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

ЦИФРОВІ ІНФОРМАЦІЙНІ СИСТЕМИ

КОНСПЕКТ ЛЕКЦІЙ

Навчальний посібник

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакалавра
за освітньою програмою «Електронні компоненти і системи»
спеціальності 171 Електроніка

Укладач: К.С. Клен

Електронне мережне навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2024

Рецензент *Найда С.А.*, д.т.н., проф.
завідувач кафедри акустичних та мультимедійних електронних систем

Відповідальний редактор *Вербицький Є.В.*, д.т.н., проф.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 6 від 28.03.2024 р.)
за поданням Вченої ради факультету електроніки
(протокол № 03/2024 від 18.03.2024 р.)

У навчальному посібнику викладено інформацію щодо основ логічного проєктування цифрових інформаційних систем, аналізу та синтезу скінченних автоматів Мілі та Мура синхронного та асинхронного типів, лічильників та регістрів та постійних запам'ятовувальних пристроїв.

Конспект лекцій містить теоретичні відомості до 16 лекцій та список рекомендованої літератури.

Навчальний посібник призначений для здобувачів ступеня бакалавра за освітньою програмою «Електронні компоненти і системи» спеціальності 171 Електроніка.

Реєстр. № НП 23/24-377. Обсяг 14,0 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Берестейський проспект, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

© К.С. Клен
© КПІ ім. Ігоря Сікорського, 2024

ЗМІСТ

Вступ 5

Лекція №1. Форми представлення інформації. Моделі сигналів та цифрових пристроїв. Логічні елементи. Області застосування ЦІС 7

Лекція №2. Булева алгебра. Закони і тотожності алгебри логіки. Способи задання логічних функцій. Геометрична інтерпретація логічних функцій. Методи мінімізації логічних функцій: карти Карно (Вейча), метод Квайна, Квайна-Мак-Класкі 18

Лекція №3. Мова VHDL. Ідентифікатори в мові VHDL. Об’єкти, операції та оператори мови VHDL. Опис інтерфейсу пристрою 44

Лекція №4. Комутатори. Мультиплексори. Пристрої зсуву. Селектори-мультиплексори. Кодуючі та декодуючі пристрої 73

Лекція №5. Суматори. Компаратори. Арифметично-логічні пристрої. Буферні елементи. Перехідні процеси в логічних схемах 101

Лекція № 6. Тригери. Тригерні схеми. Загальна характеристика тригерних схем. RS-, D-, JK-, T- та TV-тригери 124

Лекція №7. Синхронізація в цифрових схемах. Узгодження вхідних сигналів. Синхронізатори. Практичні проблеми проектування синхронних цифрових схем 147

Лекція №8. Загальна характеристика скінченних автоматів. Автомати Мілі і Мура. Закони функціонування автоматів. Способи опису роботи автоматів. Еквівалентні перетворення автоматів 164

Лекція №9. Основи аналізу цифрових автоматів. Аналіз автоматів з D-тригерами. Особливості аналізу скінченних автоматів з JK-тригерами 178

Лекція №10. Синтез скінченних автоматів 195

Лекція №11. Синтез асинхронних імпульсних автоматів. Особливості синтезу синхронних автоматів. Використання теореми Шенона при синтезі скінченних автоматів на основі JK-тригерів 209

Лекція №12. Колективна поведінка автоматів 218

Лекція №13. Теорія ігор 234

Лекція №14. Типи лічильників та особливості їх роботи. Асинхронні 247

лічильники. Лічильники з довільним модулем рахунку. Синхронні лічильники. Скінченні автомати на основі лічильників
Лекція №15. Загальне поняття про регістри. Регістри пам’яті. Регістри зсуву. Забезпечення обміну інформацією у послідовному форматі. Регістрові лічильники імпульсів (розподілювачі). Лічильники Джонсона. Системи контролю цифрової апаратури 268
Лекція №16. Постійні запам’ятовуючі пристрої. Репрограмовані ПЗП. Основи побудови флеш пам’яті. Оперативні запам’ятовуючі пристрої 290

ВСТУП

Метою вивчення дисципліни «Цифрові інформаційні системи» є отримання студентами ґрунтовних знань з питань теорії та практики побудови цифрових інформаційних систем (ЦІС). Розглядається важливий клас ЦІС – автомати Мілі та Мура синхронного та асинхронного типів, синтез та аналіз їх структур, використання постійних запам’ятовуючих пристроїв (ПЗП), а також програмованих логічних інтегральних схем (ПЛІС) та побудова на їх основі цифрових пристроїв контролю, перетворення та відображення інформації.

Дисципліна сприяє підсиленню у студентів таких компетентностей:

- здатність виконувати аналіз предметної області та нормативної документації, необхідної для проектування та застосування приладів, пристроїв, компонентів та систем електроніки;

- здатність застосовувати відповідні математичні, наукові й технічні методи, сучасні інформаційні технології і комп'ютерне програмне забезпечення, навички роботи з комп'ютерними мережами, базами даних та Інтернет-ресурсами для вирішення інженерних задач в галузі електроніки;

- здатність вирішувати інженерні задачі в галузі електроніки з урахуванням всіх аспектів розробки, проектування, виробництва, експлуатації та модернізації електронних приладів, пристроїв, компонентів та систем;

- здатність визначати та оцінювати характеристики та параметри матеріалів електронної техніки, аналогових та цифрових електронних пристроїв для проектування мікропроцесорних та електронних систем.

Після засвоєння навчальної дисципліни студенти мають продемонструвати такі програмні результати навчання:

- оцінювати характеристики та параметри матеріалів електронної техніки, розуміти основи твердотільної, функціональної, квантової та енергетичної електроніки, електротехніки, аналогової та цифрової схемотехніки, перетворювальної та мікропроцесорної техніки;

- аналізувати складні цифрові та аналогові інформаційно-вимірювальні

системи з розширеною архітектурою комп’ютерних та телекомунікаційних мереж з урахуванням специфікації вибраних технічних засобів електроніки та відповідної технічної документації;

- проектувати складні системи реального часу та засоби збору і обробки інформації, узгоджені з заданими інформаційними та програмними засобами шляхом застосування програмного забезпечення для вбудованих систем на основі мікроконтролерів;

- вміти засвоювати нові знання, прогресивні технології та інновації, знаходити нові нешаблонні рішення і засоби їх здійснення; відповідати вимогам гнучкості в подоланні перешкод та досягненні мети, раціонального використання та нормування часу, дисциплінованості, відповідальності за свої рішення та діяльність.

