числом, оскільки його знаковий біт дорівнює 0.

Розглянемо, наприклад, представлення числа -6_{10} як чотирибітового числа в доповняльному коді. Спочатку треба представити число $+6_{10}$ у двійковому вигляді $+6_{10} = 0110_2$. Далі необхідно проінвертувати всі біти: 1001_2 і додати 1: $1001_2 + 1_2 = 1010_2$. Таким чином, у доповняльному коді $-6_{10} = 1010_2$.

Від'ємне число 1101_2 дорівнює -3_{10}. Це можна показати двома шляхами:

1) оскільки ваги розрядів дорівнюють –8, 4, 2, 1, то

$$1101_2 = -8 + 4 + 0 + 1 = -3_{10};$$

2) оскільки найбільш значимий розряд числа 1101_2 дорівнює 1, то число від'ємне. Щоб знайти модуль, необхідно проінверту-

вати всі біти (0010_2) і додати 1 ($0010_2 + 1_2 = 0011_2 = 3_{10}$). Таким чином, $1101_2 = -3_{10}$.

Основною перевагою використання доповняльного коду є те, що звичний спосіб додавання двійкових чисел (див. п. 1.1.5) працює як для додатних, так і для від'ємних чисел. При цьому біт перенесення з найбільш значимого розряду двійкового числа ігнорується. Наприклад:

$$-2_{10} + 4_{10} = 1110_2 + 0100_2 = 10010_2 = 0010_2 = 2_{10}$$

(п'ятий біт суми відкидається),

$$-7_{10} + 7_{10} = 1001_2 + 0111_2 = 10000_2 = 0000_2 = 0_{10}$$

(п'ятий біт суми відкидається).

Віднімання двох двійкових чисел здійснюється шляхом інверсії знака від'ємника за допомогою операції обчислення доповнення до двох та подальшого звичайного додавання результату цієї операції та зменшуваного. Наприклад:

$$5_{10} - 3_{10}\text{: } 5_{10} = 0101_2, -3_{10} = 1101_2;$$
$$5_{10} - 3_{10} = 5_{10} + (-3_{10}) = 0101_2 + 1101_2 = 0010_2 = 2_{10},$$
$$3_{10} - 7_{10}\text{: } 3_{10} = 0011_2, -7_{10} = 1001_2;$$
$$3_{10} - 7_{10} = 3_{10} + (-7_{10}) = 0011_2 + 1001_2 = 1100_2 = -4_{10}.$$

Так само, як і двійкове число без знака, довільне N-бітове число, записане в доповняльному коді, може набути одного з $2N$ можливих значень. Проте весь цей діапазон розділений між додатними та від'ємними числами. Діапазон N-бітового числа в доповняльному коді дорівнює $[-2^{N-1}, 2^{N-1}-1]$. Оскільки в доповняльному коді відсутній від'ємний нуль, то у від'ємній частині діапазону міститься на одне число більше, ніж у додатній.

При використанні доповняльного коду додавання двох додатних або від'ємних N-бітових чисел може привести до переповнення, якщо результат буде більшим, ніж $2^{N-1}-1$, або меншим, ніж -2^{N-1}. Додавання додатного та від'ємного чисел навпаки, ніколи не приводить до переповнення. На відміну від додавання двійкових чисел без знака, поява перенесення з найбільш значимого біта не є ознакою переповнення. Замість цього індикатором переповнення є випадок, коли після додавання двох чисел з однаковими знаками знаковий біт суми не збігається зі знаковими бітами доданків. Наприклад, $6_{10} + 3_{10} = 0110_2 + 0011_2 = 1001_2 = -7_{10}$.

У результаті додавання двох додатних чисел був отриманий від'ємний результат, тобто відбулося переповнення. Дійсно правильна відповідь 9_{10}, вона лежить за межами діапазону $[-8_{10}, 7_{10}]$ чотирибітового числа в доповняльному коді. При використанні більшої кількості бітів (5 або більше) результат буде правильним $01001_2 = 9_{10}$.

Для збільшення кількості бітів представлення довільного числа, записаного в доповняльному коді, значення знакового біта має бути скопійоване в усі нові старші розряди модифікованого числа. Цю операцію називають *знаковим розширенням* (*sign extension*). Наприклад, числа 5_{10} і -5_{10} записуються в чотирибітовому доповняльному коді як 0101_2 та 1011_2, відповідно. Для збільшення кількості розрядів до семи, ми повинні скопіювати знаковий біт у три нових найбільш значимих біти модифікованого числа:

$$5_{10} = 0000101_2,$$
$$-5_{10} = 1111011_2.$$

Порівняння способів представлення двійкових чисел. Найпоширенішими способами представлення двійкових чисел є двійкові числа без знака, двійкові числа зі знаком у прямому коді і в доповняльному коді. При використанні N бітів діапазони цих представлень будуть дорівнювати $[0, 2^N-1]$, $[-2^{N-1}+1, 2^{N-1}-1]$, $[-2^{N-1}, 2^{N-1}-1]$, відповідно. Недоліком прямого коду є наявність двох комбінацій бітів, що відповідають одному числу 0, і неможливість використання стандартного алгоритму додавання. Перевагою доповняльного коду є можливість його використання для представлення як додатних, так і від'ємних цілих чисел, для всіх них працює звичний спосіб додавання. Віднімання здійснюється шляхом інверсії знака від'ємника (тобто шляхом доповнення цього числа до двох) і подальшого додавання зі зменшуваним. На практиці це найбільш поширене представлення, і тому саме його і використаємо в нашому посібнику.

1.1.7. Дробові числа

Для розв'язання багатьох практичних задач комп'ютер має працювати як з цілими, так і з дробовими числами. Представлення знакових і беззнакових цілих чисел ми вже розглянули, а

Тепер розглянемо раціональні числа, які в комп'ютері обробляються у двох представленнях: із фіксованою та плаваючою точкою. Числа із фіксованою точкою – це аналог десяткових чисел; деякі біти представляють цілу частину, а інші – дробову. Числа із плаваючою точкою (або із плаваючою комою) є аналогом експоненціального представлення числа з мантисою та порядком, яке часто використовується в науковій літературі.

Числа із фіксованою точкою. У представленні *із фіксованою точкою* біти цілої та дробової частини розділені двійковою точкою, так само як ціла і дробова частини звичайного десяткового числа розділені десятковою точкою. Положення цієї точки, а отже, і кількість бітів цілої та дробової частини, зафіксоване, тобто його не треба зберігати і передавати. Положення двійкової точки визначається домовленістю між користувачами та програмістами, які інтерпретують це число. Наприклад, якщо для цілої та дробової частини виділено по 4 біти, то перші чотири біти (0101) числа 01011100 становлять його цілу частину, а інші (1100) дробову. Це число дорівнює $2^2+1+2^{-1}+2^{-2}=5.75_{10}$. Для представлення знакових чисел із фіксованою точкою можна використовувати як прямий, так і доповняльний код.

Як приклад розглянемо представленням числа -3.75_{10} із фіксованою точкою, кількість бітів цілої та дробової частини 4 як у прямому, так і доповняльному коді. Абсолютне значення числа дорівнює 00111100_2. У прямому коді найбільш значимий біт використовується для знака, а отже, у прямому коді $-3.75_{10}=10111100_2$. Доповняльний код двійкового числа отримуємо інверсією бітів абсолютного значення і додаванням 1 до наймолодшого розряду числа, тобто молодшого розряду дробової частини, у нашому прикладі він має вагу 2^{-4}, отже $-3.75_{10}=11000100_2$.

Для коректних обчислень при використанні чисел із фіксованою точкою використовується двійкове представлення в доповняльному коді. Наприклад, суму чисел 0.75_{10} та -0.625_{10} можна порахувати таким чином. Якщо для цілої та дробової частини використовується по 4 біти, то в доповняльному коді $0.75_{10}=00001100$, $-0.625_{10}=11110110_2$. Сума $00001100_2+11110110_2=100000010_2=0.125_{10}$ обчислюється за загальними правилами, біт перенесення із найбільш значимого розряду (як і в п. 1.1.6) ігнорується.

Числа із плаваючою точкою. *Числа із плаваючою точкою* відповідають експоненціальному представленню. У цьому представленні відсутні обмеження, зумовлені наявністю тільки фіксованої кількості цілих і дробових бітів, тому воно дозволяє представляти дуже великі і дуже малі числа. Числа із плаваючою точкою мають знак, мантису (M), основу (B) і порядок (E):

$$\pm M \times B^E .$$

Наприклад, число 4.03×10^6 є десятковим експоненціальним представленням числа 4030000, де мантисою є 4.03, основа дорівнює 10, а порядок дорівнює 6. Аналогічно, $4.03 \times 10^{-6} = 0.00000403$. Положення десяткової точки визначається порядком. У двійкових чисел із плаваючою точкою основа дорівнюватиме 2, а мантиса і порядок будуть двійковими числами.

У поширеному форматі чисел із плаваючою точкою 32 біти використовуються таким чином: 1 знаковий біт, 8 бітів для збереження порядку і 23 біти для мантиси. У двійкових числах із плаваючою точкою перший біт мантиси (зліва від точки) завжди дорівнює 1 (т. зв. неявна старша одиниця), і тому його можна не зберігати, а 23 біти будуть використовуватися для інших розрядів мантис. Порядок може бути як додатним, так і від'ємним. У форматі з плаваючою точкою використовується так званий зміщений порядок, який дорівнює сумі первинного порядку (додатного чи від'ємного) плюс постійне зміщення. Для 32-бітових чисел із плаваючою точкою, у яких порядок складається із 8 бітів, використовується зміщення $2^7-1 = 127$, а отже, зміщений порядок не може бути від'ємним. Наприклад, якщо порядок дорівнює 10, зміщений порядок дорівнює $10 + 127 = 137 = 10001001_2$, для порядку –20 зміщений порядок буде $-20 + 127 = 107 = 01101011_2$.

Як приклад розглянемо представлення числа 9.5_{10} в описаному форматі з плаваючою точкою $9.5_{10} = 1001.1_2 = 1.0011_2 \times 2^3$. З урахуванням неявної старшої одиниці 23 біти мантиси будуть дорівнювати 00110000000000000000000. Зміщений порядок дорівнюватиме $3_{10} + 127_{10} = 130_{10} = 10000010_2$. Отже, число 9.5_{10} буде представлене у вигляді:

Знак: 1 біт	Зміщений порядок: 8 бітів	Мантиса (є неявна старша одиниця): 23 біти
0	10000010	00110000000000000000000

Це представлення відповідає стандарту IEEE 754, прийнятому в 1985 р. Інститутом інженерів електротехніки та електроніки (*Institute of Electricaland Electronics Engineers, IEEE*).

Особливі випадки: 0, ±∞, NaN. Стандарт IEEE для чисел із плаваючою точкою включає особливі випадки представлення таких випадків, як число 0, нескінченність і неприпустимі результати. Наприклад, представити число 0 у вигляді числа з плаваючою точкою неможливо через наявність неявної старшої одиниці. Для таких випадків зарезервовано спеціальні коди: порядок складається тільки з нулів або одиниць. Представлення з плаваючою точкою (як і представлення двійкових чисел у прямому коді) має два 0: додатний і від'ємний. Позначення NaN використовується для результатів некоректних операцій, наприклад $\sqrt{-1}, \log(-1)$. Представлення 0, +∞, –∞ та NaN відповідно до стандарту IEEE 754 наведено в табл. 2.

Таблиця 2

Число	Знак	Порядок	Мантиса
0	X	00000000	00000000000000000000000
+∞	0	11111111	00000000000000000000000
–∞	1	11111111	00000000000000000000000
NaN	X	11111111	Не нуль

Формати одинарної та подвійної точності. Окрім уже розглянутого 32-бітового формату числа із плаваючою точкою (який також називають *форматом одинарної точності*) використовується 64-бітовий формат подвійної точності. Він дозволяє представити ширший діапазон чисел з великою точністю. У табл. 3 наведено характеристики цих форматів.

Таблиця 3

Формат	Загальна кількість бітів	Кількість бітів знака	Кількість бітів порядку	Кількість бітів мантиси
single	32	1	8	23
double	64	1	11	52

Характеристики діапазону і точності форматів одинарної та подвійної точності чисел із плаваючою точкою наведено в табл. 4.

Таблиця 4

Формат	Діапазон (приблизний)	Точність, десяткових розрядів
single	$\pm 1.175494 \times 10^{-38}$ $\pm 3.402824 \times 10^{38}$	7
double	$\pm 2.22507385850720 \times 10^{-308}$ $\pm 1.79769313486232 \times 10^{308}$	15

Додавання чисел із плаваючою точкою. Додавання чисел із плаваючою точкою є досить складним і повільним процесом. Для його здійснення необхідно виконати такі дії:

1) виділити біти порядку і мантиси доданків;

2) приєднати неявну старшу одиницю до мантис обох доданків;

3) порівняти порядки і зсунути мантису числа, що має менший порядок;

4) скласти мантиси як звичайні двійкові числа;

5) за необхідності нормалізувати мантису і порядок;

6) округлити результат.

Результати обчислень, які виходять за межі доступної точності, необхідно округлювати до найближчого доступного числа. Існують декілька методів округлення числа: округлення в менший бік, округлення в більший бік, округлення в бік нуля й округлення до найближчого числа. Зазвичай використовують округлення до найближчого числа, що відповідає загальноприйнятій практиці в математиці.

1.2. Логічні елементи

Для фактичного виконання операцій над двійковими змінними використовують спеціальні електронні схеми, які називаються *цифровими*. Схеми, які виконують найпростіші операції, називають *логічними елементами* (*logic gate*). Складні цифрові схеми скрадаються з великої кількості таких логічних елементів. На входи (чи на один вхід) логічного елемента надходить один або декілька двійкових сигналів, за яким на виході формується новий двійковий сигнал. На принципових схемах вхідні сигнали зазвичай розміщують ліворуч (чи згори) від зображення елемента, а вихідні сигнали – праворуч (чи знизу). Для позначення вхідних сигналів зазвичай використовують букви (*A*, *B*, *C*,...), а для вихідного сигналу – букву *Y*. Взаємозв'язок між вхідними та вихідним сигналами можна описати за допомогою *таблиці істинності* (*truth table*), у якій наведено всі можливі комбінації значень вхідних сигналів і відповідні значення вихідного сигналу, або за допомогою виразу булевої алгебри.

1.2.1. Логічний елемент НІ

Логічний елемент НІ (або інвертор, NOT *gate*, рис. 1) має один вхід *A* і один вихід *Y*. Вихідний сигнал *Y* дорівнює інвертованому вхідному сигналу *A*, тобто, якщо сигнал на вході *A* – це НЕПРАВДА, то сигнал на виході *Y* буде ІСТИНА, і навпаки. Таблицю істинності та булеве рівняння наведено на рис. 1. Маленьке коло на графічному зображенні елемента є позначенням заперечення (інверсії). Вираз $\bar{A}$ означає інверсію A. На рис. 1, *в* зображено схемотехнічну реалізацію елемента НІ з використанням КМОН-технології. Принцип роботи схеми дуже простий: якщо на її вхід надходить високий рівень напруги (логічна 1), то відривається нижній польовий транзистор і на виході елемента формується низький рівень напруги (логічний 0), коли на вхід надходить низький рівень напруги (логічний 0), то відривається верхній польовий транзистор і на виході елемента формується високий рівень напруги (логічна 1).

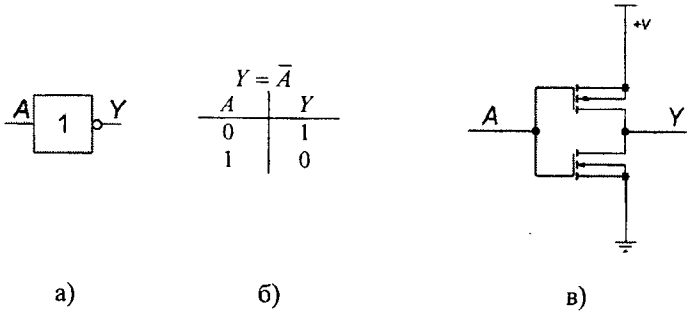


Рис. 1. Логічний елемент НІ: позначення (а), логічна функція і таблиця істинності (б), реалізація з використанням КМОН-технології (в)

1.2.2. Буфер

Буфер повторює вхідний сигнал на своєму виході. Оскільки він не виконує жодного перетворення вхідного сигналу, то може здатися, що він зайвий. На практиці буфер може забезпечити характеристики, необхідні для функціонування реального пристрою. Він може, наприклад, віддавати великий струм на електромагніт або двигун і тощо. Реалізація буфера (рис. 2, *в*) з використанням КМОН-технології являє собою два послідовно з'єднані елементи НІ.

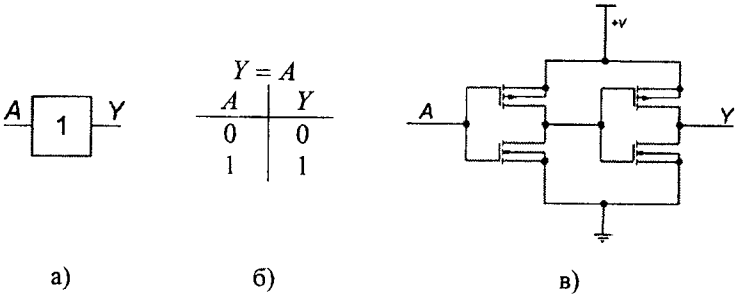


Рис. 2. Буфер: позначення (а), логічна функція і таблиця істинності (б), реалізація з використанням КМОН-технології (в)

1.2.3. Логічний елемент ТА

Логічні елементи з двома вхідними сигналами дозволяють реалізувати складні булеві функції. На виході логічного елемента ТА (*AND gate*, рис. 3), значення ІСТИНА з'являється тоді і тільки тоді, коли обидва вхідні сигнали A і B мають значення ІСТИНА. Булеве рівняння для логічного елемента ТА можна записати у вигляді $Y = A * B$. На рис. 3, *в* зображено схемотехнічну реалізацію елемента ТА з використанням КМОН-технології. Ця реалізація складається з двох послідовно з'єднаних елементів: ТА-НІ (він буде описаний далі) та інвертора. Якщо на один (чи обидва) вхід схеми надходить низький рівень напруги (логічний 0), то відривається один (чи обидва) верхній польовий транзистор і в точці C встановлюється логічна 1, яка надходить на вихідний інвертор, його вихід набуває значення 0. Коли на обидва входи схеми надходить високий рівень напруги (логічна 1), то відриваються обидва нижні польові транзистори і в точці C встановлюється логічний 0. При цьому вихідний інвертор формує логічну 1 у точці Y.

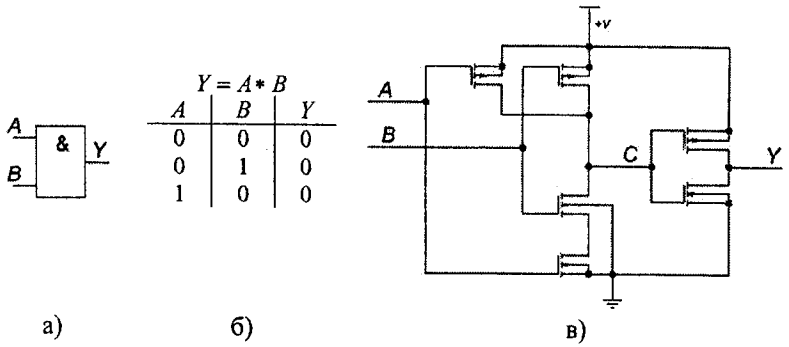
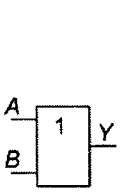


Рис. 3 Логічний елемент ТА: позначення (а), логічна функція і таблиця істинності (б), реалізація з використанням КМОН-технології (в)

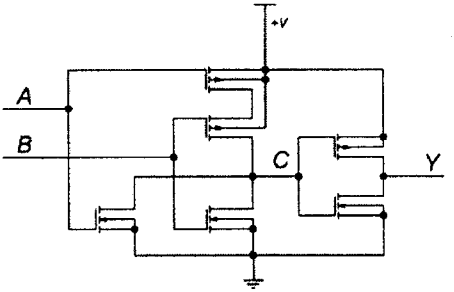
1.2.4. Логічний елемент АБО

Логічний елемент АБО (*OR gate*, рис. 4) видає значення ІСТИНА на вихід *Y*, якщо хоча б один із двох вхідних сигналів *A* або *B* має значення ІСТИНА. Булеве рівняння для логічного елемента АБО має вигляд $Y = A + B$. На рис. 4, *в* наведено реалізацію елемента АБО. Ця реалізація складається з двох послідовно з'єднаних елементів: АБО-НІ (він буде описаний далі) та інвертора. Якщо на обидва входи схеми надходить низький рівень напруги (логічний 0), то відриваються обидва верхні польові транзистори і в точці *C* встановлюється логічна 1, яка надходить на вихідний інвертор, а отже, $Y = 0$. Коли на один (чи обидва) входи схеми надходить високий рівень напруги (логічна 1), то відривається один (чи обидва) нижній польовий транзистор і в точці *C* встановлюється логічний 0, при цьому $Y = 1$.