Конспект лекцій містить інформацію по 16 лекціям. По кожній з лекцій наведено список рекомендованої літератури.

Лекція №1

Форми представлення інформації

Поняття інформації відноситься до основних понять науки про керування і тісно пов'язане з такими поняттями, як «інформаційні ресурси», «інформаційні технології», «інформаційне забезпечення», «інформаційні системи», «інформаційні структури» тощо. Для того, щоб дати найбільш повне визначення інформації, необхідно звернути увагу на те, що в її основі лежить взаємозалежність пари об’єктів – джерела і споживача інформації. Джерелами інформації, насамперед, є природні об’єкти: люди, тварини, рослини, планети та ін. Разом із цим у міру розвитку науки і техніки джерелами інформації стають наукові експерименти, машини, апарати, технологічні процеси. Значний також перелік об’єктів, що є споживачами інформації: люди, тварини, рослини, різноманітні технічні пристрої тощо [1].

Виходячи зі сказаного, дають таке визначення інформації: **інформація** – сукупність відомостей про всілякі об'єкти, явища, процеси. Більш змістовним є твердження: інформація є вибір одного варіанта (або декількох) із багатьох можливих і рівноправних. Слово «рівноправних» означає, що усі варіанти, із яких робиться вибір, мають щось спільне, тобто належать одній множині. У наукових працях розглянуто три основних варіанти вказаного вибору: якщо вибір підказаний (або вказаний), то мова йде про одержання (рецепцію) інформації; якщо вибір зроблений самостійно і випадково, то говорять про виникнення інформації; і, нарешті, якщо вибір однозначно визначений ситуацією або попередніми подіями (тобто вибору, по суті, немає), то про інформацію взагалі говорити не доводиться.

Розглянемо **основні форми подання, види і властивості інформації**, використовуючи для цього структурну схему, яка дається на рис. 1.1.

Основними *формами подання інформації є символьна, текстова і графічна*. Символьна форма ґрунтується на використанні символів – літер,

цифр, знаків і т. д., є найбільш простою і практично застосовується тільки для передачі сигналів про різні події.



Рис.1.1. Форми представлення, види і властивості інформації

Більш складною є текстова форма подання інформації. Тут, як і в попередній формі, використовуються символи – літери, цифри, математичні знаки. Однак інформація закладена не тільки в цих символах, але й у їх сполученні, порядку проходження. Завдяки взаємозв'язку символів і відображенню мови людини текстова інформація надзвичайно зручна і широко використовується в повсякденному житті.

Найбільш місткою, але і найбільш складною є графічна форма подання інформації. Образи природи, фотографії, креслення, схеми, малюнки мають велике значення в нашому житті і містять величезну кількість інформації. І хоча інформація не має ні маси, ні геометричних розмірів, жодних фізичних або хімічних властивостей, проте для її існування обов’язкова наявність якогось матеріального об’єкта, що передає або зберігає інформацію. Таких об’єктів досить багато, і їхня кількість увесь час зростає.

Моделі цифрових пристроїв

Серед усієї множини методів моделювання технічних об’єктів можна виділити особливий напрямок – логічне моделювання, тобто моделювання цифрових об’єктів з використанням ком’пютерів. Особливістю логічного

моделювання (simulation) є те, що в якості моделей об’єктів (компонентів) використовуються безпосередньо їх закони функціонування в різних формах подання [2].

Всі моделі цифрових об’єктів класифікуються за способом та за формою подання. Спосіб подання, як правило, визначає або модельований параметр або тип процедури моделювання. Класифікація моделей цифрових об’єктів за способом подання наведена на рис. 1.2.



Рис. 1.2. Класифікація моделей цифрових об’єктів за способом подання

Форма подання моделі визначає як форму візуального подання моделі, так і формат подання моделі в структурах даних системи моделювання (рис. 1.3).

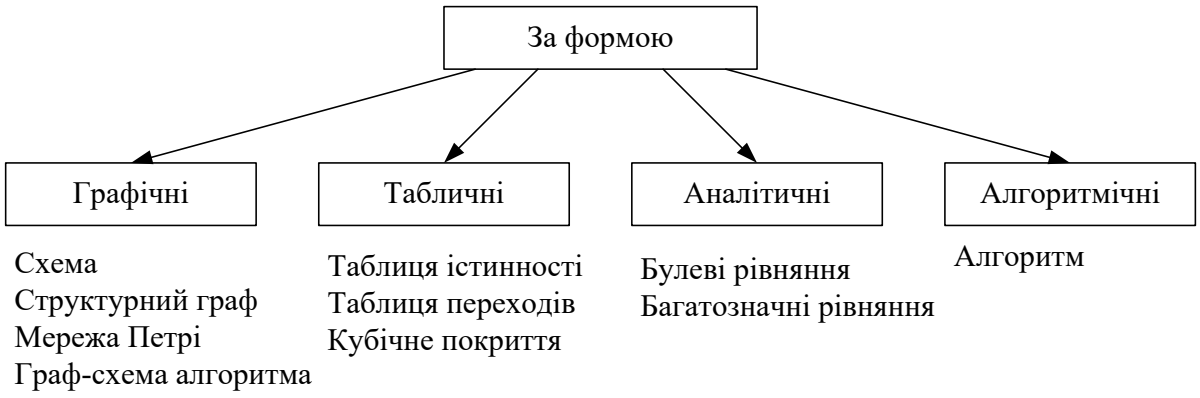
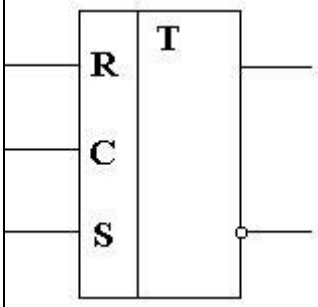
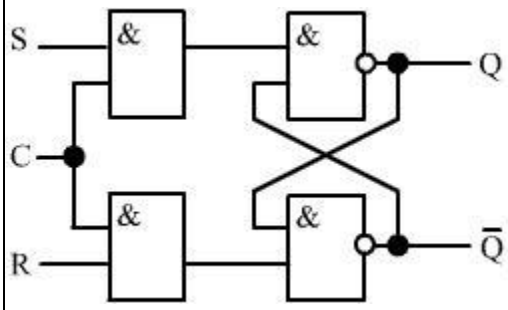
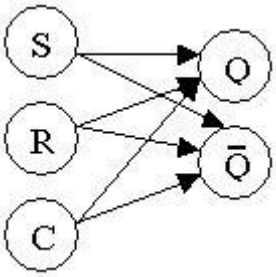
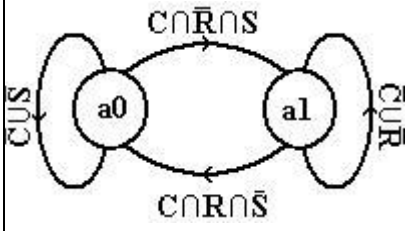


Рис. 1.3. Класифікація моделей цифрових об’єктів за формою подання

Аналітичне, графічне, програмне представлення цифрових пристроїв

Приклад аналітичного, графічного та програмного представлення синхронного RS-тригера наведений у табл. 1.1.

Таблиця 1.1. Приклад аналітичного, графічного та програмного представлення синхронного RS-тригера

Графічне подання	
Схема	Внутрішня структура
	
Структурний граф	Граф переходів
	

Табличне зображення

Повна таблиця переходів

t				t+1
C	R	S	Q	Q
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	x
1	1	1	1	x

Скорочена таблиця переходів

t			t+1
C	R	S	Q
0	x	x	Q(t)
1	0	0	Q(t)
1	0	1	1
1	1	0	0
1	1	1	x

Матриця переходів

Q(t) - Q(t+1)	C	R	S
0 - 0	a	b	a & c
0 - 1	1	0	1
1 - 0	1	1	0
1 - 1	d	d & e	f

Карта Карно

SQ / CR	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	0	x	x
10	0	1	1	1

Аналітичне подання

Рівняння функціонування тригера, отримане шляхом мінімізації карти Карно

$$Q(t+1) = \bar{C}Q \cup \bar{R}Q \cup CS$$

Алгоритмічне подання
ЯОА(VHDL)

architecture JK_tr_arc of JK_tr is
begin
process(clk, reset, set) is
variable q1 :std_logic;
variable p :std_logic_vector(0 to 1);
begin
p :=set & reset;
case p is
when "00"=>q1 :='X';
when "10"=>q1 :='0';
when "01"=>q1 :='1';
when "11"=>q1 := Q;
when others => q1 :='X';
end case;
Q<=q1; Not_Q<=not q1;
end process;
end JK_tr_arch;

Моделі сигналів

У реальних цифрових схемах циркулюють електричні сигнали, але для аналізу цифрових схем використовуються їх логічні еквіваленти. Міжнародний стандарт IEEE_1164 визначив 9 логічних значень сигналів, які використовуються при описі цифрових пристроїв з використанням мов опису апаратури (зокрема VHDL) і при виконанні логічного моделювання в промислових САПР.

У мові VHDL визначено тип std_logic, сигнали та змінні якого можуть приймати значення, визначені в зазначеному стандарті. При описі моделей цифрових пристроїв та логічному моделюванні зазвичай використовуються шість значень:

«U» – (Uninitialized) невизначене значення. Значення, яким ініціалізуються сигнали та змінні типу std_logic за замовчуванням. При нормальній роботі проекту, воно зазвичай не з'являється в моменти часу після першого призначення сигналу;

«X» – (Forcing Unknown) стан, що відповідає невизначеному значенню (0,4 - 2,4 В);

«0» – (Forcing 0) стан, що відповідає значенню логічного нуля (<0,4 В);

«1» – (Forcing 1) стан, що відповідає значенню логічної одиниці (>2,4 В);

«Z» – (High Impedance) стан високого імпедансу.

«-» – (Don't care) байдуже стан (значення). Використовується для завдання маски, оскільки при порівнянні з будь-яким значенням завжди повертає true.

У системах компілятивного логічного моделювання (наприклад, Active-HDL фірми ALDEC) одним з видів представлення результатів моделювання є т.зв. «векторні діаграми» (waveform), що відображають зміни сигналів у часі. Позначення на векторних діаграмах розглянутих раніше станів наведені на рис. 1.4.



Рис. 1.4. Позначення на векторних діаграмах розглянутих станів

Необхідно відзначити, що в простих (облікових) системах моделювання байдужий стан може об'єднуватися з невизначеним, а «високий імпеданс» в

умовах відсутності «дротовий логіки» може асоціюватися з «логічною 1». В результаті мінімальний алфавіт моделювання сигналів у системах логічного моделювання має вигляд {0, 1, X, U}.

Логічні елементи

Практична реалізація аналітичного опису подій алгебри логіки виконується у вигляді логічних схем, що будуються з логічних елементів як логічні **(**цифрові**)** автомати. При цьому проектувальника, як правило, не цікавить внутрішня будова логічних елементів. Вони розглядаються як функціональні вузли обробки цифрової інформації [1].

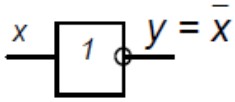


РИСУНОК 1.5. ЕЛЕМЕНТ НІ

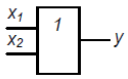
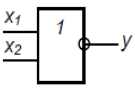
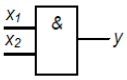
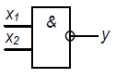
Отже, однією з найпростіших логічних функцій є функція заперечення НІ, яку ще називають операцією інверсії. Графічне позначення елемента, що реалізує таку функцію, наведене на рис. 1.5.