$Y = A + B$

A	*B*	*Y*
0	0	0
0	1	1
1	0	1
1	1	1



а) б) в)

Рис. 4. Логічний елемент АБО: позначення (а), логічна функція та таблиця істинності (б), реалізація з використанням КМОН-технології (в)

1.2.5. Інші логічні елементи

На рис. 5–7 наведено деякі інші найпоширеніші логічні елементи з двома входами. Додавання кола на виході будь-якого логічного елемента означає інверсію його вихідного сигналу. Таким чином, елемент ТА можна перетворити на ТА-НІ (*NAND gate*, рис. 5) Значення вихідного сигналу *Y* такого елемента буде ІСТИНА завжди, окрім випадку, коли обидва вхідні сигнали *A* і *B* не дорівнюють ІСТИНІ, тоді воно стає НЕПРАВДА. Аналогічно, маючи логічний елемент АБО, можна створити елемент АБО-НІ (*NOR gate*, рис. 6).

На рис. 5, *в* зображено схемотехнічну реалізацію елемента ТА-НІ з використанням КМОН-технології. Якщо на один (чи обидва) вхід схеми надходить низький рівень напруги (логічний 0), то відривається один (чи обидва) верхній польовий транзистор і в точці *C* встановлюється логічна 1. Коли на обидва входи схеми надходить високий рівень напруги (логічна 1), то відриваються обидва нижні польові транзистори і в точці *C* встановлюється логічний 0. Видно, що елемент ТА-НІ складається з меншої кількості транзисторів, ніж елемент ТА, тому в основному саме його намагаються використовувати. Схему елемента АБО-НІ наведено на рис. 6, *в*.

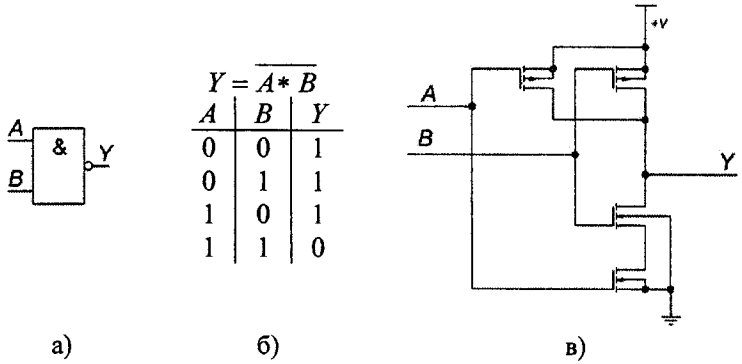


Рис. 5. Логічний елемент ТА-НІ: позначення (а), логічна функція і таблиця істинності (б), реалізація з використанням КМОН-технології (в)

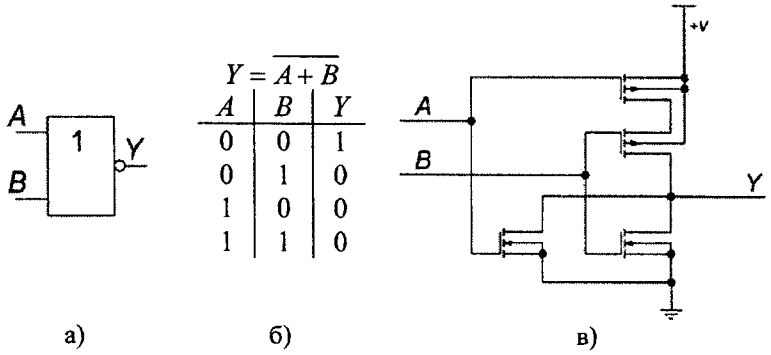
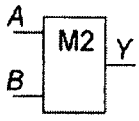


Рис. 6. Логічний елемент АБО-НІ: позначення (а), логічна функція і таблиця істинності (б), реалізація з використанням КМОН-технології (в)

На практиці також застосовується елемент обчислення суми за модулем 2 з кількістю входів N. Такий елемент видає на вихід сигнал ІСТИНА, якщо непарна кількість його вхідних сигналів має значення ІСТИНА. Цей елемент виконує функцію контролю за парністю. На рис. 7 показано такий елемент із двома входами. У випадку двох входів його називають також елементом НЕРІВНОЗНАЧНОСТІ, або елементом ВИКЛЮЧНЕ АБО.



а)

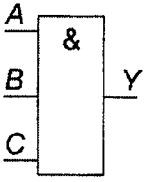
$Y = A \oplus B$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

б)

Рис. 7. Логічний елемент обчислення суми за модулем 2: позначення (а), логічна функція і таблиця істинності (б)

Досить поширеними є елементи з кількістю входів 3 або більше. Приклади таких елементів разом із відповідними таблицями істинності та булевими рівняннями наведено на рис. 8 (ТА) і рис. 9 (АБО).



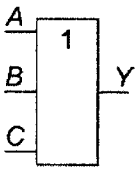
а)

$Y = A * B * C$

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

б)

Рис. 8. Логічний елемент ТА з трьома входами: позначення (а), логічна функція і таблиця істинності (б)



а)

$Y = A + B + C$

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

б)

Рис. 9. Логічний елемент АБО з трьома входами: позначення (а), логічна функція та таблиця істинності (б)

У випадку трьох (або більше) входів функції нерівнозначності (ІСТИНА, коли всі входи є однаковими), виключне АБО (ІСТИНА, коли один і тільки один вхід ІСТИНА) та сума за модулем 2 (і відповідні елементи) повністю відмінні. Це ілюструється таблицею істинності (табл. 5).

Таблиця 5

A	*B*	*C*	Сума за модулем 2	Нерівнозначність	Виключне АБО
0	0	0	0	0	0
0	0	1	1	1	1
0	1	0	1	1	1
0	1	1	0	1	0
1	0	0	1	1	1
1	0	1	0	1	0
1	1	0	0	1	0
1	1	1	1	0	0

1.2.6. Невизначене значення і третій стан

Булева алгебра використовує тільки два значення 0 і 1, яким відповідають чітко визначені діапазони напруги на входах і виходах логічних елементів. Проте в реальних схемах

можуть також виникати неприпустимі щодо булевої логіки стани, які не відповідають логічним 0 чи 1. Наприклад, при використанні традиційної КМОН-технології діапазон вхідних напруг 1,35–3,15 В заборонений, такі напруги не відповідають ні 0, ні 1, але вони можуть з'являтися в реальних системах. Для позначення таких неприпустимих або невідомих значень використовується символ *X*. Такий випадок може реалізуватися, якщо, наприклад, вхід елемента під'єднано до виходів двох інших елементів, які намагаються встановити протилежні логічні рівні (рис. 10). Цей випадок називають *змаганням*, або *конфліктом* (*contention*), здебільшого він помилковий і його слід уникати.

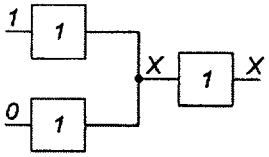


Рис. 10. Неприпустиме значення

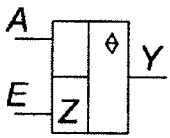
Реальна значення напруги на вході з конфліктом залежить від співвідношення вихідних опорів елементів, що видають у ділянку напруги логічного 0 та 1. Зазвичай (але не завжди) результуюче значення напруги на вході елемента опиняється в "забороненій" зоні, а його поведінка стає непередбачуваною. Змагання також може стати причиною підвищеного споживання енергії конфліктуючими елементами, унаслідок чого схема нагрівається і може бути пошкоджена.

При розробці цифрових схем також використовують символ "–" для позначення в таблицях істинності байдужих змінних, від яких не залежить стан виходів. Приклад використання байдужого значення наведено в підрозд. 3.3.

У цифровій схемотехніці зазвичай використовують шини. Наприклад, мікропроцесор, відеоконтролер і контролер жорсткого диска можуть потребувати взаємодії з підсистемою пам'яті в персональному комп'ютері. Кожна такий вузол може підключатися до загальної шини пам'яті. При цьому в деякий момент часу тільки один вузол має право видати напругу на шину. Ви-

ходи інших вузлів мають бути відключені від шини, щоб не стати причиною колізії з вузлом, що здійснює обмін даними з пам'яттю. Таким чином, значення напруги на шині можуть встановити декілька елементів, але при коректній роботі шини є тільки один елемент, який встановлює цю напругу, а всі інші перебувають у високоімпедансному, або відключеному стані. Такий стан ще називають третім станом і позначають символом *Z*. Якщо вихід елемента перебуває в стані *Z*, то це означає що він не керує станом кола, під'єднаного до виходу елемента. Поява стану *Z* далеко не завжди вказує на наявність помилки в роботі схеми. Наприклад, інший елемент схеми може подавати на електричне коло допустиму напругу логічного рівня.

Буфер з трьома станами (рис. 11), має три можливих вихідних значення: ВИСОКИЙ рівень (1), НИЗЬКИЙ рівень (0) і відключений або плаваючий (*Z*) стан. Буфер із трьома станами має вхід *A*, вихід *Y* і сигнал керування *E*. Коли сигнал дозволу (керування) має значення ІСТИНА, буфер з трьома станами працює як простий буфер, передаючи вхідне значення на вихід. Коли сигнал управління має значення НЕПРАВДА, вихід буфера перемикається в третій стан і стає плаваючим (*Z*).



а)

E	*A*	*Y*
0	0	*Z*
0	1	*Z*
1	0	0
1	1	1

б)

Рис. 11. Буфер із трьома станами: позначення (а) і таблиця істинності (б)

2. КОМБІНАЦІЙНІ ЛОГІЧНІ СХЕМИ

Цифрові схеми поділяють на два великих класи: *комбінаційні* (*combinational*) і *послідовнісні* (*sequential*) *схеми*. Вихідні сигнали комбінаційних схем залежать тільки від поточних значень вхідних сигналів. Іншими словами, такі схеми комбінують поточні значення вхідних сигналів для обчислення значення на виході. Логічні елементи – це комбінаційні схеми. Виходи послідовнісних схем залежать як від поточних, так і від попередніх значень вхідних сигналів, тобто від послідовності зміни вхідних сигналів, а отже, вони мають пам'ять про минулий стан. У комбінаційній схемі, на відміну від послідовнісної, пам'ять відсутня. Властивості та методи проектування комбінаційних і послідовнісних схем істотно різняться. Розглянемо спочатку комбінаційні схеми, а далі послідовнісні.

2.1. Булева алгебра

Оскільки входи і виходи цифрових схем мають два відмінних стаціонарних стани, то булеві функції та рівняння, що використовують змінні, які набувають двох значень ІСТИНА (1) або НЕПРАВДА (0), ідеально підходять для опису таких систем. Булева функція та всі її аргументи набувають значень 0 та 1, тобто множина можливих комбінацій значень на входах скінченна, і тому найпростішим і наочним методом задання булевої функції є табличний. Іншим способом задання булевої функції є аналітичний, де використовуються три базисні булеві функції: НІ, ТА, АБО.

Найпростішою нетривіальною булевою функцією є функція НІ, або інверсія, доповнення чи функція заперечення. Для її позначення використовується верхня риска, $Y = \overline{A}$. Якщо аргумен-

том функції є ІСТИНА, то функція НІ дорівнює НЕПРАВДА, і навпаки. Отже, ця функція описується таблицею істинності (табл. 6).

Таблиця 6

A	$Y = \overline{A}$
0	1
1	0

Функція ТА декількох аргументів називається *кон'юнкцією*, або добутком (*product*), $Y = A * B$. Результат функції дорівнює ІСТИНА тоді і тільки тоді, коли всі її аргументи дорівнюють ІСТИНА. Вона має таблицю істинності (табл. 7).

Таблиця 7

A	B	$Y = A * B$
0	0	0
0	1	0
1	0	0
1	1	1

Інша базисна функція АБО, чи *диз'юнкція*, $Y = A + B$, описується таблицю істинності (табл. 8).

Таблиця 8

A	B	$Y = A + B$
0	0	0
0	1	1
1	0	1
1	1	1

Результат функції АБО дорівнює ІСТИНА, якщо хоча б один із її аргументів дорівнює ІСТИНА. Довільну булеву функцію можна виразити, якщо використати описані вище функції НІ, ТА, АБО.

2.1.1. Булеві функції

Тепер розглянемо способи запису булевих функцій довільної кількості аргументів. Деяка змінна та її доповнення називаються *літералом*. Приклади літералів: A, $\overline{A}$, B, $\overline{B}$, C, $\overline{C}$. Операція ТА над одним або декількома літералами називається *імплікантою*, наприклад: $\overline{A}$, $A*\overline{B}$, $A*\overline{B}*\overline{C}$, $\overline{A}*B*C$. *Мінтермом* (*minterm*), елементарною кон'юнктивною формою, називають добуток, до якого входять усі аргументи функції (або їх доповнення). Наприклад, якщо розглядається функція трьох аргументів A, B, C, то $A*\overline{B}*\overline{C}$, $\overline{A}*B*C$ будуть мінтермами, а $A*\overline{B}$ – ні. Аналогічно, операція АБО над одним або декількома літералами називається *диз'юнкцією*, або сумою. *Макстерм* (*maxterm*), елементарна диз'юнктивна форма, є сумою всіх аргументів функції (або їх доповнень); $A+\overline{B}+\overline{C}$ є макстермом функції трьох змінних A, B, C.

Порядок операцій важливий при аналізі та обчисленні значень булевих функцій. У булевих рівняннях найбільший пріоритет має операція НІ, потім йде ТА, потім АБО. Це відповідає звичайним правилам елементарної математики, у якій добутки обчислюються перед сумами. Таким чином,

$$Y = A*B+C = (A*B)+C \neq A*(B+C).$$

Аналізуючи булеві функції, зазвичай використовують так звані диз'юнктивні форми для їх аналітичного запису. Якщо функція має N аргументів, то її таблиця істинності складається з 2^N рядків, по одному для кожної можливої комбінації значень аргументів функції. Кожному рядку в таблиці істинності відповідає мінтерм, який набуває значення ІСТИНА тільки для цього рядка. Як приклад у табл. 9 наведено таблицю істинності для функції виключне АБО трьох аргументів разом з усіма мінтермами. Наприклад, мінтерм другого рядка $\overline{A}*B*\overline{C}$ дорівнює 1 тільки для $A=0, B=1, C=0$, тобто саме для комбінації вхідних аргументів другого рядка, для всіх інших рядків цей мінтерм дорівнює 0.

Таблиця 9

N	*A*	*B*	*C*	Мінтерм	Виключне АБО
0	0	0	0	$\overline{A} * \overline{B} * \overline{C}$	0
1	0	0	1	$\overline{A} * \overline{B} * C$	1
2	0	1	0	$\overline{A} * B * \overline{C}$	1
3	0	1	1	$\overline{A} * B * C$	0
4	1	0	0	$A * \overline{B} * \overline{C}$	1
5	1	0	1	$A * \overline{B} * C$	0
6	1	1	0	$A * B * \overline{C}$	0
7	1	1	1	$A * B * C$	0

Із використанням мінтермів дуже легко записати довільну булеву функцію за її таблицею істинності. Для цього достатньо записати функцію АБО всіх мінтермів, що відповідають тим рядкам таблиці істинності, у яких Y має значення ІСТИНА. Для наведеного вище прикладу це буде сума мінтермів 1, 2 та 4 рядків:

$$Y = \overline{A} * \overline{B} * C + \overline{A} * B * \overline{C} + A * \overline{B} * \overline{C}.$$

Така сума мінтермів називається досконалою *диз'юнктивною нормальною формою* функції (ДНФ, *sum-of-products canonical form*). Вона є сумою мінтермів, що складаються з літералів. Зазвичай мінтерми записують у суми в тому самому порядку, як і відповідні рядки в таблиці істинності. Досконала диз'юнктивна нормальна форма також може бути записана з використанням символу суми Σ і переліку номерів рядків, мінтерми яких входять до суми:

$$Y = \overline{A} * \overline{B} * C + \overline{A} * B * \overline{C} + A * \overline{B} * \overline{C} = \Sigma(1,2,4).$$

Досконала диз'юнктивна нормальна форма дозволяє записати аналітичний вираз для довільної булевої функції будь-якої кількості аргументів, що задана таблицею істинності. На жаль, досконала диз'юнктивна нормальна форма не завжди дозволяє отримати простий і компактний вираз.

Альтернативним способом побудови аналітичного виразу булевої функцій є використання досконалої *кон'юнктивної норма-*

льної форми (*products-of-sum forms*). Кожному рядку таблиці істинності (табл. 10) відповідає *макстерм*, який має значення НЕПРАВДА тільки для цього рядка. Наприклад, макстерм для нульового рядка таблиці істинності функції виключне АБО трьох аргументів має вигляд $A+B+C$, якщо $A=0, B=0, C=0$ цей макстерм дорівнює НЕПРАВДА, при інших значеннях аргументів він буде дорівнювати ІСТИНА. Для будь-якої функції, заданої таблицею істинності, можна записати її аналітичний булевий вираз як функцію ТА всіх макстермів, для яких вихід має значення НЕПРАВДА. Досконалу кон'юнктивну нормальну форму також можна записати, використавши символ добутку Π.

Таблиця 10

	A	*B*	*C*	**Макстерм**	Виключне АБО
0	0	0	0	$A+B+C$	0
1	0	0	1	$A+B+\overline{C}$	1
2	0	1	0	$A+\overline{B}+C$	1
3	0	1	1	$A+\overline{B}+\overline{C}$	0
4	1	0	0	$\overline{A}+B+C$	1
5	1	0	1	$\overline{A}+B+\overline{C}$	0
6	1	1	0	$\overline{A}+\overline{B}+C$	0
7	1	1	1	$\overline{A}+\overline{B}+\overline{C}$	0

Досконала кон'юнктивна нормальна форма функції виключне АБО трьох аргументів має вигляд

$$Y=(A+B+C)*(A+\overline{B}+\overline{C})*(\overline{A}+B+\overline{C})*(\overline{A}+\overline{B}+C)*(\overline{A}+\overline{B}+\overline{C})=$$
$$=\Pi(0,3,5,6,7).$$

Кон'юнктивна нормальна форма і диз'юнктивна нормальна форма повністю еквівалентні. Диз'юнктивна форма дає більш короткий вираз, коли вихід має значення ІСТИНА тільки в невеликій кількості рядків таблиці істинності; кон'юнктивна форма простіша, якщо вихід має значення НЕПРАВДА тільки в декількох рядках таблиці істинності.

2.1.2. Булеві аксіоми та теореми

Для тотожного перетворення та спрощення складних і громіздких булевих виразів можна використовувати булеву алгебру так само, як використовують звичайну алгебру для спрощення математичних виразів. Правила булевої алгебри дуже схожі на правила звичайної алгебри, але в деяких випадках вони простіші, тому що всі змінні мають тільки два можливі значення: 0 або 1. Булева алгебра базується на наборі аксіом, які вважаються безумовно істинними.

Для будь-якого булевого виразу чи рівняння виконується принцип дуальності. Якщо в рівнянні замінити 0 на 1, 1 на 0, функції ТА замінити на АБО, а АБО на ТА, то булеве рівняння залишиться правильним.

У табл. 11 наведено аксіоми булевої алгебри, які визначають поняття булевої змінної, що набуває двох значень 0 та 1 (перша аксіома), і трьох базисних операцій: НІ, ТА, АБО (аксіоми 2–5, які є таблицями істинності відповідних операцій).

Таблиця 11

	Аксіома	Дуальна аксіома
1	Якщо $B \neq 1$, то $B = 0$	Якщо $B \neq 0$, то $B = 1$
2	$\overline{0} = 1$	$\overline{1} = 0$
3	$0 * 0 = 0$	$1 + 1 = 1$
4	$1 * 1 = 1$	$0 + 0 = 0$
5	$0 * 1 = 1 * 0 = 0$	$0 + 1 = 1 + 0 = 1$

За допомогою вказаних аксіом можна довести цілий ряд теорем булевої алгебри, які можна використовувати для спрощення булевих функцій, а отже, для отримання простих і компактних електронних схем, які ці функції реалізують.

У табл. 12 наведено основні булеві теореми, які використовують при проектуванні цифрових приладів для мінімізації булевих функцій. Наприклад, теореми *ідентичності* вказують на на можливість вилучення елементів: якщо на один вхід двовхідного елемента ТА завжди подається 1 чи, якщо на один вхід

двовхідного елемента АБО завжди подається 0, то такі елементи можна вилучити. Аналогічно, теореми про нульовий елемент вказують на можливість вилучення: якщо на один вхід елемента ТА завжди подається 0, то на його виході завжди буде 0 і його можна вилучити і т. д. Доведення цих теорем відносно просте, оскільки булеві змінні набувають лише двох значень, то і теорему можна довести простим перебором усіх можливих комбінацій параметрів.