Функцію, що її виконує логічний пристрій, для полегшення сприйняття часто представляють у вигляді таблиці, яку називають таблицею істинності. Кількість стовпців цієї таблиці дорівнює числу змінних, що входять до функції, і є ще один стовпець, у якому вказують значення функції для кожної з можливих комбінацій вхідних змінних, а їхньому числу відповідає кількість рядків таблиці. У загальному випадку кількість рядків дорівнює 2^n, де n – число змінних.

Об’єднана таблиця істинності деяких основних логічних функцій, що залежать від двох змінних (усього таких функцій шістнадцять), наведена в табл. 1.2. На рисунку також представлено інформацію про математичний запис функцій, їхні назви, графічне позначення. З таблиці видно, що, наприклад, функція АБО істинна (дорівнює 1), якщо істинною є хоча б одна із змінних, що

до неї входять, а функція І – тільки у випадку істинності обох змінних.

Таблиця 1.2. Деякі логічні функції двох змінних

Вхідні змінні		Функція у			
x_1	x_2	АБО	АБО-НІ	І	І-НІ
0	0	0	1	0	1
0	1	1	0	0	1
1	0	1	0	0	1
1	1	1	0	1	0
Математичний запис (формула)		$y = x_1 + x_2 = x_1 \vee x_2$	$y = \overline{x_1 + x_2}$	$y = x_1 \cdot x_2 = x_1 \wedge x_2$	$y = \overline{x_1 \cdot x_2}$
Назва функції		Логічне додавання (диз`юнкція) – функція АБО	Заперечення логічного додавання (стрілка Пірса) – функція АБО-НІ	Логічне множення (кон`юнкція) – функція І	Заперечення логічного множення (штрих Шеффера) – функція І-НІ
Графічне позначення елемента, що реалізує функцію					

Якщо вхідні сигнали змінювати з часом, як це показано на рис. 1.6, можна отримати часові діаграми, що відповідають динамічному режиму роботи логічних елементів.

У загальному випадку кількість вхідних змінних (кількість входів)

логічних елементів, необхідних для реалізації складних логічних функцій, може бути будь-якою. Реально у елементів, що випускаються у вигляді ІМС, вона, як правило, становить 2 (чотири елементи в одному корпусі ІМС, що мають спільні кола живлення), 3 (три елементи), 4 (два елементи), 8 (один елемент). Частіше це елементи І-НІ, АБО-НІ.

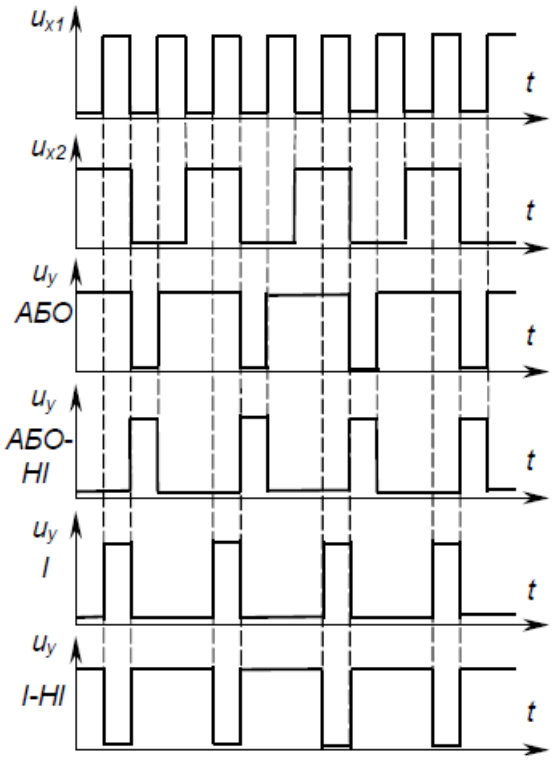


Рисунок 1.6. Часові діаграми роботи деяких двовходових логічних елементів

Області застосування ЦІС

ПЛІС широко використовується для побудови різних за складністю і за можливостями цифрових пристроїв [2]. Це додатки, де необхідна велика кількість портів вводу-виводу (бувають ПЛІС з більш ніж 1000 виводів («пінів»)), цифрова обробка сигналу (ЦОС), цифрова відеоаудіоаппаратура, високошвидкісна передача даних, криптографія, проектування і прототипування ASIC, як мостів (комутаторів) між системами з різною логікою і напругою живлення, реалізація нейрочіпів, моделювання квантових обчислень.

Цифрові схеми:

- Логічні елементи
- Тригери
- Лічильники
- Регістри
- Буферні перетворювачі
- Шифратори
- Дешифратори
- Цифровий компаратор
- Мультиплексори
- Демультиплексори
- Суматори
- полусумматора
- Ключі
- Арифметичні логічні пристрої
- Мікроконтролери
- (Мікро) процесори (у тому числі ЦП для комп'ютерів)
- Однокристальні мікрокомп'ютери
- Мікросхеми та модулі пам'яті
- ПЛІС (програмовані логічні інтегральні схеми)

Перелік посилань

1. Схемотехніка електронних систем: У 3 кн.2. Цифрова схемотехніка: / В.І. Бойко, А.М. Гуржій, В.Я. Жуйков та ін. – К.:Вища шк., 2004. -423 с.

2. Рябенький В.М., Жуйков В.Я., Гулий В.Д. Цифрова схемотехніка: Навч. посібник. – Львів: Новий світ – 2000, 2009, – 736 с.