Теореми про комутативність та асоціативність аналогічні відповідним теоремам звичайної алгебри. Теореми про дистрибутивність у булевій і звичайній алгебрі різняться: у булевій алгебрі операції ТА і АБО дистрибутивні відносно АБО і ТА, відповідно, а у звичайній алгебрі операція додавання не дистрибутивна відносно множення. Теореми поглинання, комбінування і узгодженості не мають аналогів у звичайній алгебрі. Теорема де Моргана стверджує, що інверсія результату множення декількох термів дорівнює сумі інверсій кожного терму. Відповідно до теореми де Моргана елемент ТА-НІ еквівалентний елементу АБО з інвертованими входами, а елемент АБО-НІ еквівалентний елементу ТА з інвертованими входами. Ця теорема широко застосовується для мінімізації булевих функцій. Наприклад:

$$Y = A + B * C + \overline{(D * E * F + G * H)} = \overline{\overline{A} * \overline{BC} * (D * E * F + G * H)} =$$

$$= \overline{\overline{A} * \overline{B * C} * \overline{\overline{D * E * F} * \overline{G * H}}}.$$

Тут теорему де Моргана використано двічі. Не важно помітити, що для створення електронної схеми, яка реалізує цю функцію, знадобляться тільки логічні елементи ТА-НІ з різною кількістю входів.

Таблиця 12

Назва	Теорема	Дуальна теорема
	Теореми однієї змінної	
1. Ідентичність	$B*1=B$	$B+0=B$
2. Нульовий елемент	$B*0=0$	$B+1=1$
3. Ідемпотентність	$B*B=B$	$B+B=B$
4. Інволюція	$\overline{\overline{B}}=B$	
5. Доповняльність	$B*\overline{B}=0$	$B+\overline{B}=1$
	Теореми двох змінних	
6. Комутативність	$B*C=C*B$	$B+C=C+B$
7. Асоціативність	$(B*C)*D=B*(C*D)$	$(B+C)+D=B+(C+D)$
8. Дистрибутивність	$(B*C)+(B*D)=B*(C+D)$	$(B+C)*(B+D)=B+(C*D)$
9. Поглинання	$B*(B+C)=B$	$B+(B*C)=B$
10. Комбінування	$(B*C)+(B*\overline{C})=B$	$(B+C)*(B+\overline{C})=B$
11. Узгодженість	$(B*C)+(\overline{B}*D)+(C*D)=$ $=(B*C)+(\overline{B}*D)$	$(B+C)*(\overline{B}+D)*(C+D)=$ $=(B+C)*(\overline{B}+D)$
12. Теорема де Моргана	$\overline{B_0*B_1*B_2*\ldots}=\overline{B}_0+\overline{B}_1+\overline{B}_2+\ldots$	$\overline{B_0+B_1+B_2+\ldots}=\overline{B}_0*\overline{B}_1*\overline{B}_2*\ldots$

2.1.3. Спрощення булевих виразів. Карти Карно

Теореми булевої алгебри дозволяють спрощувати булеві вирази. Наприклад, вираз $Y = \overline{A} * \overline{B} * C + \overline{A} * B * C$, для реалізації якого без спрощення буде потрібно два елементи ТА на три входи, елемент АБО та інвертори, можна суттєво спростити:

$$Y = \overline{A} * \overline{B} * C + \overline{A} * B * C = \overline{A} * C * \left(\overline{B} + B\right) = \overline{A} * C,$$

для реалізації такого спрощеного виразу необхідний лише один елемент ТА на два входи та інвертор.

Основним принципом спрощення диз'юнктивних виразів є комбінування термів з використанням рівності $D * E + D * \overline{E} = D$, де D, E – довільні імпліканти. Вираз диз'юнктивної форми називають *мінімізованим*, якщо він включає мінімальну можливу кількість імплікант. Якщо є декілька виразів з однаковою кількістю імплікант, мінімальним буде той, у якому менше літералів.

Імпліканта називається *простою* (*prime implicant*), якщо вона не може бути об'єднана з іншими імплікантами в рівнянні для того, щоб утворити нову імпліканту з меншою кількістю літералів. Усі імпліканти в мінімальному рівнянні мають бути простими, інакше вони можуть бути об'єднані, щоб зменшити кількість літералів.

Розглянемо приклад спрощення складного булевого виразу

$$Y = A * \overline{B} * C * D + A * B * C * D +$$
$$+ A * B * \overline{C} * D + \overline{A} * B * C * D + \overline{A} * B * \overline{C} * D,$$

який відповідає таблиці істинності (табл. 13):

$$Y = A * \overline{B} * C * D + A * B * C * D + A * B * \overline{C} * D +$$
$$+ \overline{A} * B * C * D + \overline{A} * B * \overline{C} * D =$$
$$= \left(A * \overline{B} * C * D + A * B * C * D\right) + \left(A * B * C * D + A * B * \overline{C} * D\right) +$$
$$+ \left(\overline{A} * B * C * D + \overline{A} * B * \overline{C} * D\right) =$$
$$= A * C * D + A * B * D + \overline{A} * B * D = A * C * D + B * D.$$

Таблиця 13

	A	*B*	*C*	*D*	Мінтерм	*Y*
0	0	0	0	0	$\bar{A}*\bar{B}*\bar{C}*\bar{D}$	0
1	0	0	0	1	$\bar{A}*\bar{B}*\bar{C}*D$	0
2	0	0	1	0	$\bar{A}*\bar{B}*C*\bar{D}$	0
3	0	0	1	1	$\bar{A}*\bar{B}*C*D$	0
4	0	1	0	0	$\bar{A}*B*\bar{C}*\bar{D}$	0
5	0	1	0	1	$\bar{A}*B*\bar{C}*D$	1
6	0	1	1	0	$\bar{A}*B*C*\bar{D}$	0
7	0	1	1	1	$\bar{A}*B*C*D$	1
8	1	0	0	0	$A*\bar{B}*\bar{C}*\bar{D}$	0
9	1	0	0	1	$A*\bar{B}*\bar{C}*D$	0
10	1	0	1	0	$A*\bar{B}*C*\bar{D}$	0
11	1	0	1	1	$A*\bar{B}*C*D$	1
12	1	1	0	0	$A*B*\bar{C}*\bar{D}$	0
13	1	1	0	1	$A*B*\bar{C}*D$	1
14	1	1	1	0	$A*B*C*\bar{D}$	0
15	1	1	1	1	$A*B*C*D$	1

Для спрощення використовують *теореми ідемпотентності* та *дистрибутивності*. Більш систематичний і наочний підхід до спрощення булевих виразів запропонував у 1953 р. М. Карно, і тому цей метод називають *методом карт Карно*. Особливо успішне застосування він має, якщо кількість аргументів функції не перевищує чотири.

Із розглянутих вище прикладів видно, що логічна мінімізація здійснюється шляхом об'єднання термів $I*A$ та $I*\bar{A}$, що складаються з імпліканти I, змінної A та її інверсії $\bar{A}$, після цього змінна A зникає з виразу $I*A+I*\bar{A}=I$. Карти Карно дозволяють легко знаходити терми, які можна об'єднати, для цього їх розташовують у таблиці.

На рис. 12 наведено карту Карно для функції чотирьох аргументів змінних, що були розглянуті. Верхній рядок дає чотири

можливі значення для аргументів A і B. Ліва колонка дає чотири можливі значення аргументів C та D. Кожна клітинка карти Карно відповідає рядку таблиці істинності і містить значення функції Y із цього рядка. Номер відповідного рядка таблиці істинності наведено у верхньому лівому кутку кожної клітинки карти Карто малим шрифтом. Наприклад, верхня ліва клітинка відповідає нульовому рядку таблиці істинності й показує, що значення функції Y дорівнюватиме 0, коли $ABCD = 0000$. Як і кожному рядку в таблиці істинності, так і кожній клітинці карти Карно відповідає окремий мінтерм.

		AB			
		00	01	11	10
CD	00	^{0} 0	^{4} 0	^{12} 0	^{8} 0
	01	^{1} 0	^{5} 1	^{13} 1	^{9} 0
	11	^{3} 0	^{7} 1	^{15} 1	^{11} 1
	10	^{2} 0	^{6} 0	^{14} 0	^{10} 0

Рис. 12. Карта Карно функції чотирьох аргументів

Зауважимо, що кожна клітинка (і відповідний мінтерм) відрізняється від сусідньої зміною лише одного аргументу функції. Тобто сусідні клітинки різняться лише значенням одного літерала, яке дорівнює 1 в одній клітинці та 0 у сусідній. Наприклад, клітинки, які представляють 4-й і 12-й рядки таблиці істинності сусідні, оскільки їх мінтерми $\overline{A} * B * \overline{C} * \overline{D}$ та $A * B * \overline{C} * \overline{D}$ різняться тільки в аргументі A. Це досягається переліком аргументів A і B у верхньому рядку (а аргументів C і D у лівому стовпчику) в особливому порядку: 00, 01, 11, 10. Цей порядок називають *кодом Грея* (*Gray code*).

На відміну від звичайного бітового порядку за збільшенням величини (00, 01, 10, 11), у коді Грея сусідні елементи відрізняються тільки одним бітом. Наприклад, сусідні елементи 01 та 11 коду Грея відрізняються тільки зміною першого біту з 0 на 1, тоді як сусідні елементи 01 10 звичайного порядку за збільшенням величини відрізняються двома бітами. Окрім цього, карти Карно "закільцьовані". Клітинка з крайнього правого боку таблиці сусідня з крайньою лівою того самого рядка, оскільки вони

відрізняються лише в одному аргументі (A). Таким чином, можна умовно згорнути карту в циліндр, з'єднавши краї.

Карти Карно забезпечують простий візуальний спосіб мінімізації булевих виразів. Для цього треба зобразити на карті Карно прямокутні блоки, до яких входять клітинки з $Y = 1$. Кожний блок має бути максимально великим, а їхня кількість має бути мінімальною. Кожний такий блок на карті Карно є імплікантою. Максимально можливий блок є первинною імплікантою. Для мінімізації функції достатньо записати суму імплікант, які відповідають всім блокам.

Правила відшукання мінімального виразу з карт Карно такі:

- треба використовувати найменшу кількість блоків, необхідних для покриття всіх 1;
- 1 на карті Карно може входити в декілька блоків;
- усі клітинки в кожному блоці обов'язково мають містити тільки 1;
- кількість клітинок кожного блоку в кожному напрямку має бути степенем двійки (тобто 1, 2 або 4);
- кожний блок має бути максимально великим;
- блок може зв'язувати краї карти Карно.

На рис. 12 зображено карту Карно, у якій два блоки покривають всі 1, клітинка 15-го рядка входить в обидва блоки.

Для кожного блоку треба записати імпліканту, що йому відповідає. Аргументи, для яких пряма й інверсна форми потрапляють до одного блоку, виключаються з імпліканти. Таким чином, великому блоку відповідатиме імпліканта $B * D$, а маленькому – $A * C * D$. У результаті можемо отримати вираз $Y = A * C * D + B * D$, який був отриманий раніше з використанням булевої алгебри.

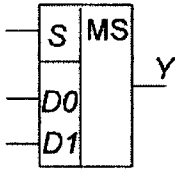
У таблицях істинності і картах Карно можна використовувати байдуже значення "–", якщо стан виходу не важливий або відповідна комбінація входів ніколи не виникає. Такі виходи можуть набувати значення 0 або 1 залежно від того, як вирішить розробник (чи програма оптимізації). У картах Карно байдужі значення дозволяють провести ще більшу логічну мінімізацію. Їх можна включати до блоків, якщо це допомагає покрити оди-

ниці або меншою кількістю блоків, або великими блоками, причому байдужі значення можна і не покривати, якщо це не допомагає мінімізації.

2.2. Базові комбінаційні блоки

2.2.1. Мультиплексори

У цифровій схемотехніці зазвичай використовуються мультиплексори. Залежно від значення сигналу вибору мультиплексор обирає один із декількох входів і подає його значення на свій вихід. Найпростіший мультиплексор має два входи (2:1), умовне позначення і таблицю істинності якого показано на рис. 13. Залежно від сигналу вибору (або керуючого сигналу) S мультиплексор передає на вихід один із двох вхідних сигналів: якщо $S = 0$, то $Y = D0$, якщо $S = 1$, то $Y = D1$.



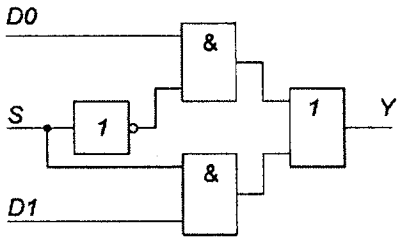
а)

S	$D0$	$D1$	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

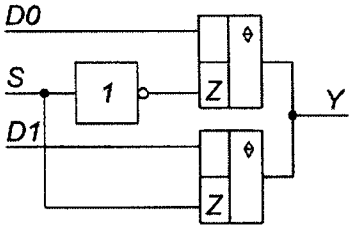
б)

Рис. 13. Мультиплексор з двома входами: позначення (а) і таблиця істинності (б)

Двовхідний мультиплексор може бути побудований з використанням диз'юнкції кон'юнкцій, як показано на рис. 14, *а*, або буферів з трьома станами – рис. 14, *б*: за допомогою інвертора сигнал вибору завжди активує тільки один буфер.



а)



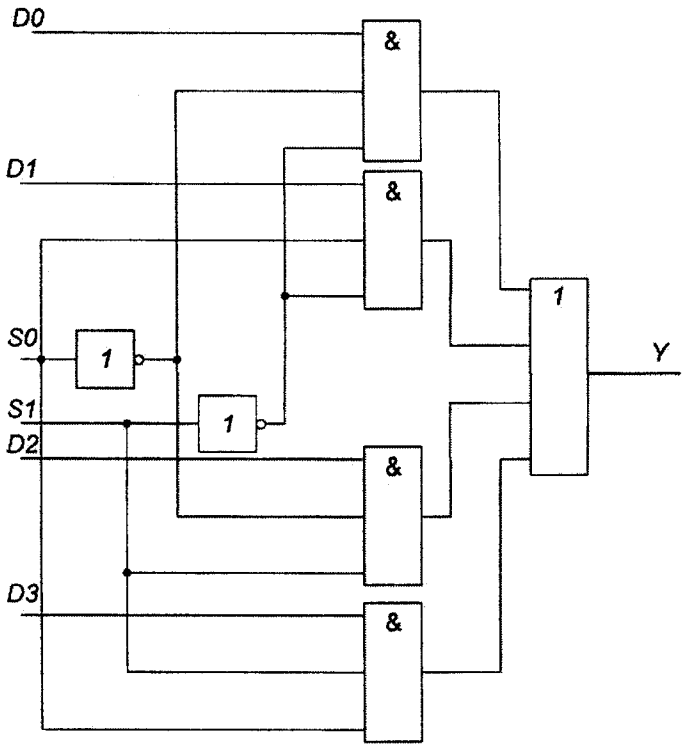
б)

Рис. 14. Реалізації двовхідного мультиплексора

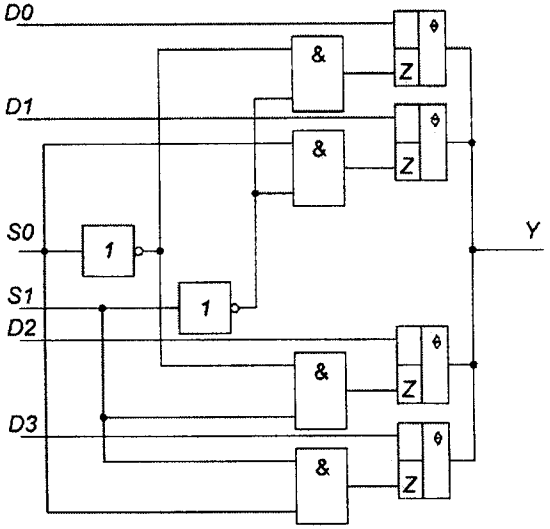
На практиці використовують мультиплексори з більшою кількістю входів: якщо кількість інформаційних входів дорівнює N, то кількість керуючих сигналів буде $\log_2 N$. Чотиривхідний мультиплексор (4:1) має чотири входи даних, два сигнали вибору і один вихід. Чотиривхідний мультиплексор може бути побудований з використанням диз'юнкції кон'юнкцій (рис. 15, *а*), буферів з трьома станами рис. 15, *б* або двовхідних мультиплексорів рис. 15, *в*. Аналогічні схеми застосовують для побудови мультиплексорів з більшою кількістю входів.

Мультиплексори можна використовувати як *таблиці перетворення* (*lookup tables*) для обчислення довільної логічної функції. На рис. 16 наведено приклад такого використання мультиплексора для обчислення логічної функції, заданої таблицею істинності. Аргументи A і B подаються на входи керування S мультиплексора. На входи даних D мультиплексора подаються кон-

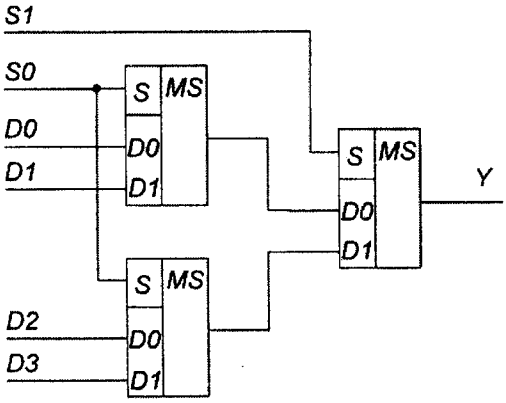
станти 0 чи 1 згідно з відповідним рядком таблиці істинності. Аналогічним чином 2^N-вхідний мультиплексор можна запрограмувати для реалізації довільної логічної функції N аргументів, а зміною вхідних даних мультиплексор може бути перепрограмований для обчислення іншої. При цьому фактично буде використовуватися схема, подібна наведеним на рис. 15, тобто досить громіздка, оскільки ніяка мінімізація не виконується.



а)

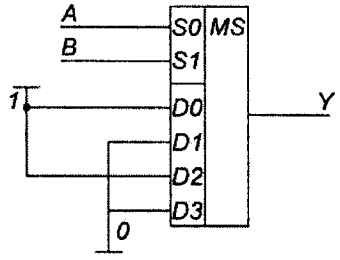


б)



в)

Рис. 15. Реалізація чотиривхідного мультиплексора: дворівнева логіка (а), буфери з трьома станами (б), ієрархічна (в)



а)

A	B	Y
0	0	1
0	1	0
1	0	1
1	1	0

б)

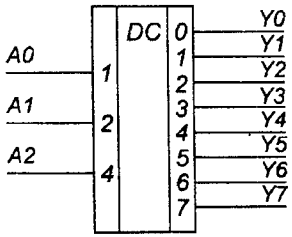
Рис. 16 Реалізація довільної логічної функції на мультиплексорах

2.2.2. Дешифратори

Дешифратор має N входів і 2^N виходів. Логічна 1 подається на один і тільки на один вихід залежно від набору вхідних значень. На рис. 17 зображено дешифратор 3:8 (три на вісім) і його таблиця істинності. Виходи такого дешифратора утворюють *прямий унітарний код* (*one-hot code*). Код має таку назву тому, що в будь-який момент часу тільки один із виходів може набувати значення 1.

На рис. 18 наведено реалізацію дешифратора 3:8, яка використовує 8 елементів АБО та три інвертори. На кожний елемент АБО подається вхідний сигнал у прямій або інверсній формі. Очевидно, що дешифратор N:2^N може бути побудований із 2^N N-вхідних елементів АБО. Кожний вихід у дешифраторі є мінтермом, наприклад, $Y4 = A2 * \overline{A1} * \overline{A0}$. Дешифратор обчислює всі мінтерми. Завдяки цьому дешифратор можна використовувати разом з елементами АБО для реалізації довільної логічної функції. На рис. 19 наведено реалізацію функції виключне АБО ($Y = \overline{A2} * \overline{A1} * A0 + \overline{A2} * A1 * \overline{A0} + A2 * \overline{A1} * \overline{A0}$) із трьома аргументами, що використовує дешифратор 3:8 та один елемент АБО (див. табл. 9). Оскільки кожний вихід дешифратора представляє

один мінтерм, то функція побудована як логічне АБО всіх необхідних мінтермів: $Y = Y1 + Y2 + Y4$, інші виходи дешифратора не використовуються. Для обчислення функції N аргументів, що має M логічних 1 у таблиці істинності, треба мати дешифратор $N{:}2^N$ та M-вхідний елемент АБО. Цей елемент має бути підключеним до всіх мінтермів, що містять логічну 1 у таблиці істинності.



а)

A2	A1	A0	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

б)

Рис. 17. Дешифратор 3:8: позначення (а) і таблиця істинності (б)

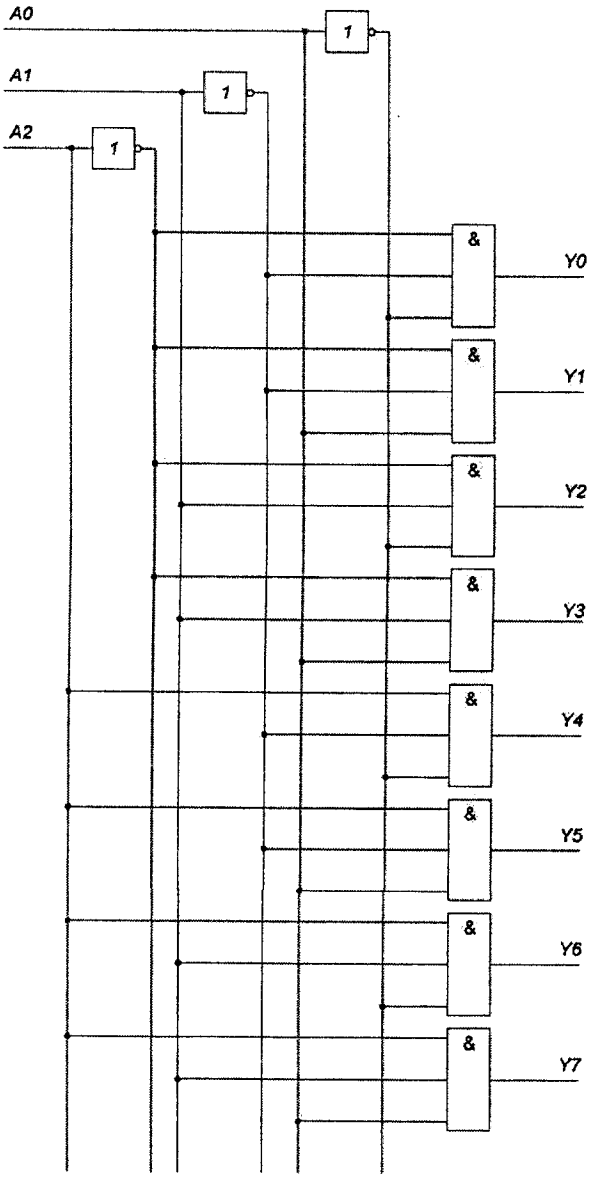


Рис. 18. Реалізація дешифратора 3:

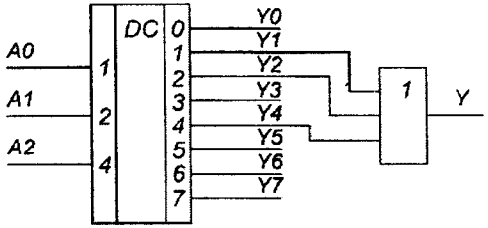
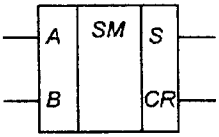


Рис. 19. Обчислення функції виключне АБО з використанням дешифратора 3:8

2.2.3. Суматор та компаратор

Арифметичні схеми є основним функціональним вузлом будь-якого комп'ютера. Найпоширенішою арифметичною схемою є суматор. Існує багато схем суматорів, але всі вони використовують однорозрядні схеми додавання: півсуматор (*half adder*) і повний суматор (*full adder*). Півсуматор має два входи: A і B та два виходи S і CR, де S – це сума A і B. Якщо і A і B одночасно дорівнюють 1, то вихід S має дорівнювати 2, але таке число не може бути представлене з використанням одного двійкового розряду. У цьому випадку, окрім того, що $S = 1$, також виставляється біт перенесенням CR у наступний розряд. Півсуматор може бути побудований з елементів виключне АБО і ТА (рис. 20). Повний суматор, на відміну від півсуматора, має ще один вхід – вхід перенесення *cr*, на який надходить перенесення з молодшого розряду.



а)

$$S = A \oplus B$$

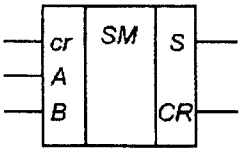
$$CR = A * B$$

A	B	S	CR
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

б)

Рис. 20. Півсуматор: позначення (а), логічні вирази та таблиця істинності (б)

N-розрядний суматор додає два *N*-розрядні числа (*A* і *B*), а також вхідне перенесення *cr*, і формує *N*-розрядний результат *S* і вихідне перенесення *CR*. Такий суматор називається суматором з перенесенням, що поширюється (*carry propagate adder*, CPA), оскільки вихідне перенесення одного розряду переходить у наступний розряд. Умовне позначення такого суматора показано на рис. 21.



а)

$$S = A \oplus B \oplus cr$$

$$CR = A * B + A * cr + B * cr$$

cr	*A*	*B*	*S*	*CR*
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

б)

Рис. 21. Повний суматор: позначення (а), логічні вирази та таблиця істинності (б)

Найпоширенішими реалізаціями суматора з перенесенням, що поширюється є: суматори з послідовним перенесенням (*ripple-carry adders*), із прискореним перенесенням (*carry-lookahead adders*) і префіксні суматори (*prefix adders*), рис. 22.

Найпростіший спосіб реалізації *N*-розрядного суматора – це об'єднання в ланцюг *N*-повних суматорів. Вихід *CR* кожного розряду надходитиме на вхід *cr* наступного розряду і т. д. (рис. 23). Така схема називається *суматором з послідовним перенесенням* (*ripple-carry adder*). Модуль повного суматора багато разів використовується для побудови більшої системи. Такий суматор має дуже серйозний недолік: швидкість його роботи спадає при збільшенні кількості розрядів *N*.

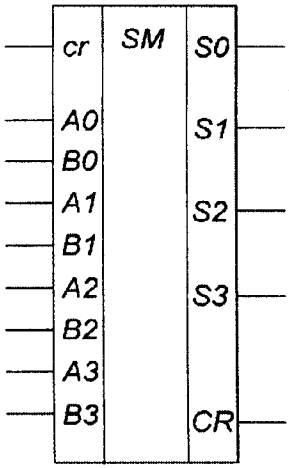


Рис. 22. Чотирирозрядний суматор
із перенесенням, що поширюється

Дійсно, $S31$ залежить від $C30$, який залежить від $C29$, який, у свою чергу, залежить від $C28$ і т. д. до *cr_in*, $A0$ та $B0$ (рис. 23). Перенесення проходить через увесь ланцюг. Затримка такого суматора лінійно збільшується разом із кількістю розрядів.

Головною причиною повільної роботи багаторозрядних суматорів з послідовним перенесенням є необхідність проходження сигналу перенесення через усі біти суматора. Суматори з *прискореним перенесенням* (*carry-lookahead adder*, CLA) представляють інший тип суматорів із перенесенням, що поширюється, яке вирішує цю проблему шляхом розділення суматора на блоки та реалізації цих блоків так, щоб визначити вихідне перенесення блоку щойно стало відомим його вхідне перенесення.

Таким чином, не треба чекати проходження перенесення через усі повні суматори всередині блоку. Наприклад, 32-розрядний суматор може бути розділений на 8 чотирирозрядних суматорів.

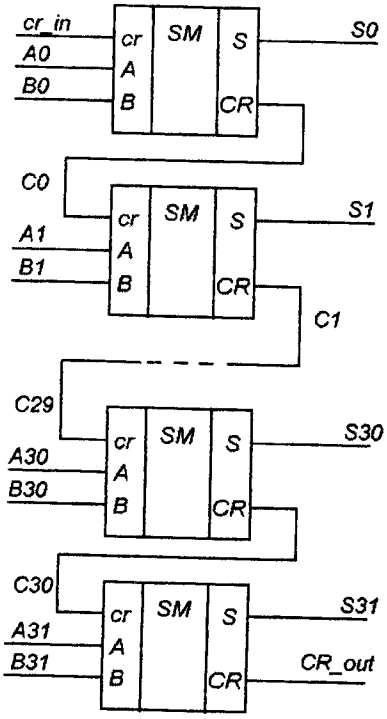


Рис. 23. Багаторозрядний суматор з послідовним перенесенням

Суматори з прискореним перенесенням використовують сигнали генерації (G) і поширення (P), які визначають вихід перенесення цілого блоку чи окремого розряду. Кажуть, що i-й розряд суматора генерує перенесення, якщо він видає перенесення на своєму виході незалежно від наявності перенесення на вході. Це відбувається, коли обидва входи i-го розряду суматора A_i та B_i одночасно дорівнюють 1. Таким чином, сигнал генерації G_i можна визначити як $G_i = A_i * B_i$. Розряд називається таким, що поширює перенесення, якщо вихідний сигнал перенесення з'являється за наявності вхідного перенесення. Розряд поширюватиме вхідний сигнал перенесення (C_{i-1}), якщо або A_i,

або B_i дорівнюють 1, тобто $P_i = A_i + B_i$. Використовуючи ці визначення, можна переписати логіку формування сигналу перенесення для певного розряду. Розряд i суматора формуватиме вихідний сигнал перенесення C_i, якщо він або генерує перенесення G_i, або поширює вхідне перенесення $P_i * C_{i-1}$. Отже, перенесення з i-го розряду суматора можна записати так:

$$C_i = G_i + P_i * C_{i-1} = A_i * B_i + (A_i + B_i) * C_{i-1}$$

Поняття сигналів генерації та їх поширення використовується і для багаторозрядних блоків. *Генеруючим перенесення блоком* називається такий блок, який створює вихідне перенесення незалежно від вхідного сигналу перенесення цього блоку. Блок називається таким, що поширює перенесення, якщо його вихідне перенесення виникає при надходженні вхідного перенесення. Сигнали $G_{i:j}$ та $P_{i:j}$ визначаються як сигнали генерації та поширення для блоку, що охоплює розряди від i до j.

Блок генерує перенесення, якщо найстарший розряд генерує перенесення або якщо старший розряд поширює перенесення, згенероване попереднім розрядом і т. д. Наприклад, логіка блоку генерації для блоку, що охоплює розряди від 0 до 3, буде такою:

$$G_{3:0} = G_3 + P_3 * (G_2 + P_2 * (G_1 + P_1 * G_0)).$$

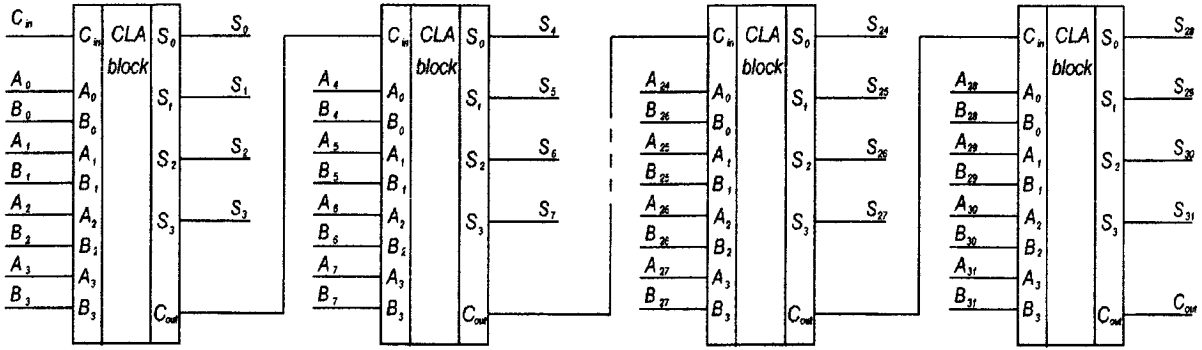
Блок поширює перенесення, якщо всі розряди, що входять до його складу, це перенесення поширюють. Логіка поширення для блоку, що охоплює розряди від 0 до 3 дуже проста:

$$P_{3:0} = P_3 * P_2 * P_1 * P_0.$$

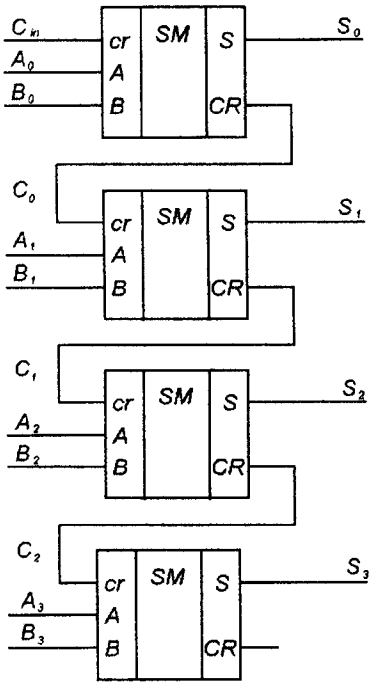
За допомогою блокових сигналів генерації та поширення можна швидко визначити вихідне перенесення блоку C_i, використовуючи його вхідне перенесення C_j:

$$C_i = G_{i:j} + P_{i:j} * C_j.$$

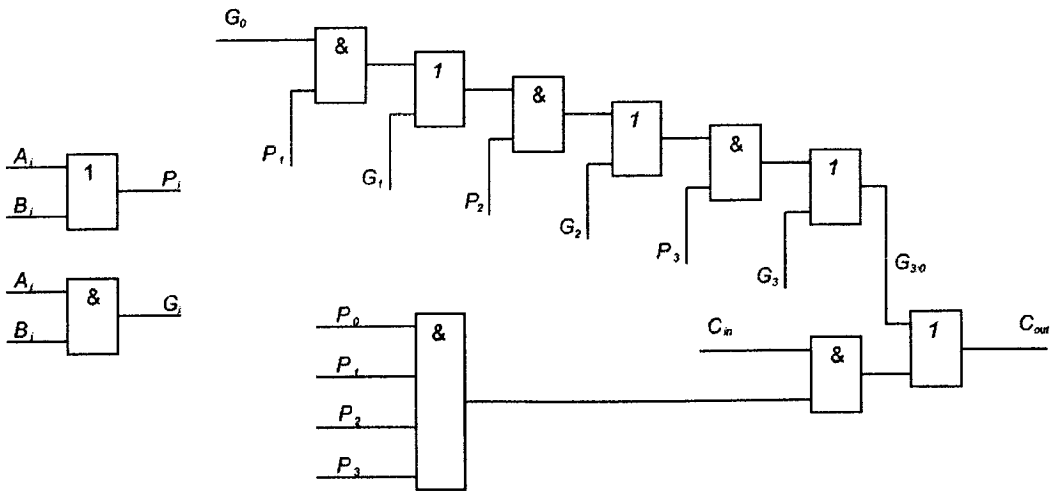
На рис. 24, *а* зображено 32-розрядний суматор з прискореним перенесенням, що складається із восьми чотирирозрядних блоків. Кожний блок містить чотирирозрядний суматор з послідовним перенесенням (рис. 24, *б*), подібний розглянутому вище, і схему прискореного перенесення, яка визначає вихідне перенесення блоку за вхідним, цю схему показано на рис. 24, *в*.


C_{in}
CLA block
A_0 B_0 A_1 B_1 A_2 B_2 A_3 B_3
S_0 S_1 S_2 S_3 C_{out}
A_4 B_4 A_5 B_5 A_6 B_6 A_7 B_7
S_4 S_5 S_6 S_7
A_{24} B_{24} A_{25} B_{25} A_{26} B_{26} A_{27} B_{27}
S_{24} S_{25} S_{26} S_{27}
A_{28} B_{28} A_{29} B_{29} A_{30} B_{30} A_{31} B_{31}
S_{28} S_{29} S_{30} S_{31} C_{out}

a)



б)



в)

Рис. 24. Суматор з прискореним перенесенням

Усі блоки суматора одночасно обчислюють однобітові і блокові сигнали генерації та поширення. Критичний шлях розпочинається з обчислення G_0 і $G_{3:0}$ у першому блоці суматора. Сигнал C_{in} потім поширюється до C_{out} через логічні елементи ТА/АБО всіх блоків. Для великого суматора це відбувається набагато швидше, ніж поширення перенесення через кожен наступний розряд суматора.

Як було показано, суматори можуть додавати додатні та від'ємні числа, якщо вони представлені в доповняльному коді. Віднімання проводиться також дуже просто: знак другого числа треба змінити на протилежний, потім числа додаються. Зміна знака числа в доповняльному коді проводиться шляхом інверсії бітів і додавання 1.

Для обчислення $Y = A - B$ спочатку створюється доповняльний код числа B: інвертуються всі розряди B і додається 1; $-B = \overline{B} + 1$. Отримане значення додається до A. Цю суму можна отримати одним суматором, який має вхід перенесення: він має обчислити $A + \overline{B}$ за $C_{in} = 1$. На рис. 25 наведено таку схему для віднімання двох чотирирозрядних чисел.

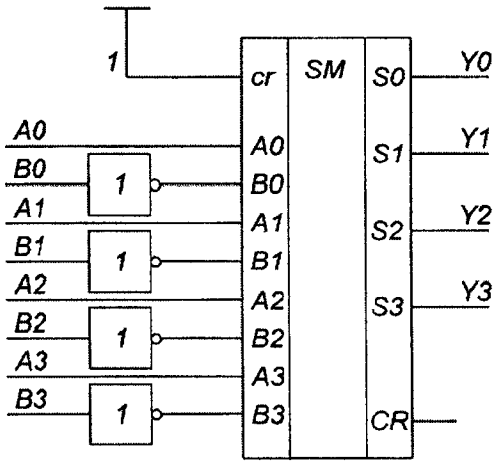


Рис. 25. Чотирирозрядна схема віднімання

У цифровій електроніці також застосовуються компаратори, які визначають, чи рівні (однакові за величиною) між собою два двійкових числа, а якщо ні, то визначають, яке з них більше/менше за інше. Компаратор отримує два N розрядних двійкових числа A та B. Існують два типи компараторів – рівності та величини. *Компаратор рівності* має один вихідний сигнал, який показує, чи рівні між собою A та B ($A = B$), але він не визначає, яке із чисел більше. *Компаратор величини* має декілька вихідних сигналів і визначає відношення величин A та B.

Компаратор рівності має дуже просту апаратну реалізацію. На рис. 26 показано реалізацію чотирирозрядного компаратора рівності. На вході за допомого елементів ВИКЛЮЧНЕ АБО-НІ перевіряється рівність усіх розрядів вхідних чисел. Числа будуть рівними, якщо всі відповідні розряди рівні.

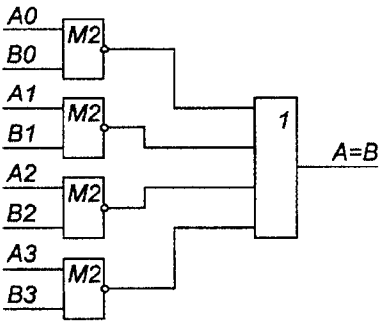


Рис. 26. Чотирирозрядний компаратор рівності

Компаратор величини (рис. 27) обчислює A-B і аналізує всі розряди та знак рівності.

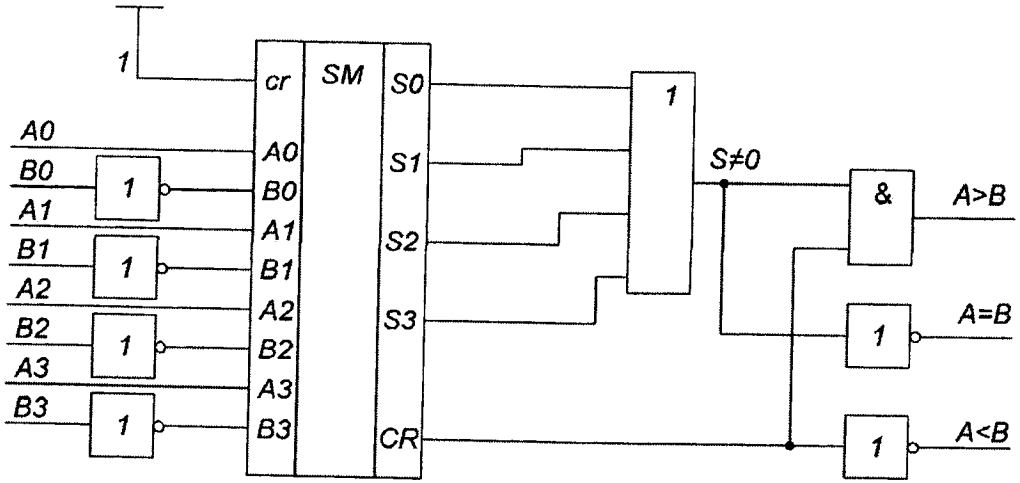


Рис. 27. Чотирирозрядний компаратор величини

2.2.4. Схеми множення

Множення беззнакових двійкових чисел цілком подібне до десяткового множення В обох випадках часткові добутки формуються шляхом множення окремих розрядів множника на все множене, але множення окремих розрядів множника та множеного у випадку двійкових чисел дуже просте, для його обчислення використовується операція ТА. Зсунуті часткові добутки потім додаються описаними вище суматорами (рис. 28).

У загальному випадку схема множення NxN перемножує два N-розрядних числа і видає $2N$-розрядний результат. Часткові добутки при двійковому множенні дорівнюють або множеному, або нулю. Множення одного розряду двійкових чисел рівносильне операції ТА, і тому для формування часткових добутків використовуються логічні елементи ТА.