Лекція №2

Булева алгебра

У практиці інженерної діяльності часто мають місце ситуації, при яких має значення не рівень сигналів, що поступають з відповідних датчиків, а лише наявність чи відсутність таких сигналів. Наприклад, у системах охоронної сигналізації необхідно знати, замкнені чи не замкнені двері або вікна в приміщенні, що охороняється. У системах автоматики часто необхідно знати, чи не перевершує кількість рідини в цистерні заданий рівень, чи не є тиск у котлі нижчим визначеної межі, чи не перевершує температура в приміщенні задану величину і т. п. Схеми, що дають можливість розв'язувати поставлені задачі, можуть описуватись виразами типу: "лампочка на пульті охоронної сигналізації горить, якщо всі вікна замкнені (точніше, замкнено перше **і** друге **і** третє **і**… вікно)". Або "лампочка не горить, якщо хоча б одне вікно відкрите (тобто може бути відкритим перше **або** друге **або** третє **або** перше **і** друге **або**...)". Такі вирази називаються *логічними*.

При проектуванні подібних систем задаються відповідним рівнем напруги живлення, і наявність чи відсутність її дає можливість одержувати відповіді на поставлені питання. Оскільки рівень напруги може бути різним і задаватись прийнятою елементною базою, то з метою формалізації опису подібних схем приймаються деякі умови. Як приклад, високий рівень напруги приймається за "**1**", низький – відповідно, за "**0**". У такому разі приведені вище вирази можуть бути формалізовані: якщо контакти, що фіксують положення вікон, позначити як аргументи x_1, x_2, …, x_n, які можуть приймати лише значення "**1**" або "**0**", то напругу на лампочці можемо розглядати як функцію y, яка теж приймає одне з двох аналогічних значень [1].

Математичний апарат, що оперує з аргументами та функціями, які набувають тільки двох значень – "**0**" та "**1**" – називається *двійковою* (*булевою*) *алгеброю* або *алгеброю логіки*. Такий математичний апарат для розв'язання задач формальної логіки розробив ірландський математик Дж. Буль.

Логічні змінні (аргументи), як і змінні звичайної алгебри, позначаються літерами латинського алфавіту з різними індексами – наприклад, x_0, x_1, x_2, x_3. Індекс при змінній може одночасно означати розряд двійкового числа.

Якщо змінна x_i набуває значення $x_i = 1$, то таке її значення називають *істинним*. Протилежне $x_i = 0$ називають *хибним* і умовно позначають $\overline{x_i}$, що означає заперечення істинного значення аргументу (в зарубіжній практиці операція заперечення позначається апострофом x'). Два елементи булевої алгебри – подія істинна і подія хибна – називають її *константами*.

Булева функція позначається літерою y і є двійковою функцією двійкових аргументів. Умовне її позначення

$$y = f(x_1, x_2, ..., x_n).$$

Булева функція, яка залежить від n аргументів, називається n-вимірною і є повністю визначеною, якщо вказані значення її для всіх двійкових наборів значень її аргументів. Кількість таких наборів дорівнює 2^n. *Тобто, областю визначеності функції* ***n*** *змінних є сукупність дискретних точок* ***n****-вимірного простору, причому кожна з точок є комбінацією значень цих змінних* (*кодовою комбінацією*). Оскільки можливі 2^n різних комбінацій логічних змінних, то область визначення функції складається зі скінченої величини – 2^n точок. Це, в свою чергу, означає, що кожна функція може бути задана таблицею значень, які вона приймає в точках її області визначеності.

Функція повністю визначена, якщо задані її значення в усіх точках області визначеності. Значення функції вибираються з множини "**0**" і "**1**". Якщо ж значення функції не задано в одній або кількох точках, то вона є *неповністю визначеною*. Кодові комбінації, при яких функція невизначена, називаються *факультативними*. У практиці цифрової схемотехніки існує велика кількість неповністю визначених функцій. Довизначення їх, якщо це необхідно, забезпечується встановленням їх значень – "**0**" або "**1**" –довільним шляхом.

Усі можливі логічні функції *n* змінних можна створити за допомогою трьох основних операцій:

а) логічне заперечення (інверсія, операція **НІ**); позначається рискою над відповідною функцією або аргументом – такий зв'язок між аргументом *x* та функцією *y*, при якому *y* істинна тоді і тільки тоді, коли значення *x* хибне, і навпаки;

б) логічне додавання (диз'юнкція, операція **АБО**), яке позначається символами (V), (+) – така функція, яка істинна тоді і тільки тоді, коли одночасно істинні всі логічні змінні;

в) логічне множення (кон'юнкція, операція **І**), яке позначається символами (∧), (·), (&) – така функція, яка хибна тоді і тільки тоді, коли одночасно хибні всі додавані змінні. Для позначення еквівалентності логічних виразів використовується знак (=).

Слід пам'ятати, що операція кон'юнкції є старшою операцією і виконується раніше диз'юнкції.

Приклад 2.1. Записати вираз функції трьох змінних, яка приймає істинні значення при умові, що будь-яка пара змінних одночасно має істинні значення.

Розв'язання. Введемо умовні позначення змінних x_0, x_1, x_2.

У загальному плані функція матиме вигляд:

$$y = f(x_2, x_1, x_0).$$

Оскільки істинні значення функції визначаються будь-якою парою логічних змінних, тобто або x_0 і x_1, або x_0 і x_2, або x_1 і x_2, то аналітична форма запису функції прийме вигляд:

$$y_1 = x_0 \cdot x_1 + x_0 \cdot x_2 + x_1 \cdot x_2.$$

Функція може мати і дещо іншу форму запису, якщо враховувати, що при виконанні будь-якої з умов, що закладені в функцію, обмежень на значення третьої змінної не накладається.

Приклад 2.2. Формалізувати та записати у вигляді булевих функцій висловлення: лампочка охоронної сигналізації світиться, коли всі три двері приміщення зачинені.

Розв'язання. Позначимо логічні змінні x_1, x_2, x_3 істинними, якщо відповідні двері зачинені. В такому випадку істинне значення функції ("лампочка сигналізації світиться") визначається за формулою:

$$y_2 = x_1 \cdot x_2 \cdot x_3 .$$

Приклад 2.3. Формалізувати та записати у вигляді булевих функцій висловлення: температурна сигналізація вмикається, коли хоча б один з двох датчиків зафіксує температуру 70°.