На рис. 29 показано апаратну реалізацію помножувача 4x4. Помножувач отримує множене A, множник B і обчислює добуток P. Кожний частковий добуток дорівнює результату операцій ТА, аргументами яких є окремі розряди множника (B_3, B_2, B_1 або B_0) і всі розряди множеного (A_3, A_2, A_1, A_0). Для N-розрядних операндів існуватиме N часткових добутків і N–1 рівнів одноразрядних суматорів. Наприклад, для помножувача 4x4 частковий добуток першого ряду – набір бітів A_3B_0, A_2B_0, A_1B_0, A_0B_0. Цей частковий добуток додається до зсунутого на один розряд вліво другого часткового добутку A_3B_1, A_2B_1, A_1B_1, A_0B_1. Для обробки кожного наступного розряду до схеми додається ряд елементів ТА і суматорів. Очевидно, що побудова схеми рис. 29 повністю відповідає класичному правилу множення рис. 28

					A_3	A_2	A_1	A_0
				$\times$	B_3	B_2	B_1	B_0
					A_3B_0	A_2B_0	A_1B_0	A_0B_0
				A_3B_1	A_2B_1	A_1B_1	A_0B_1	
			A_3B_2	A_2B_2	A_1B_2	A_0B_2		
+		A_3B_3	A_2B_3	A_1B_3	A_0B_3			
	P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0

Рис. 28 Множення двійкових чисел

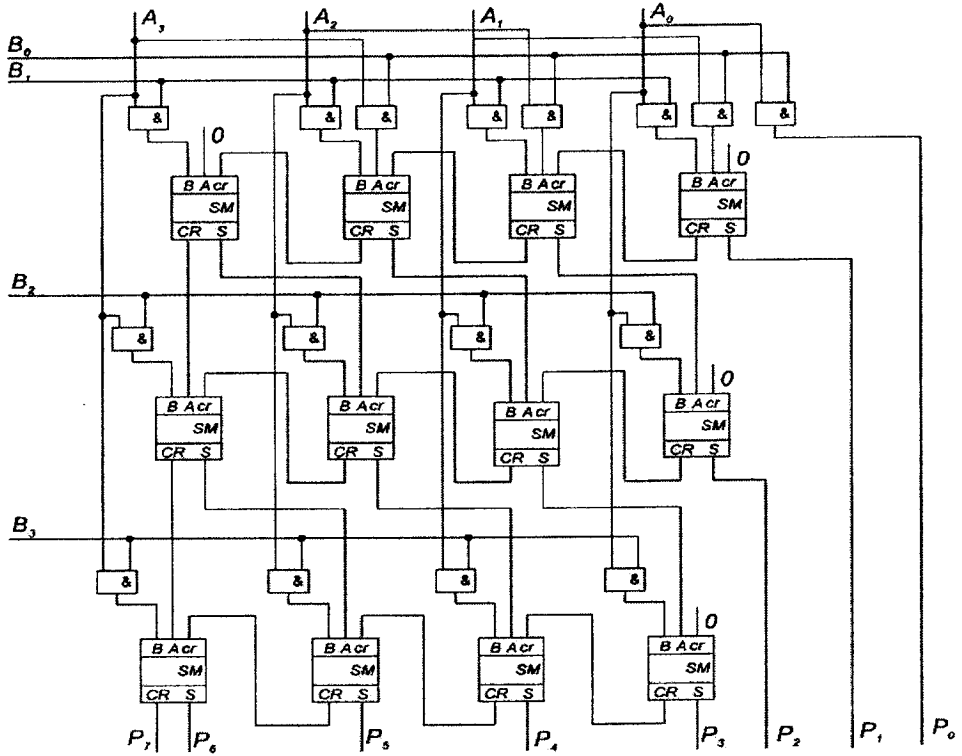


Рис. 29. Схема множення 4×4

2.2.5. Схеми зсуву

Схеми зсуву і схеми циклічного зсуву пересувають розряди двійкового числа вліво чи вправо на певну кількість позицій, а отже, вони множать або ділять число на деякий степінь 2. Існують декілька типів таких схем:

- схеми логічного зсуву пересувають розряди числа вліво (LSL, <<) чи вправо (LSR, >>) і заповнюють порожні розряди нулями. Наприклад, 11001 >> 2 = 00110; 11001 << 2 = 00100;
- схеми арифметичного зсуву вправо (ASR, >>>) пересувають розряди числа вправо і заповнюють найбільш значущі розряди значенням знакового біта початкового числа. Арифметичний зсув вліво (ASL) збігається з логічним зсувом (LSL). Наприклад: 11001 >>> 2 = 11110;
- схеми циклічного зсуву пересувають так розряди по кругу, що порожні місця заповнюються розрядами, які висунені з іншого кінця. Наприклад: 11001 ROR 2 = 01110; 11001 ROL 2 = 00111.

N-розрядна схема зсуву відносно проста, але громіздка (рис. 30). Її можна побудувати із N мультиплексорів N:1. Розряди вхідного сигналу зсуваються на 0 – (N–1) розрядів залежно від значення $\log_2 N$ ліній вибору ($S0$, $S1$). На рис. 30 показано апаратну реалізацію чотирирозрядної схеми зсуву, яка має дві лінії вибору ($S0$, $S1$), що визначають, на скільки розрядів (0–3) відбувається зсув. Якщо $S0 = S1 = 0$, то зсув не відбувається, $Y = A$.

Зсув вліво на N розрядів множить число на 2^N. Наприклад, зсув вліво $000101_2 = 5_{10}$ на 3 розряди означає множення на $2^3 = 8$, $000101_2 << 3 = 101000_2 = 40_{10}$. Арифметичний зсув вправо на N розрядів ділить число на 2^N. Наприклад, зсув вправо $11100_2 = -4_{10}$ на 2 розряди означає ділення на $2^2 = 4$, $-4_{10} = 11100_2 >>> 2 = 11111_2 = -1_{10}$.

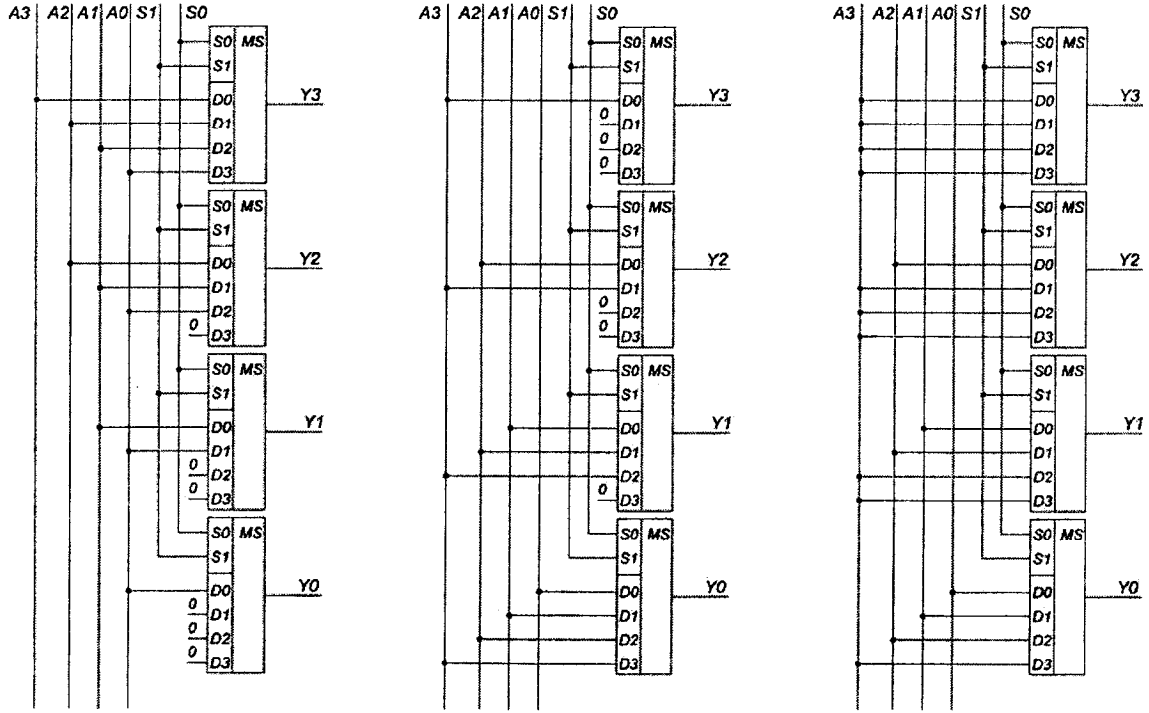


Рис. 30. Схема зсуву: а) ASL та LSL; б) LSR; в) ASR

2.3. Часові характеристики комбінаційних логічних схем

2.3.1. Швидкодія логічних елементів

Усі цифрові схеми складаються з транзисторів, опорів, ємностей (можливо паразитних), а отже, вони мають скінченну швидкодію. Таким чином, зміна вихідного сигналу схеми відбувається з деякою затримкою після зміни вхідного сигналу. На рис. 31 показано часову діаграму переключення інвертора, де видно затримку між зміною входу та наступною зміною його виходу. Перехід від НИЗЬКОГО рівня до ВИСОКОГО називається *позитивним перепадом*, або *переднім фронтом*. Аналогічно, перехід від ВИСОКОГО рівня до НИЗЬКОГО (на рис. 31 він не показаний) називається *негативним перепадом*, або *зрізом чи заднім фронтом*. Величина затримки вимірюється від моменту часу, коли вхідний сигнал *A* досягає рівня 50 %, до моменту досягнення рівня 50 % вихідним сигналом *Y*. Рівень 50 % – це точка, у якій напруга на вході або виході елемента міститься точно посередині між напругами 0 та 1.

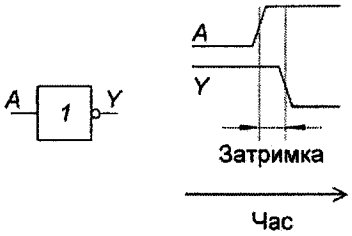


Рис. 31. Затримка цифрової схеми

Логічні схеми характеризуєтся затримкою поширення (*propagation delay*, t_{pd}) і затримкою реакції, або відгуку (*contamination delay*, t_{cd}). Затримка поширення t_{pd} – це максимальний час від початку зміни входу до моменту, коли всі виходи схеми досягнуть стаціонарних значень. Затримка реакції t_{cd} – це мінімальний час від моменту, коли вхід змінився, до моменту, коли будь-який із виходів почне змінювати своє значення.

На рис. 32 зображено затримки поширення і реакції інвертора, де видно, що вхід *A* спочатку мав деяке значення, яке змінюється на протилежне в певний момент часу. Унаслідок роботи внутрішньої схеми інвертора через деякий час змінюється його вихідний сигнал *Y*. Сигнал *Y* може почати змінюватися через часовий інтервал t_{cd} після зміни *A* (але не раніше), *Y* точно набуде нового значення не пізніше, ніж через інтервал t_{pd} після зміни *A*.

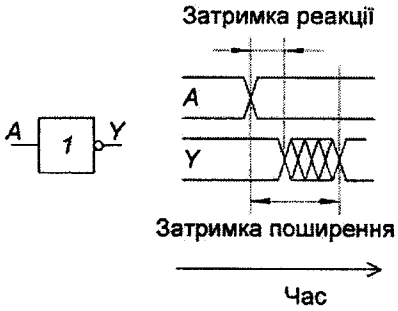


Рис. 32. Затримки реакції та поширення

Коли розробники кажуть про затримку схеми, вони у більшості випадків мають на увазі найбільше можливе значення затримки поширення. Конкретні числові значення затримок поширення і реакції логічного елемента визначаються внутрішньою структурою схеми і технологією її реалізації. Зазвичай розробники отримують ці дані з довідників.

2.3.2. Затримки в комбінаційній схемі

Як і окремий логічний елемент, так і складна комбінаційна логічна схема характеризуються затримками поширення і реакції. Конкретні значення цих величин визначаються не тільки часовими характеристиками окремих елементів, але й шляхом, який проходить сигнал від входу схеми до її виходу. На рис. 33 наведено комбінаційну схему, що має чотири входи. Критичний шлях (*critical path*), виділений жирними лініями, – це шлях від входу *A* до виходу *Y*. Він відповідає ділянці з найбільшою затримкою й є найповільнішим, оскільки вхідному сигналу треба

пройти через чотири елементи до виходу (рис. 34, *а*). Саме він обмежує швидкість, з якою працює схема. Найкоротшим шляхом у схемі є шлях від входу *D* до виходу Y, на рисунку він зображений пунктиром. Це найшвидший шлях у схемі, оскільки вхідному сигналу до виходу треба пройти лише через один елемент (рис. 34, *б*)).

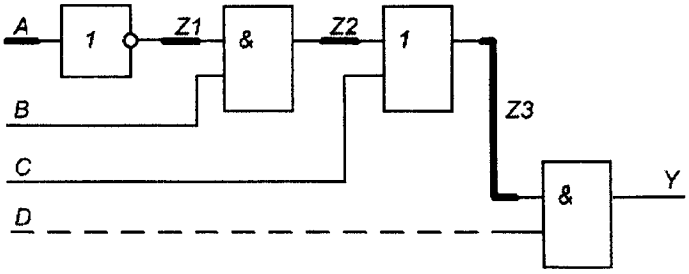


Рис. 33. Критичний шлях та найкоротший шлях

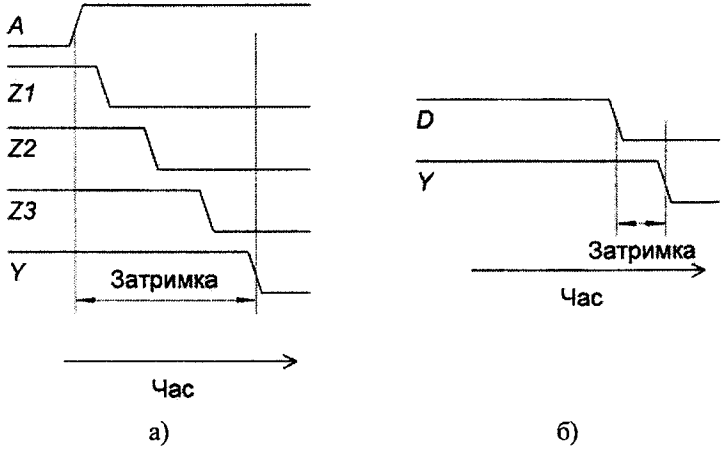


Рис. 34. Затримки в критичному (а) і найкоротшому шляхах (б)

Затримка поширення комбінаційної схеми є сумою затримок поширення всіх елементів критичного шляху, а затримка реакції

є сумою затримок реакції всіх елементів найкоротшого шляху. Для схеми, наведеній на рис. 33, ці затримки дорівнюють

$$t_{pd} = t_{pd\_INV} + 2t_{pd\_AND} + t_{pd\_OR},$$

$$t_{cd} = t_{cd\_AND}.$$

При проектуванні комбінаційних схем зазвичай віддають перевагу схемам з найменшими затримками, але враховують ще й енергоспоживання, вартість і наявність компонентів.

2.3.3. Імпульсні перешкоди

При проектуванні комбінаційних логічних схем може скластися така ситуація, коли одна зміна вхідного сигналу схеми викликає декілька послідовних змін вихідного сигналу до встановлення його стаціонарного значення, тобто на виході схеми виникають паразитні імпульси чи імпульсні перешкоди. Паразитні імпульси можуть виникнути, якщо вхідний сигнал поширюється по двох ділянках, які мають різні затримки та обидві визначають вихідний сигнал.

Як приклад схеми, де виникають паразитні імпульси, можна розглянути мультиплексор (див. рис. 13), схема якого повторена на рис. 35, *а*. Якщо $D0 = D1 = 1$, а керуючий сигнал S переходить із 1 у 0, то вихідний сигнал Y дорівнює 1 і не має змінюватися. Проте при перемиканні S виникає паразитний імпульс, оскільки S впливає на вихід Y двома шляхами: коротким – через елементи ТА і АБО; довгим – через інвертор та елементи ТА і АБО. Відповідні часові діаграми перемикання елементів схеми наведено на рис. 35, *б*.

Видно, що причиною виникнення паразитного імпульсу на виході є неодночасна зміна вхідних сигналів елемента АБО: сигнал C, який входить у короткий шлях, встановлюється на 0 до того, як сигнал B, який входить у довгий шлях встановиться на 1. Після завершення всіх перехідних процесів у схемі вихід Y набуває свого коректного стаціонарного стану 1. Якщо затримка інвертора дорівнює 0, а обидва елементи ТА мають однакові затримки, то паразитний імпульс не виникає.

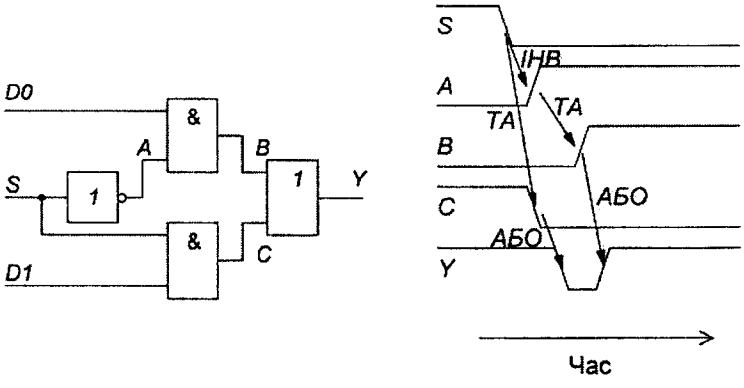


Рис. 35. Схема мультиплексора (а) і його часова діаграма при перемиканні керуючого сигналу (б)

Зазвичай паразитні імпульси не викликають проблем, оскільки вихід комбінаційної схеми врешті набуває коректного значення. При проектуванні послідовнісних схем (див. далі) використовують інформацію з виходів комбінаційних схем після завершення всіх перехідних процесів у них, це основна вимога, яка обмежує швидкодію реальних систем (див. підрозд. 3.4).

3. ПОСЛІДОВНІСНІ СХЕМИ

Зупинимось детально на розгляді послідовнісних схем. Значення вихідних сигналів послідовнісної логічної схеми, на відміну від комбінаційної, залежить як від поточних, так і попередніх значень вхідних сигналів, а отже, послідовнісні логічні схеми мають пам'ять. Послідовнісні логічні схеми можуть явно зберігати попередні значення певних входів, а можуть зберігати в так званих змінних стану меншу кількість інформації, яку називають *станом системи*. Ці змінні містять усю інформацію про минуле, необхідну для визначення майбутньої поведінки схеми.

Найпростішими послідовнісними схемами є тригери і фіксатори (див. далі).

3.1. Тригери і фіксатори

Тригером називається логічна схема з позитивним зворотним зв'язком, що має два стаціонарних стани, які позначаються 0 та 1, і може зберігати інформацію. Приклад такої схеми показано на рис. 36, вона складається з пари інверторів, замкнених у кільце.

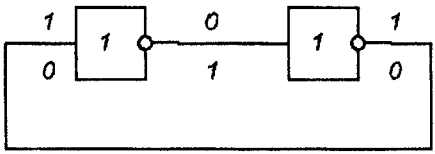


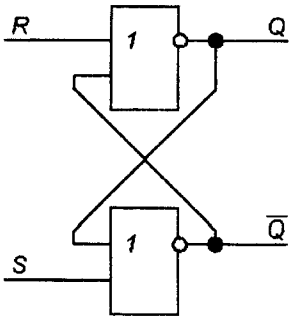
Рис. 36. Бістабільна логічна схема

Очевидно, що схема має зворотний зв'язок, і він позитивний. Схема має два стабільних стани: перший стан – вихід першого інвертора 0, вихід другого (і вхід першого 1); другий стан – вихід першого інвертора 1, вихід другого (і вхід першого 0). Така

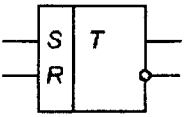
бістабільна схема зберігає 1 біт інформації. На практиці таку схему неможливо використовувати, оскільки в ній відсутні засоби (інформаційні входи та відповідні логічні елементи) для запису інформації. Проте проста модифікація схеми на рис. 36 приводить до схеми *RS*-тригера, який застосовується на практиці. Його ми розглянемо далі.

3.1.1. RS-тригер

Однією з найпростіших послідовнісних схем є *RS*-тригер (від англ. *Reset* і *Set*), яка складається з двох перехресно включених елементів АБО-НІ. Такий тригер має два входи – R та S і два виходи Q та $\overline{Q}$ (рис. 37). Принципи роботи *RS*-тригера і схеми із циклічно включеними інверторами аналогічні, але стан *RS*-тригера контролюється входами R та S, які скидають і встановлюють вихід Q.



а)



б)

S	R	Q	$\overline{Q}$	Номер комбінації
0	0	$Q_{попер}$	$\overline{Q}_{попер}$	1
0	1	0	1	2
1	0	1	0	3
1	1	0	0	4

в)

Рис. 37. RS-тригер: схема (а), позначення (б), таблиця істинності (в)

Для аналізу роботи *RS*-тригера розглянемо його поведінку за всіх можливих комбінацій вхідних сигналів:

1) $S = R = 0$. При $S = R = 0$ тригер працює так само, як і схема на рис. 36, і має два стабільні стани:

- якщо $Q = 1$, то $\bar{Q} = 0$, на входи верхнього елемента АБО-НІ надходять два 0, на його виході буде 1, як і було припущено;
- якщо $Q = 0$, то $\bar{Q} = 1$, на входи верхнього елемента АБО-НІ надходять 0 та 1, на його виході буде 0, як і було припущено.

Отже, схема дійсно має два стабільні стани, і поки $S = R = 0$ вона перебуває в одному з них й не може перейти в інший.

2) $S = 0$, $R = 1$. На вході верхнього елемента АБО-НІ буде як мінімум одна 1 – вхід R, а отже, вихід $Q = 0$. Обидва входи нижнього елемента АБО-НІ дорівнюють 0 ($Q = 0$ та $S = 0$), і тому вихід $\bar{Q} = 1$;

3) $S = 1$, $R = 0$. На вхід нижнього елемента АБО-НІ надходить принаймні одна одиниця S, тому вихід $\bar{Q} = 0$. На обидва входи верхнього елемента АБО-НІ надходить 0, отже $Q = 1$;

4) $S = 1$, $R = 1$. Якщо на входах обох елементів АБО-НІ буде по логічній 1, то на їх виходах будуть 0, $Q = \bar{Q} = 0$.

Таким чином, комбінації вхідних сигналів 2 та 3 дозволяють керувати станом тригера: коли на вхід S надходить 1, вихід Q стає 1, а $\bar{Q}$ – 0, коли на вхід R надходить 1, вихід Q стає 0, а $\bar{Q}$ – 1. Якщо ні на один із входів не надходить логічна 1 (комбінація 1), то на обох виходах зберігається попереднє значення $Q_{\text{попер}}$, яке було встановлене раніше комбінацією 2 чи 3. Комбінація 4 вважається некоректною, і тому зазвичай не використовується.

Так само, як і інвертори включені в кільце, так і *RS*-тригер є бістабільним елементом з 1 бітом стану Q. Станом можна керувати за допомогою входів R та S. Коли на R надходить 1, вихід скидається в 0. Коли високий рівень приходить на S, вихід встановлюється в 1. Якщо ні на один вхід не прийшла логічна 1,

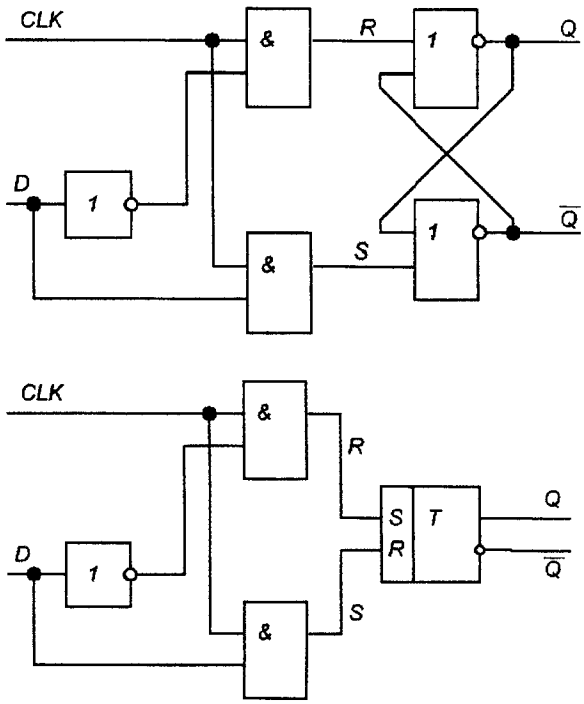
тригер зберігає свій попередній стан, значення виходів не змінюється. Таким чином, попередні значення вхідних визначає одну змінну стану Q, тригер зберігає 1 біт. Стан тригера визначається тільки тим, яка була остання операція: скидання або установка.

3.1.2. D-тригер-фіксатор

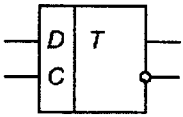
D-тригер-фіксатор дуже поширений елемент у цифровій схемотехніці. Недоліком RS-тригера є некоректна поведінка у випадку $S = 1$, $R = 1$ і нерозділеність питань "коли" і "як" у контексті зміни стану тригера. Подання логічної одиниці на вхід S чи R визначає не лише, ЩО станеться, але і КОЛИ це станеться. При проектуванні послідовнісних схем бажано мати тригер з двома принципово різними вхідними сигналами: один вхід визначає інформацію, яка буде записана в тригер, а другий – коли це відбудеться. Це зроблено в D-тригері-фіксаторі (рис. 38). Вхід даних D визначає, яким буде наступний стан, а вхід тактового сигналу CLK визначає, коли цей стан зміниться.

Якщо $CLK = 0$, то $R = S = 0$ і незалежно від значення D, Q зберігає своє попереднє значення. Якщо $CLK = 1$, то залежно від D на виході одного елемента ТА буде 1, а на виході іншого – 0. Вихідна частина D-тригера-фіксатора є RS-тригером, тобто, якщо $CLK = 1$, то $Q = D$. У D-тригері-фіксаторі некоректна поведінка при одночасному поданні сигналів скидання й установки ($R = S = 1$) виключена.

Таким чином, тактовий сигнал CLK контролює, КОЛИ дані проходять через D-тригер-фіксатор. Коли $CLK = 1$, він "прозорий", тобто вхідні дані D проходять на вихід Q, як через звичайний буфер. Поки $CLK = 1$, стан D-тригера-фіксатора змінюється неперервно. Коли $CLK = 0$, D-тригер-фіксатор "непрозорий", він не пропускає нові дані з входу D на вихід Q, а Q зберігає своє попереднє значення. Такий тригер також називають D-тригером, що керується рівнем сигналу, синхронним фіксатором, прозорим фіксатором.



а)



б)

CLK	D	S	R	Q	$\overline{Q}$
0	–	0	0	$Q_{попер}$	$\overline{Q}_{попер}$
1	0	0	1	0	1
1	1	1	0	1	0

в)

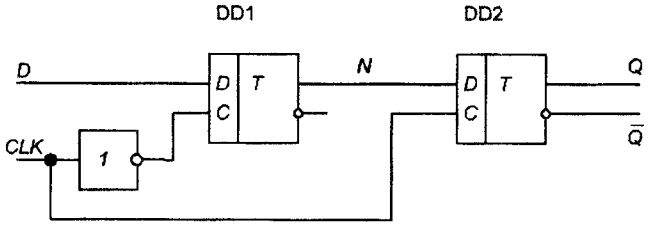
Рис. 38. *D*-тригер-фіксатор: схема (а), позначення (б), таблиця істинності (в)

Практика проектування цифрових пристроїв показує, що бажано розробляти схеми, стан яких змінюється тільки в певні моменти часу, які визначаються спеціальними тактовими сигналами. Далі розглянемо дуже важливий елемент таких схем – *D*-тригер, що синхронізується фронтом.

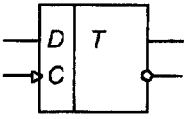
3.1.3. D-тригер

D-тригер синхронізується фронтом сигналу, тобто він змінює свій стан тільки при зміні сигналу *CLK* з 0 до 1. Цим він відрізняється від *D*-тригера-фіксатора, стан якого змінюється неперервно, поки *CLK* = 1. Для більшості практичних застосувань синхронізація фронтом більш зручна.

D-тригер може бути побудований з двох послідовно включених *D*-тригерів-фіксаторів. Як показано на рис. 39, *а*, тактові сигнали, які подаються на них, є булевими доповненнями один одного. Перший *D*-тригер-фіксатор (*DD*1) називають *ведучим* (*master*), а другий (*DD*2) – *веденим* (*slave*). Умовне позначення *D*-тригера наведено на рис. 39, *б*.



а)



б)

Рис. 39. *D*-тригер: схема (а), позначення (б)

Коли $CLK = 0$, master-фіксатор відкрито, а slave-фіксатор закрито. Отже, значення з входу D проходить до кола N. Коли $CLK = 1$, master-фіксатор закривається, а slave-фіксатор відкривається. Значення з N проходить на вихід Q, але N відключається від D. Отже, те значення, яке було на вході D безпосередньо перед переходом CLK з 0 у 1, одразу потрапляє на вихід Q після того, як тактовий сигнал встановлюється в 1. В інші моменти часу Q зберігає своє попереднє значення, оскільки закритий фіксатор (чи перший чи другий) постійно блокує шлях між D і Q. Таким чином, D-тригер копіює значення з D на Q по передньому фронту тактового імпульсу і зберігає всю його решту часу. Вхід D визначає наступний стан тригера. Фронт визначає момент часу, коли стан буде оновлений.

D-тригер також називають MS-тригером, master-slave-тригером і тригером, що синхронізується фронтом. Трикутник у позначенні вказує на те, що вхід синхронізується фронтом.

3.1.4. Тригери із функцією скидання, встановлення та дозволу

У тригері із функцією скидання є ще один вхід R (*reset*, скидання). Коли на R надходить 0, такий тригер працює як звичайний D-тригер. Коли на R надходить 1, тригер ігнорує вхід D і встановлює свій стан в 0. Аналогічно, вхід S (*set*, встановлення, за наявності) дозволяє встановити стан тригера в 1. Тригери з функцією скидання та встановлення використовуються, наприклад, коли треба встановити певний стан в усіх тригерах системи після вмикання живлення або після її скидання. Умовне позначення такого тригера показано на рис. 40. Такі тригери можуть скидатися і встановлюватися як синхронно, так і асинхронно. Тригери першого типу скидаються і встановлюються тільки по фронту сигналу CLK. Тригери другого типу скидаються і встановлюються одразу при надходженні логічної 1 на вхід R чи S незалежно від тактового сигналу.

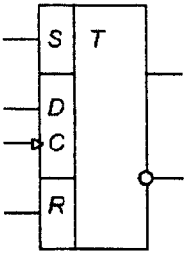


Рис. 40. D-тригер з функціями скидання та встановлення

Деякі тригери (рис. 41) мають ще один вхід – *E* (*enable*, дозволити). Цей вхід визначає, чи будуть дані завантажені по фронту сигналу *CLK*, чи ні. Коли на *E* подається логічна 1, то такий тригер працює так само, як і звичайний *D*-тригер. Якщо на *E* надходить логічний 0, то тригер ігнорує тактовий сигнал і зберігає свій стан. Можливу реалізацію такого тригера зображено на рис. 41, *б*): вхідний мультиплексор вибирає, чи подавати нові дані на вхід *D*, якщо на $E = 1$, чи подавати на вхід *D* старе значення з виходу *Q* тригера, якщо $E = 0$. Такі тригери використовуються, якщо треба завантажувати значення в тригер не по кожному фронту тактового сигналу, а тільки при виконанні деякої умови.

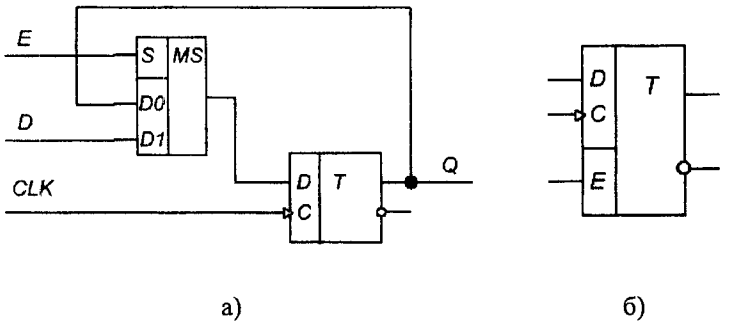
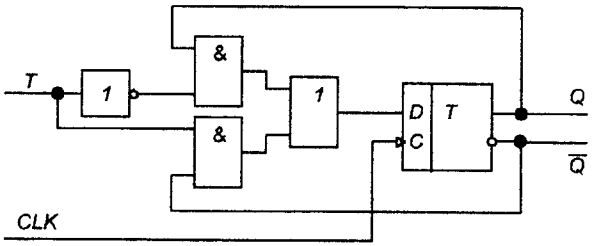


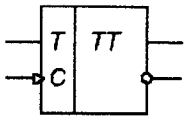
Рис. 41. *D*-тригер із входом дозволу: схема (а), позначення (б)

3.1.5. Т-тригер

На основі *D*-тригера проектується багато корисних схем цифрової електроніки, зокрема, *T*-тригер, схему якого та умовне позначення зображено на рис. 42. Ця схема є *D*-тригером зі зворотним зв'язком. Залежно від входу *T* на інформаційний вхід *D*-тригера подається його вихідний сигнал Q (якщо $T = 0$) або інверсія цього сигналу $\overline{Q}$ (якщо $T = 1$). Таким чином, поведінка тригера при надходженні переднього фронту тактового сигналу *CLK* залежить від *T*: якщо $T = 1$, стан тригера змінюється на протилежний по кожному передньому фронту, якщо $T = 0$, стан не змінюється. *T*-тригери застосовуються в лічильниках і схемах ділення частоти.



а)



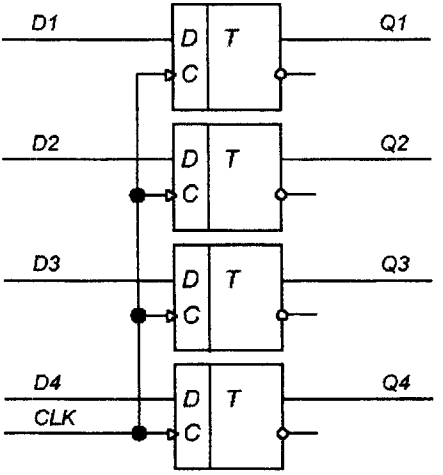
б)

Рис. 42. *T*-тригер: схема (а), позначення (б)

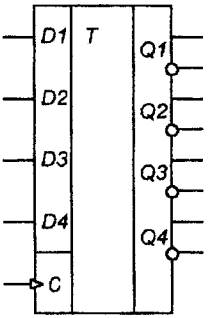
3.1.6. Регістр

N-розрядний регістр – це набір з *N* тригерів із загальним тактовим сигналом (і входами скидання, встановлення та дозволу, якщо вони є). Таким чином, усі біти регістру оновлюються одночасно. Регістр є основним блоком при побудові більшості по-

слідовнісних схем. На рис. 43 наведено схему й умовне позначення чотирирозрядного регістру із входами $D4{:}1$ і виходами $Q4{:}1$. $D4{:}1$ та $Q4{:}1$ є чотирирозрядними шинами.



а)



б)

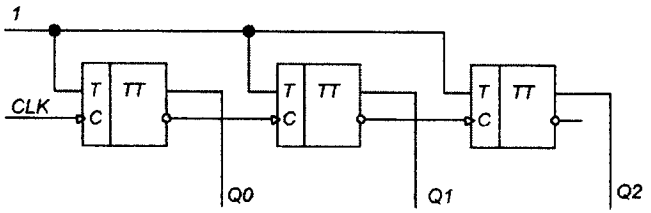
Рис. 43. Чотирирозрядний регістр: схема (а), позначення (б)

3.2. Базові послідовнісні схеми

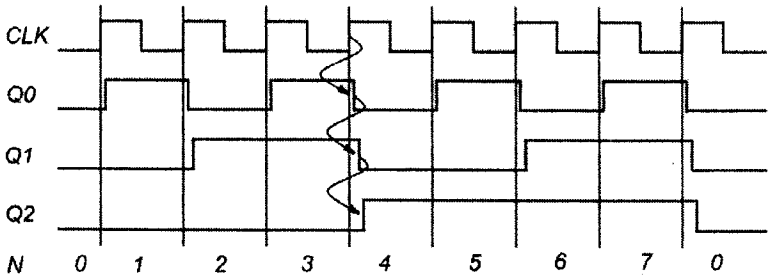
3.2.1. Лічильники

N-розрядний двійковий лічильник являє собою послідовнісну схему, яка має вхід тактового сигналу, *N*-розрядний вихід *Q*, і, можливо, входи завантаження чи скидання. Вихід лічильника послідовно набуває всіх 2^N можливих значень *N*-розрядного двійкового числа, перехід до наступного значення відбувається по передньому фронту тактового імпульсу. Залежно від напрямку зміни стану розрізняють такі лічильники: 1) що підсумовують (*N*-розрядне двійкове число збільшується по фронту тактового імпульсу); 2) що віднімають (число зменшується по фронту тактового імпульсу)); 3) реверсивні, у яких є спеціальний вхід, що керує напрямком відліку.

Найпростішим є асинхронний лічильник (рис. 44). Основою такого лічильника є *T*-тригер (п. 3.1.5), який змінює свій стан на протилежний по передньому фронту кожного вхідного імпульсу. Аналогічно можна побудувати лічильник, який віднімає (рис. 45). Назва таких лічильників зумовлена тим, що тригери в них з'єднані послідовно, а отже, їх стан змінюється в часі також послідовно, усі розряди лічильника перемикаються неодночасно. Це призводить до великих затримок перемикання, наприклад, якщо розрядів багато, а також до появи некоректних сигналів на виході лічильника, коли одна частина тригерів уже перемкнулася, а інша ще не встигла перемкнутися.

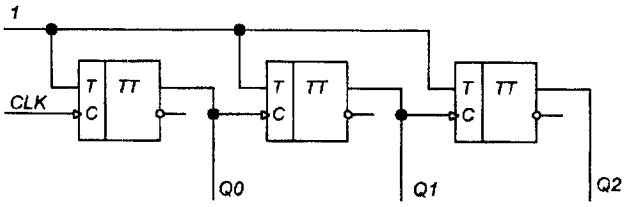


а)

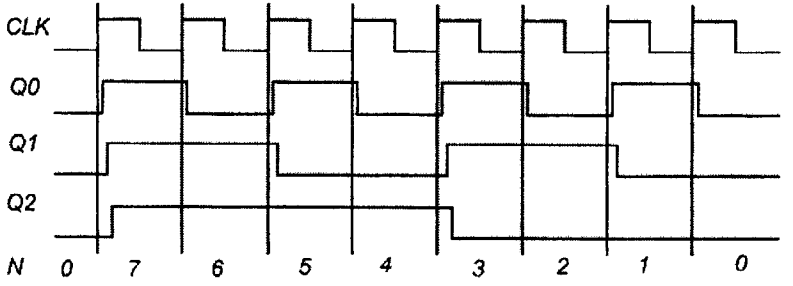


б)

Рис. 44. Трирозрядний асинхронний підсумовуючий лічильник: схема (а), часова діаграма (б)



а)



б)

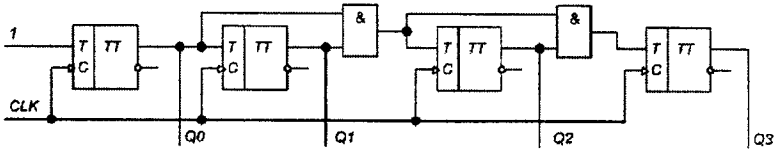
Рис. 45. Трирозрядний асинхронний лічильник, що віднімає: схема (а), часова діаграма (б)

Розглянемо схему синхронного лічильника, у якому всі тригери будуть перемикатися одночасно по передньому фронту кожного тактового імпульсу (рис. 46). Бачимо, що в підсумовую-

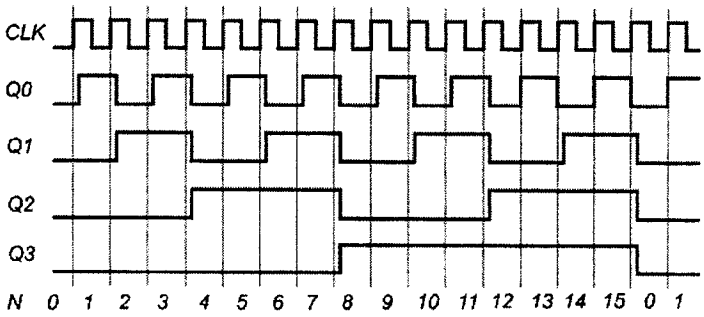
чому лічильнику перемикання розряду з номером i відбувається лише за наявності в усіх попередніх розрядах 1. Отже, на керуючий вхід i-го тригера необхідно подати сигнал

$$T_i = Q0 * Q1 * \ldots * Q(i-1).$$

Цей сигнал перенесення можна сформувати за допомогою багатовхідного кон'юнктора, кількість входів якої визначається номером розряду, для якого формується перенесення, а можна за допомогою двовхідних кон'юнкторів у кожному розряді. Відповідна схема синхронного лічильника показана на рис. 46. Для побудови реверсивного лічильника треба додати до кожного розряду лічильника мультиплексор, який подаватиме на вхід T-тригера цього розряду прямий або інверсний виходи тригера попереднього розряду.



а)



б)

Рис. 46. Чотирирозрядний синхронний підсумовуючий лічильник: схема (а), часова діаграма (б)

При використанні мікросхем програмованої логіки зазвичай реалізують лічильник, подібний зображеному на рис. 47. Цей N-розрядний лічильник складається із суматора та регістру. На кожному циклі суматор додає 1 до величини, яка зберігається в регістрі.