Розв'язання. Приймемо за істинне значення логічних змінних x_1, x_2 показ датчика, що відповідає температурі 70°.

У такому випадку логічна функція має вигляд:

$$y_3 = x_1 + x_2 + x_1 \cdot x_2 .$$

Приклад 2.4. Змагання зі штанги судять три судді: головний, що знаходиться проти помосту, і два бокові. Якщо суддя вважає, що вага взята, він натискає кнопку, яка знаходиться на його столі. Для спортсмена вага вважається взятою, якщо загорається лампочка біля помосту. Умова загорання лампочки наступна: вона загорається, якщо головний і, як мінімум, один з бокових суддів натиснули кнопки на своїх столах. Формалізувати умову загорання лампочки.

Розв'язання. Приймемо за логічні змінні кнопки, що знаходяться на столах суддів: x_1 – кнопка головного судді; x_2 та x_3 – бокових. Приймемо за істинне значення натиснуту кнопку $x_i = 1$. Тоді умова загорання лампочки формально може бути записана в вигляді:

$$y = x_1\, x_2\, x_3 + x_1\, x_2\, \overline{x_3} + x_1\, \overline{x_2}\, x_3 .$$

Технічна реалізація булевих функцій, а, відповідно, і їх фізична інтерпретація добре ілюструється за допомогою контактних схем, в яких логічна змінна x_i відповідає замкненому контакту. Схеми, що ілюструють

реалізацію операцій кон’юнкції та диз’юнкції, наведені відповідно на рис. 2.1, ***а***, ***б***.

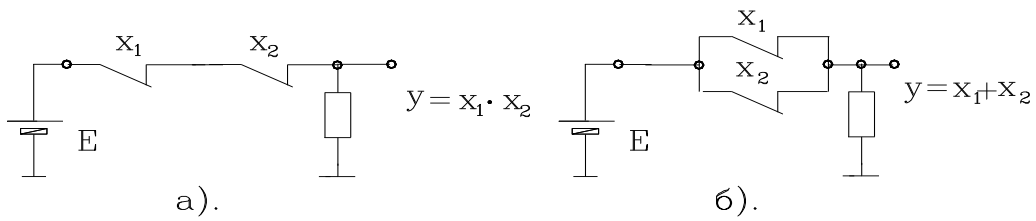


Рис. 2.1. Схеми, що ілюструють реалізацію операцій кон’юнкції та диз’юнкції

Закони і тотожності алгебри логіки

В алгебрі логіки використовується ряд аксіом (тотожностей) та законів. Основними з них є наступні: *переміщувальний* (*властивість комутативності*); *сполучний* (*властивість асоціативності*); *розподільний* (*властивість дистрибутивності*); *інверсії* (*теорема де Моргана*). Головні аксіоми та закони булевої алгебри наведені у табл. 2.1 [1].

Таблиця 2.1. Головні аксіоми та закони булевої алгебри

Назва аксіоми чи закону	Вирази
Аксіоми (тотожності)	$0 \cdot x = 0$ $1 + x = 1$ $0 + x = x$ $x \cdot x = x$ $x + x = x$ $x \cdot \bar{x} = 0$ $x + \bar{x} = 1$ $\bar{\bar{x}} = x$
Закони комутативності	$x_1 + x_2 = x_2 + x_1$ $x_1 \cdot x_2 = x_2 \cdot x_1$

Закони асоціативності	$x_1 + x_2 + x_3 = x_1 + (x_2 + x_3) = (x_1 + x_2) + x_3 =$ $= (x_1 + x_3) + x_2$ $x_1 \cdot x_2 \cdot x_3 = x_1 \cdot (x_2 \cdot x_3) =$ $= x_2 \cdot (x_1 \cdot x_3) = x_3 \cdot (x_1 \cdot x_2)$
Закони дистрибутивності	$x_1 \cdot (x_2 + x_3) = x_1 \cdot x_2 + x_1 \cdot x_3$ $x_1 + x_2 \cdot x_3 = (x_1 + x_2) \cdot (x_1 + x_3)$
Закони інверсії (теорема де Моргана, принцип подвійності)	$\overline{x_1 \cdot x_2} = \overline{x_1} + \overline{x_2}$ $\overline{x_1 + x_2} = \overline{x_1} \cdot \overline{x_2}$
Закони поглинання	$x_1 + x_1 \cdot x_2 = x_1$ $x_1 \cdot (x_1 + x_2) = x_1$

Закони інверсії, які відображають властивість взаємного перетворення операцій логічного множення і додавання в алгебрі логіки, називають *принципом подвійності*.

Способи задання логічних функцій

Існують такі способи задання або запису логічних функцій – *аналітичний*, *табличний*, *за допомогою карт Карно*, *графічний* та *кубічний* [1].

Аналітично логічна функція може бути записана різними комбінаціями кон'юнкцій та диз'юнкцій логічних змінних. Зазвичай логічні функції записуються або у вигляді суми добутків логічних змінних (диз'юнкція кон'юнкцій) або у вигляді логічного добутку їх сум (кон'юнкція диз'юнкцій). Наведення функції у вигляді диз'юнкції кон'юнкцій називають *диз'юнктивною нормальною формою* (**ДНФ**):

$$y = x_1\overline{x_2} + \overline{x_1}x_3 + x_1x_2\overline{x_3}\ ,$$

а запис у вигляді кон'юнкції диз'юнкцій – відповідно, *кон'юнктивною нормальною формою* (**КНФ**):

$$y = (x_1 + x_2)(x_2 + \overline{x_3})(\overline{x_1} + x_2 + x_3)\,.$$

Інверсія у відповідності з теоремою де Моргана будь-якої функції, приведеній в одній формі, призводить до заміни запису на іншу форму.

Будь-яка логічна функція, задана в аналітичній формі, може бути перетворена на **ДНФ** або **КНФ** за допомогою тотожностей та законів алгебри логіки. При цьому для однієї і тієї ж функції може існувати декілька рівнозначних диз'юнктивних та кон'юнктивних нормальних форм.

У той же час, існує лише один вид **ДНФ** та **КНФ**, в яких функція може бути записана єдиним чином. Такі форми називаються *досконалими диз'юнктивними* (*кон'юнктивними*) *нормальними формами* (**ДДНФ**, **ДКНФ**). Вони характеризуються тим, що в **ДДНФ** кожна кон'юнкція, а в **ДКНФ** кожна диз'юнкція містять усі логічні змінні даної функції, з інверсіями або без них.

Оскільки кожна кон'юнкція функції, що наведена у **ДДНФ**, визначає її істинне значення, відповідаюче "**1**", то такі кон'юнкції називаються *конституєнтами одиниці* (*мінтермами*). Аналогічно, диз'юнкції функції, що наведені у **ДКНФ**, називаються *конституєнтами нуля* (*макстермами*).

Якщо замінити логічні змінні та їх заперечення одиницями та нулями, то кожна кон'юнкція буде представляти собою двійкове число.

Приклад 2.5. Функцію з прикладу 2.4 зобразити в скороченій диз'юнктивній канонічній формі.

Розв'язання. Перепишемо функцію, дещо змінивши нумерацію і порядок розміщення змінних:

$$y = x_2 \cdot x_1 \cdot x_0 + x_2 \cdot x_1 \cdot \overline{x_0} + x_2 \cdot \overline{x_1} \cdot x_0 .$$

Скорочена диз'юнктивна канонічна форма приведеної функції матиме вигляд (у порядку розміщення кон'юнкцій):

$$y = \bigvee_{0}^{7} 5, 6, 7 .$$

Аналогічно, функцію можна зобразити і у вигляді добутку макстермів. Така форма запису називається *канонічним добутком*. Наприклад:

$$y = \bigwedge_{0}^{7} 2, 4 = (x_2 + \overline{x_1} + x_0)(\overline{x_2} + x_1 + x_0) .$$

Легко бачити можливість конвертації в представленні функції у вигляді макстермів та мінтермів, оскільки кожна з них доповнює функцію до повного перебору логічних змінних. Як приклади, можемо записати:

$$y = \bigvee_{0}^{7} 2, 6, 7 = \bigwedge_{0}^{7} 0, 1, 3, 4, 5;$$

$$y = \bigvee_{0}^{7} 0, 1, 3, 5 = \bigwedge_{0}^{7} 2, 4, 6, 7;$$

$$y = \bigvee_{0}^{15} 0, 4, 5, 9, 11, 13, 15 = \bigwedge_{0}^{15} 1, 2, 3, 6, 7, 8, 10, 12, 14.$$

Індекси біля умовних позначень операцій диз'юнкції та кон'юнкції вказують на діапазон можливих мінтермів та макстермів логічних функцій. Нижній індекс іноді не вказується.

Досконала диз'юнктивна нормальна форма запису дозволяє легко перейти до інших форм запису – *табличної* та *карт Карно*.

Таблиця 2.2. Функції двох змінних

x_1	x_0	y_1	y_2	y_3	y_4	y_5
0	0	0	0	0	0	1
0	1	1	1	1	0	0
1	0	1	1	1	0	0
1	1	x	1	0	1	1

У табл. 2.2 наведені функції $y_1 \dots y_5$ двох змінних x_0 та x_1. Табличний спосіб полягає у тому, що функція задається у вигляді таблиці відповідності (таблиці істинності станів). У таблицю вписують усі можливі комбінації аргументів у порядку зростання їх індексів і при кожній комбінації встановлюється значення функції. Кількість всіх можливих сполук аргументів, а, отже, і кількість значень функції дорівнює 2^n, де n – кількість логічних змінних. З табличної форми запису легко перейти до аналітичної, використовуючи досконалу диз'юнктивну форму запису логічних функцій. Для цього функція записується як диз'юнкція конституєнт одиниці.

Приклад 2.6. Зобразити функцію з прикладу 2.4 у табличній формі запису.

Розв'язання. Функція трьох змінних у табличній формі (табл. 2.3) складатиметься з чотирьох стовбців, у трьох з яких буде розміщений повний перебір логічних змінних.

Таблиця 2.3. Функція трьох змінних

N	x_2	x_1	x_0	y
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

Логічна функція іноді може бути неповністю визначеною. У табл. 2.2 приводиться форма запису функції y_1, яка є невизначеною при $x_1 = x_0 = 1$. При переході до аналітичної форми запису вона повинна бути довизначена.

Функції двох змінних займають у булевій алгебрі особливе місце. Для двох змінних кількість булевих функцій дорівнює 16. Ці функції називаються *елементарними* і складають максимальний набір функцій двох змінних. У табл. 2.4 наведені всі елементарні функції двох змінних.

За своєю застосованістю вони різні. Найбільш широке використання знаходять функції **І**, **АБО**, **І-НІ**, **АБО-НІ**, **ВИКЛ. АБО**. Вони універсальні тому, що за їх допомогою можна записати будь-яку іншу функцію.

Розроблений математичний апарат аналізу та синтезу булевих функцій найбільш відповідає цим функціям. Набір логічних функцій **І**, **АБО**, **НІ** дозволяє реалізувати будь-яку іншу функцію і називається *функціонально повним*.