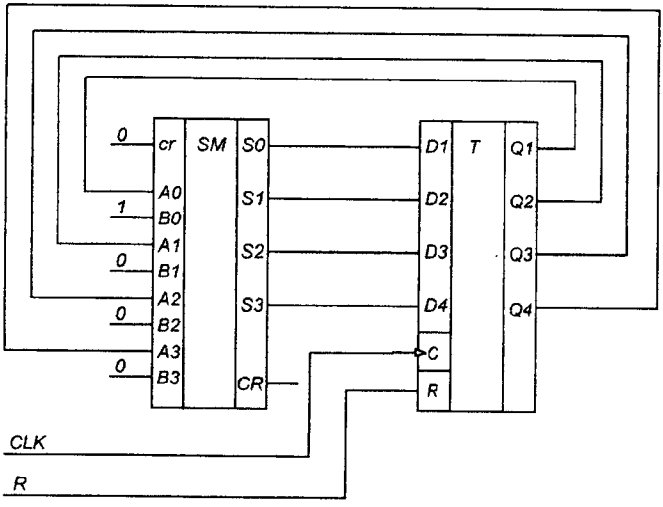


Рис. 47. Чотирирозрядний лічильник на основі суматора

3.2.2. Регістри зсуву

Регістр зсуву, який має вхід тактового сигналу, послідовний вхід S і N паралельних виходів $Q(N–1):0$ зображено на рис. 48. По кожному передньому фронту тактового імпульсу в перший тригер регістру записується новий біт із входу S, а вміст наступних тригерів зрушується вперед. Регістр зсуву можна розглядати як послідовно-паралельний перетворювач. На вхід S надходять послідовні дані (по одному біту за один період тактового сигналу). Після N циклів останні N значеннь вхідного сигналу можна паралельно зчитати з виходу Q.

Як показано на рис. 48, регістр зсуву може бути побудований із N послідовно з'єднаних D-тригерів. Інколи регістри зсуву мають вхід скидання для ініціалізації всіх тригерів. Також застосовуються паралельно-послідовні перетворювачі, у які паралельно завантажуються N бітів, після чого вони послідовно (по одному біту за раз) надходять на вихід.

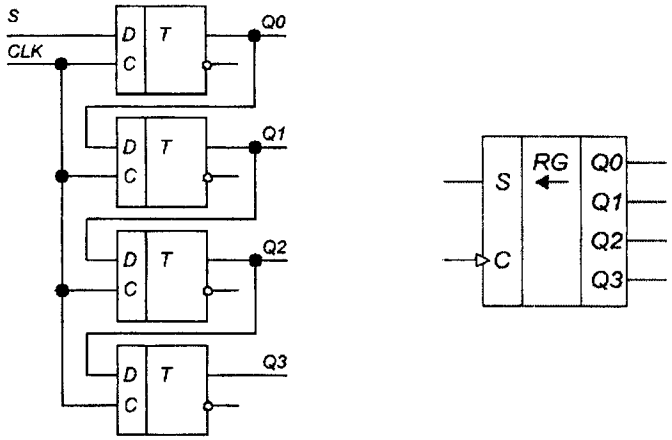


Рис. 48. Чотирирозрядний регістр зсуву: схема (а), позначення (б)

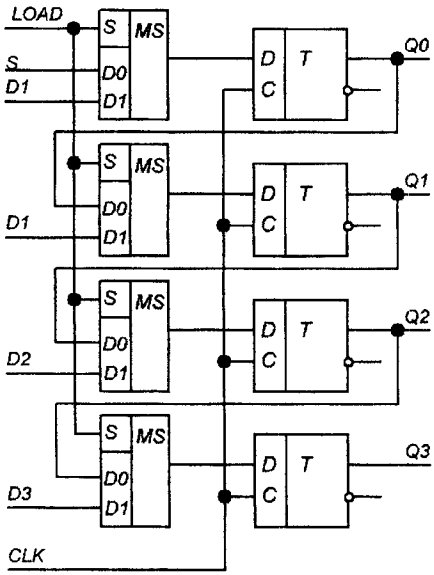


Рис. 49. Чотирирозрядний регістр зсуву з завантаженням

Схемотехніка паралельно-послідовного перетворювача і схемотехніка регістру зсуву подібні. Регістр зсуву можна модифікувати для виконання як послідовно-паралельного, так і паралельно-послідовного перетворення, якщо до нього додати паралельний вхід $D(N–1):0$, сигнал керування *LOAD* та N мультиплексорів, як показано на рис. 49. Коли вхід *LOAD* активований, у всі тригери паралельно завантажуються дані з багаторозрядного входу $D(N–1):0$.

3.3. Скінченні автомати

Майже всі последовнісні логічні схеми, які використовуються в сучасній електроніці, можна зобразити у формі, представленій на рис. 50 чи рис. 51. Такі представлення називають *скінченними автоматами*. Вони дістали свою назву через те, що схема з N-розрядним регістром може перебувати в одному з 2^N станів, тобто в скінченній їх кількості. Скінченний автомат має M входів, K виходів і N бітів станів. На вхід скінченного автомата також подається тактовий сигнал і, можливо, сигнал скидання. Скінченний автомат складається з двох блоків комбінаційної логіки: логіки переходу в наступний стан і логіки формування вихідного сигналу, а також із регістру, у якому зберігається поточний стан. По фронту кожного тактового імпульсу автомат переходить у наступний стан, який визначається поточним станом і значеннями на входах.

3.3.1. Автомати Мура і Милі

Існують два основні класи скінченних автоматів, які різняться своїми функціональними описами. В *автоматі Мура* вихідні значення залежать лише від поточного стану (рис. 50), тоді як в *автоматі Милі* вихід залежить як від поточного стану, так і від поточних вхідних даних (рис. 51). Скінченні автомати задають систематичний спосіб проектування синхронних послідовнісних схем за заданим функціональним описом. Існує цілий ряд програмних засобів, які спрощують таке проектування.

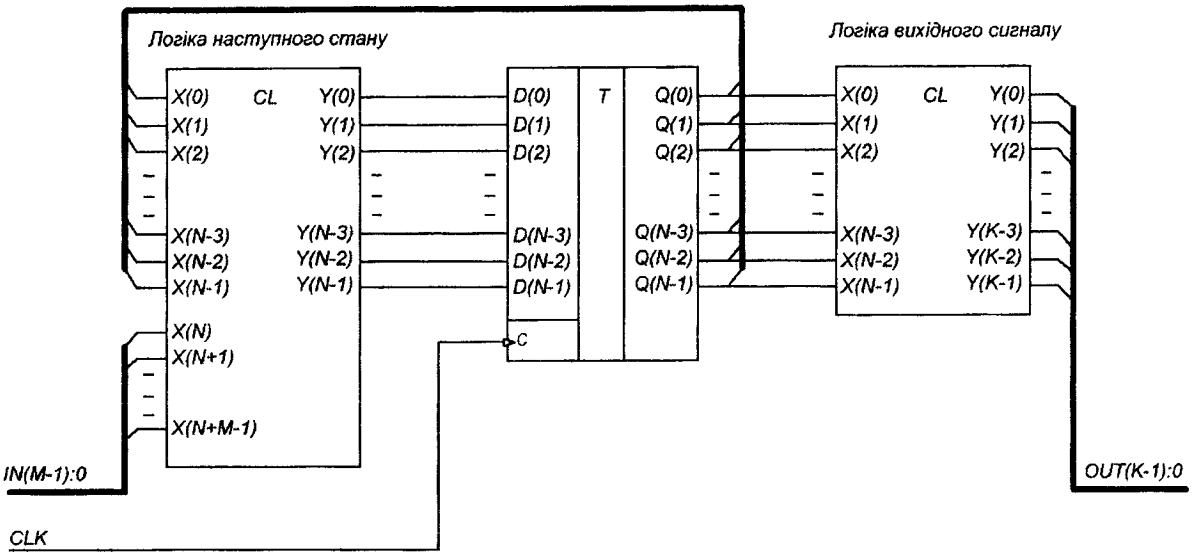


Рис. 50. Скінченний автомат Мура

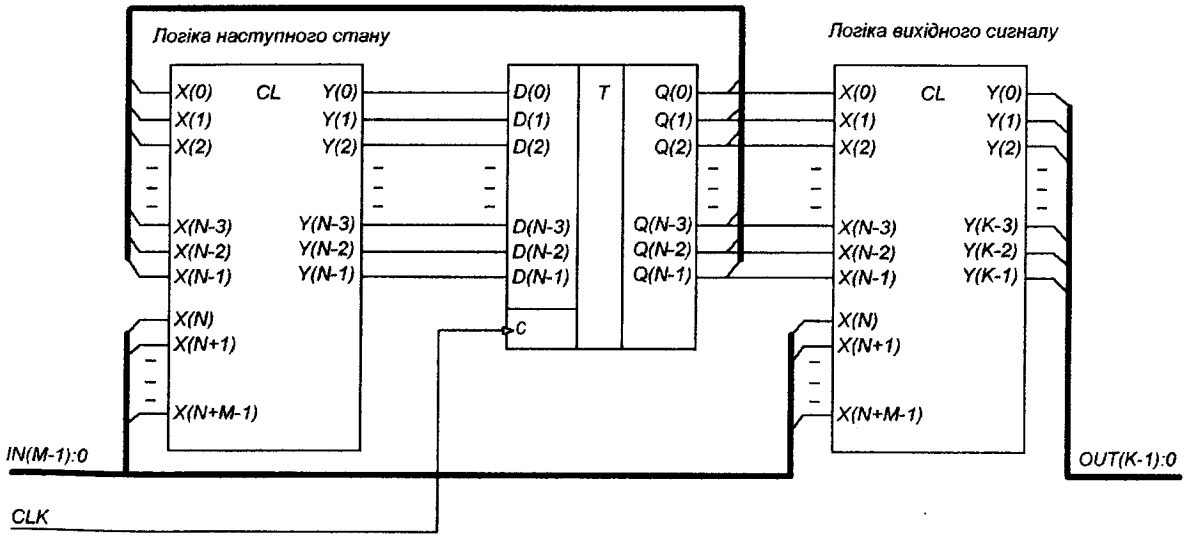


Рис. 51. Скінченний автомат Мілі

3.3.2. Стани автомата. Граф переходів

Як приклад розглянемо процес проектування скінченного автомата, який буде задовольняти таким вимогам:

1. Автомат має вхід тактового сигналу *CLK*, сигналу скидання *RESET*, інформаційний вхід *X* і вихід *Z*;

2. Зміна стану автомата відбувається по передньому фронту кожного тактового імпульсу *CLK*;

3. Вихід *Z* дорівнює 1, якщо протягом двох останніх періодів тактового сигналу вхід *X* дорівнював 1. Якщо ця умова не виконується, *Z* дорівнює 0.

Приклад бажаної роботи цього автомата наведено в табл. 14.

Таблиця 14

N тактового імпульсу	0	1	2	3	4	5	6	7	8	9	10	11	12
X	0	1	0	1	1	0	1	1	1	0	1	1	1
Z	0	0	0	0	0	1	0	0	1	1	0	0	1

Спочатку необхідно визначити, скільки станів потрібно для реалізації автомата і за яких умов відбуваються переходи між цими станами. У нашому прикладі система перебуває в початковому стані *A* поки на вхід *X* надходять тільки 0, вихід *Z* дорівнює 0. Коли на вхід *X* надходить 1, система має перейти в інший стан, стан *B*. У стані *B* вихід *Z* також дорівнює 0. У стані *B* система має відреагувати на значення вхідного сигналу *X* у момент надходження фронту наступного тактового імпульсу. Якщо $X = 0$, система має повернутися у початковий стан *A*, а якщо $X = 1$, перейти в наступний стан, стан *C*, і видати на вихід 1, $Z = 1$. Система має залишатися у стані *C* доти, поки $X = 1$, а якщо $X = 0$, повернутися в початковий стан *A*.

Описані вище вербально правила переходу можна зобразити за допомогою графа, який наведено на рис. 52.

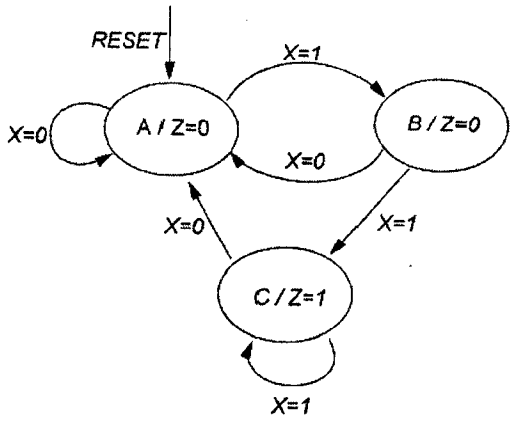


Рис. 52. Граф скінченного автомата

Граф має три вузли, які відповідають трьом станам системи. Для кожного вузла вказано значення вихідного сигналу. Переходи між вузлами показані стрілками, біля яких вказані умови, за яких ці переходи відбуваються. Якщо за деякої умови стан системи не має змінюватися, відповідна стрілка починається і закінчується на одному вузлі. Також на графі показано сигнал скидання *RESET*, який переводить систему в початковий стан *A* незалежно від інших умов. Окрім графів для опису скінченних автоматів використовують таблиці станів (подібні табл. 15 для скінченного автомата, що розробляється), у яких для всіх станів автомата вказується наступний стан за всіх комбінацій значень вхідних сигналів і значення вихідного сигналу.

Таблиця 15

Поточний стан	**Наступний стан**		**Вихід *Z***
	$X = 0$	$X = 1$	
A	*A*	*B*	0
B	*A*	*C*	0
C	*A*	*C*	1

Оскільки в скінченному автоматі, що проектується, є тільки три стани, то для їх кодування можна вибрати простий дворoз-

рядний ($Y2$, $Y1$) двійковий код, причому комбінація $Y2 = Y1 = 1$ не використовуватиметься. Для збереження стану знадобляться два тригери, де стану A відповідає $Y2 = Y1 = 0$, стану B – $Y2 = 0$, $Y1 = 1$, а стану C – $Y2 = 1$, $Y1 = 0$. Слід зазначити, що це не єдиний можливий спосіб кодування станів скінченного автомата.

Таблицю станів з вибраним кодуванням наведено в табл. 16, з якої видно, що вихід автомата залежить тільки від його стану, отже, буде спроектовано скінченний автомат Мура, а регістр для збереження стану автомата складатиметься з двох D-тригерів. Вихідні сигнали логіки визначення наступного стану ($Y1N$, $Y2N$) надходять на вхід цих тригерів. Оскільки комбінація змінних станів $Y1 = Y2 = 1$ не використовується, то наступний стан, що їй відповідає, не може виникнути. У таблиці істинності для відповідних нових значень змінних стану використовують байдуже значення, $Y1N$ = "–", $Y2N$ = "–", при проектуванні для мінімізації булевих виразів його За таблицею станів (табл. 16) легко заповнити таблиці істинності логіки визначення наступного стану (табл. 17) та логіки визначення вихідного сигналу (табл. 18).

Таблиця 16

Поточний стан		**Наступний стан**				**Вихід Z**
		$X = 0$		$X = 1$		
	$Y2;Y1$		$Y2N;Y1N$		$Y2N;Y1N$	
A	00	A	00	B	01	0
B	01	A	00	C	10	0
C	10	A	00	C	10	1
	11		–		–	–

Таблиця 17

X	$Y2$	$Y1$	$Y2N$	$Y1N$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	–	–
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	–	–

Таблиця 18

Y2	Y1	Z
0	0	0
0	1	0
1	0	1
1	1	–

Після мінімізації (з урахуванням байдужих значень) можна отримати прості вирази для $Y1N$, $Y2N$, Z:

$$Y1N = X * \overline{Y1} * \overline{Y2},$$

$$Y2N = X * (Y1 + Y2),$$

$$Z = Y2$$

і побудувати принципову схему скінченного автомата (рис. 53).

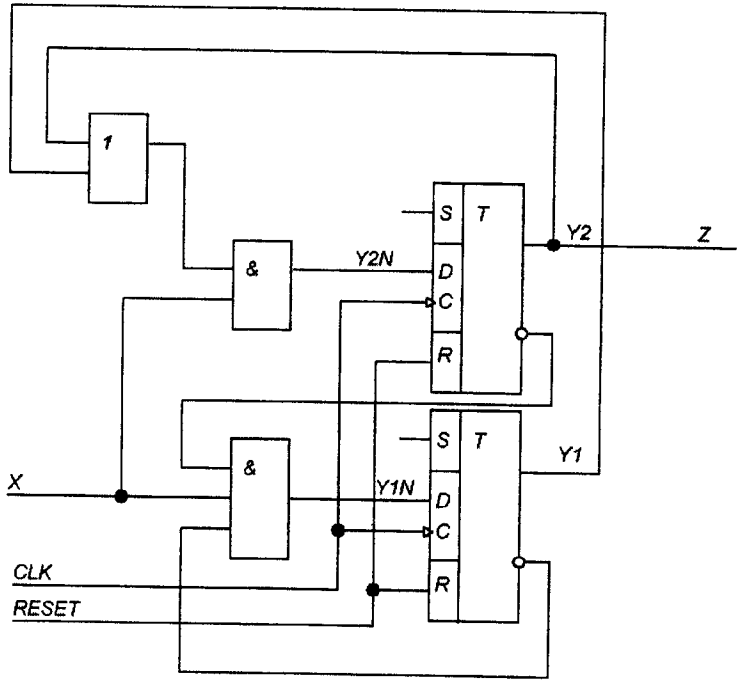


Рис. 53. Принципова схема скінченного автомата

Схема має вхід скидання *RESET*, який скидає обидва тригери, тобто переводить автомат у початковий стан *A*. Робота автомата ілюструється часовою діаграмою, що наведена на рис. 54. Видно, що автомат відповідає описаній вище специфікації.

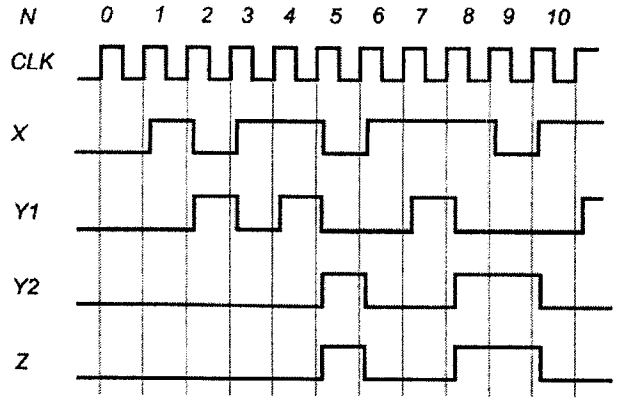


Рис. 54. Часова діаграма скінченного автомата

Скінченні автомати є потужним інструментом для систематичного проектування послідовнісних схем за заданою специфікацією. Для проектування автомата необхідно виконати таку послідовність дій:

1) визначити вхідні та вихідні сигнали автомата;
2) визначити стани автомата й умови переходів між ними. Створити граф автомата;
3) обрати метод кодування станів, від якого залежить складність схемної реалізації;
4) скласти таблицю переходів і таблицю виходів;
5) виконати мінімізацію булевих функцій і скласти вирази логіки наступного стану та вихідних сигналів;
6) зобразити принципову схему скінченного автомата.

3.3.3. Кодування станів

У попередньому прикладі кодування станів і виходів було обране довільно. Вибір іншого кодування привів би до іншої

схеми. При проектуванні скінченного автомата основна проблема полягає в способі обрання кодування, при якому буде використовуватися найменша кількість елементів, а швидкодія схеми буде максимальною. На жаль, простого способу знайти найкраще кодування не існує, окрім простого перебору всіх варіантів, що майже неможливо, якщо кількість станів велика. Зазвичай можна знайти ефективне кодування таким чином, щоб зв'язані стани або виходи мали загальні біти. Під час пошуку набору можливих кодувань і вибору найбільш раціональної з них використовують системи автоматизованого проектування (САПР).

Як приклад розглянемо два варіанти повторного проектування скінченного автомата (його граф наведено на рис. 52), які різняться кодуванням станів. У першому варіанті буде змінене тільки кодування стану C: $Y2 = Y1 = 1$ (було $Y2 = 1$, $Y1 = 0$), а комбінація $Y2 = 1$, $Y1 = 0$ не використовуватиметься. Таблицю станів з новим кодуванням наведено в табл. 19. Легко побудувати таблиці істинності логіки визначення наступного стану (табл. 20) і логіки визначення вихідного сигналу (табл. 18).

Таблиця 19

Поточний стан		**Наступний стан**				**Вихід Z**
		$X = 0$		$X = 1$		
	$Y2;Y1$		$Y2N;Y1N$		$Y2N;Y1N$	
A	00	A	00	B	01	0
B	01	A	00	C	11	0
C	11	A	00	C	11	1
	10		–		–	–

Таблиця 20

X	$Y2$	$Y1$	$Y2N$	$Y1N$
0	0	0	0	0
0	0	1	0	0
0	1	0	–	–
0	1	1	0	0
1	0	0	0	1
1	0	1	1	1
1	1	0	–	–
1	1	1	1	1

Таблиця 21

Y2	Y1	Z
0	0	0
0	1	0
1	0	-
1	1	1

Вказаним таблицям відповідають такі вирази для $Y1N$, $Y2N$, Z:

$$Y1N = X,$$

$$Y2N = X * Y1,$$

$$Z = Y2.$$

Принципову схему автомата, що відповідає табл. 19, зображено на рис. 55. Порівнюючи рис. 53 і 55, можна легко помітити, що кодування A: $Y2 = Y1 = 0$, B: $Y2 = 0$ $Y1 = 1$ і C: $Y2 = Y1 = 1$ приводить до суттєво простішої схеми, ніж кодування A: $Y2 = Y1 = 0$, B: $Y2 = 0$ $Y1 = 1$ і C: $Y2 = 1$ $Y1 = 0$.

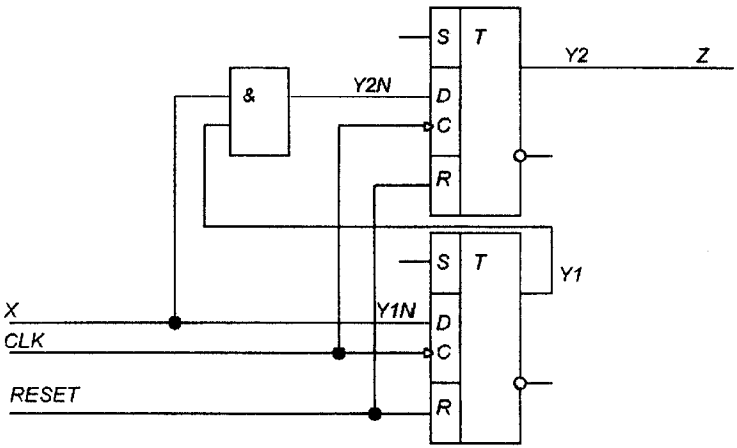


Рис. 55.Принципова схема скінченного автомата з альтернативним кодуванням станів

Одним із важливих рішень при проектуванні скінченного автомата є вибір між двійковим кодуванням станів (00, 01, 10) і прямим кодуванням (001, 010, 100), яке ще називають кодуванням "1 із N" (*one-hot*). При двійковому кодуванні, яке використаному в попередніх прикладах, кожному стану було поставлено

у відповідність двійкове число (номер цього стану). Очевидно, що автомат із K станами потребує $\log_2 K$ бітів стану (і відповідну кількість тригерів для його збереження).

При прямому кодуванні для кожного стану використовується один біт стану. Отже, у скінченного автомата з прямим кодуванням і трьома станами коди станів будуть 001, 010 і 100. Кожний біт стану зберігається в тригері. Таким чином, пряме кодування вимагає більшої кількості тригерів, ніж двійкове. Проте при використанні прямого кодування логіка визначення наступного стану і логіка формування вихідних сигналів зазвичай спрощується. Найкращий вибір кодування залежить від особливостей конкретного автомата.

У другому альтернативному варіанті скінченного автомата Рис. використовується пряме кодування. У табл. 22 наведено таблицю станів при використанні прямого кодування, а в табл. 23 та 24, таблиці істинності логіки визначення наступного стану і логіки визначення вихідного сигналу.

Таблиця 22

Поточний стан		**Наступний стан**				**Вихід** Z
		$X=0$		$X=1$		
	$Y3;Y2;Y1$		$Y3N;Y2N;Y1N$		$Y3N;Y2N;Y1N$	
A	001	A	001	B	010	0
B	010	A	001	C	100	0
C	100	A	001	C	100	1

Таблицям істинності (табл. 23 і 24) відповідають такі булеві вирази:

$$Y1N = \overline{X},$$
$$Y2N = X * Y1,$$
$$Y3N = X * \overline{Y1},$$
$$Z = Y3.$$

Видно, що наступний стан автомата і його вихід не залежать від $Y2$, а отже, його можна не обчислювати і не зберігати та зекономити один тригер і один логічний елемент ТА.

Таблиця 23

X	*Y3*	*Y2*	*Y1*	*Y3N*	*Y2N*	*Y1N*
0	0	0	0	–	–	–
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	–	–	–
0	1	0	0	0	0	1
0	1	0	1	–	–	–
0	1	1	0	–	–	–
0	1	1	1	–	–	–
1	0	0	0	–	–	–
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	0	1	1	–	–	–
1	1	0	0	1	0	0
1	1	0	1	–	–	–
1	1	1	0	–	–	–
1	1	1	1	–	–	–

Таблиця 24

Y3	*Y2*	*Y1*	*Z*
0	0	0	–
0	0	1	0
0	1	0	0
0	1	1	–
1	0	0	1
1	0	1	–
1	1	0	–
1	1	1	–

Принципову схему автомата з прямим кодуванням наведено на рис. 56. У ній використовується на один елемент більше, ніж у схемі з альтернативним двійковим кодуванням, яку наведено на рис. 55. Отже, кодування *A*: $Y2 = Y1 = 0$; *B*: $Y2 = 0$, $Y1 = 1$; *C*: $Y2 = Y1 = 1$ оптимальні. У загальному випадку, оптимальне кодування залежить від особливостей конкретного скінченного автомата.

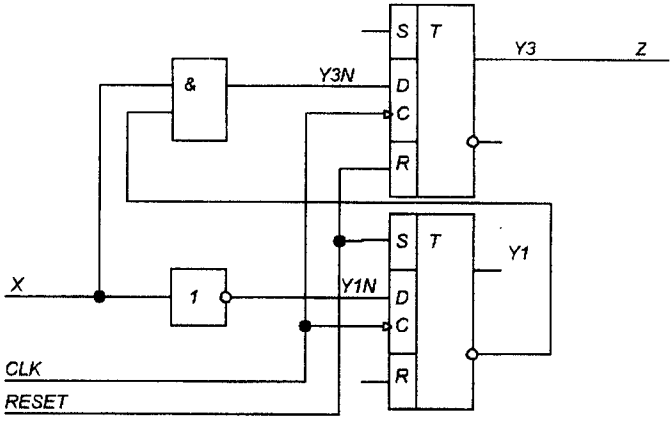


Рис. 56. Принципова схема скінченного автомата з прямим кодуванням станів

3.3.4. Автомати Милі

Як приклад розглянемо процес проектування автомата Милі, який аналогічний автомату Мура (п. 3.3.2). Із табл. 14 видно, що в розглянутому вище автоматі Мура значення вихідного сигналу, який відповідає двом послідовним 1 на вході автомата, виникає на наступному періоді тактового сигналу після другої 1 на вході. Тепер буде спроектовано автомат, який видає 1 на вихід одночасно з появою другої 1 на вході. У цьому автоматі вихід залежатиме від поточних вхідних даних, а отже, це буде автомат Милі. Приклад бажаної роботи нового автомата наведено в табл. 25. Для цього автомата достатньо використати тільки два стани: A – на вхід надійшов 0, B – на вхід системи надійшла 1. Вихідний сигнал $Z = 1$, якщо система перебуває в стані B (тобто на її вході на попередньому періоді тактового сигналу була 1) і на її вході 1. На рис. 57 показано граф автомата Милі, який відповідає цій специфікації. Вихідні сигнали автомата Милі вказують біля стрілок переходів, оскільки вони залежать як від стану,

з якого цей перехід відбувається, так і від вхідного сигналу, який вказано на цій самій стрілці переходу.

Таблиця 25

N тактового імпульсу	0	1	2	3	4	5	6	7	8	9	10	11	12
X	0	1	0	1	1	0	1	1	1	0	1	1	1
Z	0	0	0	0	1	0	0	1	1	0	0	1	1

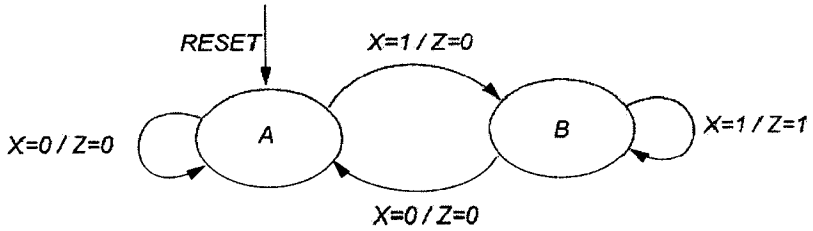


Рис. 57. Граф скінченного автомата Милі

Таблицю станів автомата Милі, що відповідає графу на рис. 57, наведено в табл. 26.

Таблиця 26

Поточний стан	Наступний стан		Вихід *Z*	
	X = 0	*X* = 0	*X* = 0	*X* = 0
A	*A*	*B*	0	0
B	*A*	*B*	0	1

Скінченний автоматі, що проектується, має тільки два стани, для кодування яких можна вибрати однорозрядний (*Y*) двійковий код, а для збереження тільки один тригер. Таблиця станів з обраним кодуванням наведено в табл. 27. Видно, що вихід *Z* залежить як від поточного стану, так і від входу *X*. За цією таблицею можна отримати дуже прості вирази для наступного стану автомата і його вихідного сигналу:

$$YN = X, \qquad Z = X * Y.$$

Таблиця 27

Поточний стан		Наступний стан				Вихід	
		$X=0$		$X=1$		$X=0$	$X=1$
	Y		YN		YN	Z	Z
A	0	A	0	B	1	0	0
B	1	A	0	B	1	0	1

Принципову схему скінченного автомата Милі, який відповідає табл. 27, наведено на рис. 58.

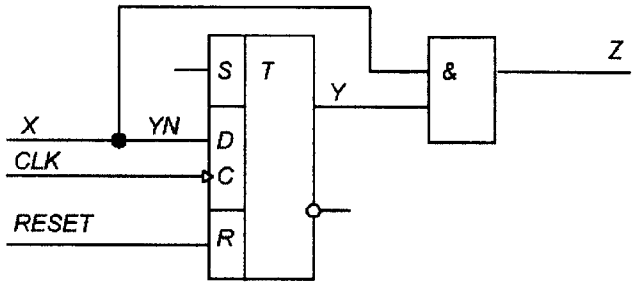


Рис. 58. Принципова схема скінченного автомата Милі

Зазвичай скінченні автомати Милі мають менше станів і простішу принципові схему, ніж функціонально аналогічні автомати Мура. Це зумовлено тим, що можливість використовувати поточні входи для визначення вихідних сигналів надає розробнику додаткову гнучкість при прийнятті проектних рішень.

3.4. Синхронізація послідовнісних схем

Як зазначено в п. 3.1.3, *D*-тригер копіює сигнал зі свого входу на вихід по передньому фронту тактового імпульсу. Цей процес називають *фіксацією* (*sampling*) *сигналу по фронту тактового імпульсу*. Поведінка тригера вважається коректною, якщо сигнал на вході стабільний (дорівнює 0 або 1 і не змінюється) протягом часового інтервалу, який охоплює передній фронт так-

тового імпульсу. Якщо вхідний сигнал не буде стабільним протягом цього інтервалу, поведінка тригера стане непередбачуваною, такої ситуації при проектуванні пристроїв слід уникати.

Частину часового інтервалу, коли вхід послідовнісного елемента має бути стабільним, перед фронтом тактового імпульсу називають *часом передустановлення* (*setup time*), а після фронту – *часом утримання* (*hold time*). Отже, всі перехідні процеси в послідовнісній схемі мають завершитися до початку часу передустановлення і не повинні починатися до завершення часу утримання. Таким чином, період тактових імпульсів має бути досить великим, щоб перехідні процеси всіх сигналів встигли завершитися. Ця вимога обмежує швидкодію всієї системи.

Часові характеристики послідовнісної схеми. Приклад часових характеристик тригера (чи іншої послідовнісної схеми) наведено на рис. 59. Після переходу 0→1 тактового сигналу *CLK* (переднього фронту тактового імпульсу) вихід (чи декілька виходів) схеми може почати змінюватися не раніше ніж через час t_{ccq} (затримка реакції CLK-to-Q, *contamination delay CLK-to-Q*) і має набути стаціонарного значення не пізніше ніж через час t_{pcq} (затримка поширення CLK-to-Q, *propagation delay CLK-to-Q*)). Ці величини є найменшими і найбільшими затримками схеми, відповідно. Для того, щоб схема працювала коректно, її інформаційний вхід (або входи) має бути стабільним упродовж деякого часу передустановлення (*setup time*) t_{setup} перед переднім фронтом тактового імпульсу і не має змінюватися впродовж часу утримання (*hold time*) t_{hold} після цього фронту. Виконання цієї вимоги гарантує, що в процесі перемикання тригерів схеми значення на їх входах не змінюватимуться.

На рис. 60 показано типову структуру синхронної послідовнісної схеми. По передньому фронту тактового імпульсу на виході регістру $R1$ формується вихідний сигнал (чи сигнали) $Q1$. Ці сигнали надходять на вхід блоку комбінаційної логіки, вихідні сигнали якого, у свою чергу, надходять на вхід (чи входи) $D2$ регістру $R2$. По передньому фронту наступного тактового імпульсу в регістри записується нова інформація. Проаналізуємо вимоги до періоду тактового сигналу, які пов'язані із затримками у блоці комбінаційної логіки та часами передустановлення й

утримання. Саме ці умови обмежують тактову частоту системи і відповідно її швидкодію.

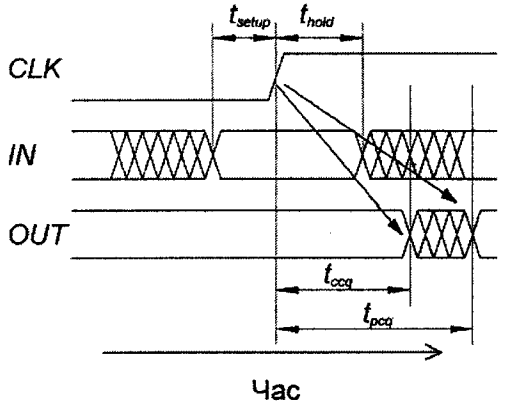


Рис. 59. Часові характеристики послідовнісної схеми

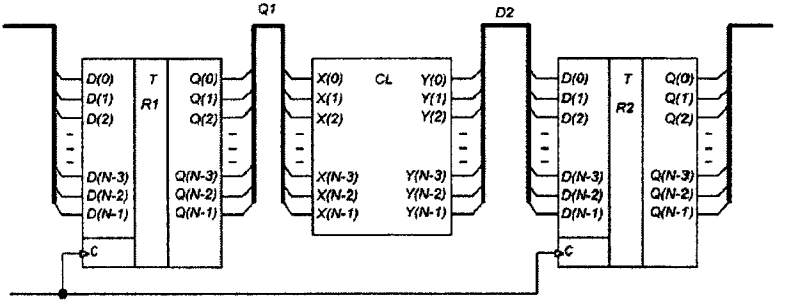


Рис. 60. Структура типової послідовнісної схеми

Як зображено на рис. 61, вихідний сигнал блоку комбінаційної логіки набуває остаточного значення через максимальний час затримки поширення від моменту встановлення його вхідного сигналу. Стрілками показано максимальні затримки поширення в тракті регістр $R1$–комбінаційна логіка.

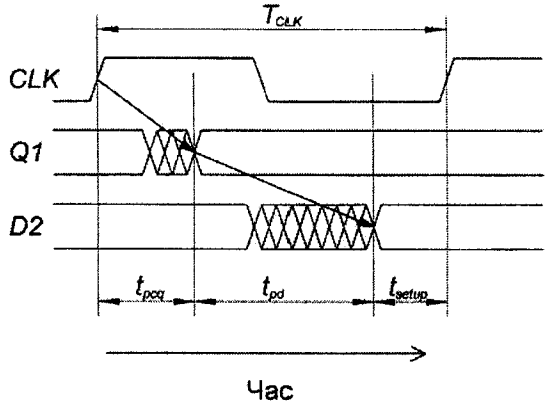


Рис. 61. Максимальна затримка між регістрами послідовнісної схеми

Для виконання обмеження за часом передустановлення регістру *R*2 сигнал *D*2 має встановитися не пізніше ніж за час передустановлення до фронту наступного тактового імпульсу. Таким чином, можна отримати вираз для мінімальної тривалості періоду синхросигналу:

$$T_{CLK} > t_{pcq} + t_{pd} + t_{setup}.$$

При проектуванні реальних приладів період тактового сигналу майбутнього виробу зазвичай задається з міркувань конкурентоспроможності. Окрім цього, характеристики тригерів: затримка поширення сигналу тригером від фронту тактового сигналу до виходу (CLK-to-Q) t_{pcq} і час передустановлення t_{setup} визначені виробником і не можуть бути змінені. Отже, максимальна затримка поширення комбінаційної схеми – єдиний параметр, який може змінювати розробник, для неї можна отримати таке обмеження:

$$T_{CLK} - t_{pd} - t_{setup} > t_{pcq}.$$

Якщо затримка поширення в комбінаційній схемі занадто велика, то вхід *D*2 може не встигнути прийняти свій стаціонарний

стан до початку інтервалу часу, у якому вхід регістру $R2$ має бути стаціонарним для забезпечення коректності його роботи. Таким чином, $R2$ буде працювати в заборонених умовах, а його поведінка буде непередбачуваною. Проблему можна розв'язати шляхом збільшення періоду тактового сигналу або повторним проектуванням комбінаційної схеми для зменшення затримки поширення.

Регістр $R2$ на рис. 60 має також обмеження часу утримання. Його вхід, $D2$, не повинен змінюватися впродовж часу t_{hold} після переднього фронту тактового імпульсу. Як показано на рис. 62, вихідний сигнал блоку комбінаційної логіки може почати змінюватися не раніше закінчення часу реакції після завершення зміни його вхідного сигналу.

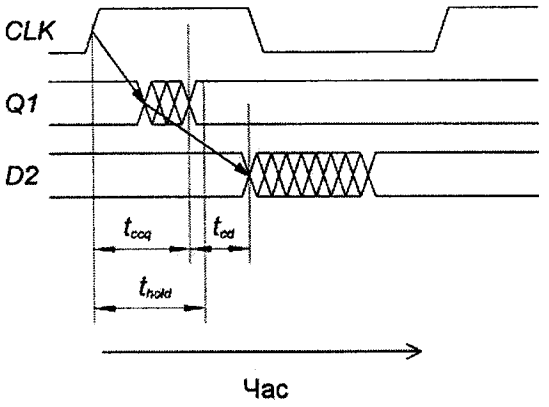


Рис. 62. Мінімальна затримка між регістрами послідовнісної схеми

Отже, $D2$ може почати змінюватися через $t_{ccq} + t_{cd}$ після переднього фронту тактового імпульсу (або пізніше). Таким чином, вимога стабільності вхідного сигналу тригера протягом часу утримання після переднього фронту тактового імпульсу приводить до нерівності

$$t_{ccd} + t_{cd} > t_{hold} \quad \text{або} \quad t_{cd} > t_{hold} - t_{ccd}.$$

Ці нерівності надають обмеження часу утримання або мінімальної затримки. Незважаючи на те, що обмеження часу утримання не впливають на тактову частоту системи, воно дуже важливе. Якщо воно порушуються, то єдиним шляхом розв'язання проблеми є збільшення затримки реакції комбінаційної схеми, що вимагає її повторного проектування.

Послідовнісні схеми мають обмеження часів передустановлення й утримання, які встановлюють максимальну та мінімальну затримки в комбінаційній логічній схемі між тригерами. Сучасні тригери зазвичай спроектовані таким чином, що затримка в комбінаційній логіці може бути нульовою, тобто тригери можуть бути розміщені безпосередньо один за одним. Максимальна затримка обмежує кількість послідовних логічних елементів, ввімкнених один за одним на критичному шляху швидкодіючої схеми.

У реальних системах тактові імпульси надходять на входи різних регістрів неодночасно. Цей розкид за часом, який називають *розфазуванням*, або *розкидом фаз тактового сигналу* (*clock skew*), примушує розробників додатково збільшувати період тактових сигналів і орієнтуватися на найбільш несприятливу комбінацію часів спрацювання тригерів.

ЛІТЕРАТУРА

1. *Бойт К.* Цифровая электроника / К. Бойт. – М. : Техносфера, 2007.

2. *Угрюмов Е. П.* Цифровая схемотехника : учеб. пособ. для ВУЗов / Е. П. Угрюмов. 2 изд., перераб. и доп. – СПб. : БХВ-Петербург, 2004.

3. *Погорілий С.Д.* Програмне конструювання: Підручник за редакцією академіка АПН України Третяка О.В. / С.Д. Погорілий ВПЦ Київський університет. 2007.

4. *Потемкин И. С.* Функциональные узлы цифровой автоматики / И. С. Потемкин. – М. : Энергоатомиздат, 1988.

5. *Brown S.* Fundamentals of digital logic with VHDL design / S. Brown, V. Zvonko. 3rd ed. – Toronto : McGraw-Hill, 2009.

6. Harris D. Digital Design and Computer Architecture / D. Harris, S. Harris. 2 ed. – N.Y. : Elsevier, 2013.

ЗМІСТ

Навчальне видання

БАРАБАНОВ Олександр Валерійович
БАУЖА Олександр Стасісович

ОСНОВИ ЦИФРОВОЇ СХЕМОТЕХНІКИ

Навчальний посібник

Оригінал-макет виготовлено ВПЦ "Київський університет"

Формат 60x84$^{1/16}$. Ум. друк. арк. 5,1. Наклад 100. Зам. № 216-7959.
Гарнітура Times New Roman. Папір офсетний. Друк офсетний. Вид. № Рф4.
Підписано до друку 30.01.17

Видавець і виготовлювач
ВПЦ "Київський університет"
01601, Київ, б-р Т. Шевченка, 14, кімн. 43
☎ (044) 239 32 22; (044) 239 31 72; тел./факс (044) 239 31 28
e-mail: vpc_div.chief@univ.kiev.ua
http: vpc.univ.kiev.ua
Свідоцтво суб'єкта видавничої справи ДК № 1103 від 31.10.02