Таблиця 2.4. Елементарні функції двох змінних

N	x_0	0	1	0	1	Назва функції	Позначення
	x_1	0	0	1	1		
0		0	0	0	0	Константа нуль	0
1		0	0	0	1	Кон’юнкція, **І**	$x_1 \cdot x_0$
2		0	0	1	0	Заборона по x_0	$x_1 \cdot \overline{x_0}$
3		0	0	1	1	Змінна x_1	x_1
4		0	1	0	0	Заборона по x_1	$\overline{x_1} \cdot x_0$
5		0	1	0	1	Змінна x_0	x_0
6		0	1	1	0	**Викл. АБО**, сума за mod 2	$x_1 \oplus x_0$
7		0	1	1	1	Диз’юнкція, **АБО**	$x_1 + x_0$
8		1	0	0	0	**АБО-НІ**, функція Пірса	$\overline{x_1 + x_0}$
9		1	0	0	1	Рівнозначність, еквівалентність	$x_1 \equiv x_0$
10		1	0	1	0	Заперечення x_0	$\overline{x_0}$
11		1	0	1	1	Імплікація по x_0	$x_1 + \overline{x_0}\ \ x_0 \to x_1$
12		1	1	0	0	Заперечення x_1	$\overline{x_1}$
13		1	1	0	1	Імплікація по x_1	$x_1 \to x_0\ \ \overline{x_1} + x_0$
14		1	1	1	0	Функція Шеффера **І-НІ**	$\overline{x_1\ x_0}$
15		1	1	1	1	Константа 1	1

У практичній схемотехніці найбільш поширеними є системи, які реалізують логічні функції **І-НІ**, **АБО-НІ**, **ВИКЛ. АБО**. Вони дозволяють найбільш просто реалізовувати різні функції, мати більшу кількість входів, прості в технічній реалізації.

Теорема Шенона

Широке використання при перетворенні логічних функцій знаходять теорема Шенона та ряд тотожностей, які витікають з неї [1].

Теорема Шенона формулюється так: будь-яку функцію *n* зміних можна зобразити в формі:

$$f(x_{n-1}, \ldots, x_i, \ldots, x_0) = \overline{x_i} \cdot f(x_{n-1}, \ldots, 0, \ldots, x_0) + x_i \cdot f(x_{n-1}, \ldots, 1, \ldots, x_0). \quad (2.1)$$

Теорема Шенона виявляється дуже корисною при виконанні перетворень логічних виразів, що містять операцію **ВИКЛ. АБО**.

Приклад 2.7. Виконати перетворення логічної функції:

$$y = x_2 x_1 \oplus (x_3 + \overline{x_1}) \oplus x_3 x_1.$$

Розв'язання. Використовуючи теорему Шенона, виконаємо наступний ряд перетворень:

$$x_2 x_1 \oplus (x_3 + \overline{x_1}) \oplus x_3 x_1 = \overline{x_1}\,[x_2 \cdot 0 \oplus (x_3 + \overline{0}) \oplus x_3 \cdot 0] +$$
$$+ x_1\,[x_2 \cdot 1 \oplus (x_3 + \bar{1}) \oplus x_3 \cdot 1] = \overline{x_1}\,[0 \oplus 1 \oplus 0] + x_1\,[x_2 \oplus x_3 \oplus x_3] = \overline{x_1} + x_1 \cdot \overline{x_2} = \overline{x_1} + \overline{x_2}.$$

З теоремою Шенона (1.17) пов'язані тотожності:

$$\begin{aligned} &\overline{x_i} \cdot f(x_{n-1}, \ldots, x_i, \ldots, x_0) = \overline{x_i} \cdot f(x_{n-1}, \ldots, 0, \ldots, x_0); \\ &x_i \cdot f(x_{n-1}, \ldots, x_i, \ldots, x_0) = x_i \cdot f(x_{n-1}, \ldots, 1, \ldots, x_0). \end{aligned} \quad (2.2)$$

Виходячи з теореми де Моргана, тотожностям (2.2) відповідають наступні тотожності:

$$\begin{aligned} &\overline{x_i} + f(x_{n-1}, \ldots, x_i, \ldots, x_0) = \overline{x_i} + f(x_{n-1}, \ldots, 1, \ldots, x_0); \\ &x_i + f(x_{n-1}, \ldots, x_i, \ldots, x_0) = x_i + f(x_{n-1}, \ldots, 0, \ldots, x_0). \end{aligned} \quad (2.3)$$

Тотожності (2.2) і (2.3) широко використовуються для спрощення логічних виразів. З них витікають формули, які широко використовуються:

$$x + x \cdot y = x + 0 \cdot y = x;$$
$$x + \bar{x} \cdot y = x + \bar{0} \cdot y = x + y.$$

Приведені тотожності дають можливість суттєво спрощувати складні функції багатьох змінних, особливо при наявності заперечень.

Приклад 2.8. Спростити логічну функцію:

$$f(x_1, x_2, x_3) = \overline{\overline{x_2 \cdot \overline{x_1} \oplus x_3 \cdot \overline{x_2}} \oplus x_1 + x_3 \cdot \overline{x_2} \cdot \overline{x_1}}.$$

Використовуючи тотожність (2.2) відносно $\overline{x_2}$, маємо:

$$\overline{\overline{0 \cdot \overline{x_1} \oplus x_3 \cdot \overline{0}} \oplus x_1 + x_3 \cdot \overline{0} \cdot \overline{x_1}} = \overline{\overline{x_3} \oplus x_1 + x_3 \cdot \overline{x_2}}.$$

З тотожності (2.3) знаходимо:

$$\overline{x_3} \oplus x_1 + x_3 = x_3 + \overline{0} \oplus x_1 = x_3 + \overline{x_1}.$$

В результаті знаходимо:

$$\overline{\overline{x_3 + \overline{x_1}} \cdot \overline{x_2}} = \overline{x_3} \cdot \overline{x_2} \cdot x_1.$$

Приведені вище тотожності використовуються для того, щоб розкладати складні логічні функції на більш прості.

Розкладання Ріда

Широке використання знаходить також інший тип розкладання функцій – розкладання Ріда [1]. Воно базується на використанні наступної умови: якщо $x \cdot y = 0$, то $x + y = x \oplus y$.

Дійсно,

$$x \oplus y = x \cdot \overline{y} + \overline{x} \cdot y + x \cdot y + x \cdot y = x\,(y + \overline{y}) + y\,(x + \overline{x}) = x + y.$$

Це дає можливість розкладання Шенона

$$f(x_{n-1}, \ldots, x_i, \ldots, x_0) = \overline{x_i} \cdot f_0 + x_i \cdot f_1,$$

де $f_0 = f(x_{n-1}, \ldots, 0, \ldots, x_0)$; $f_1 = f(x_{n-1}, \ldots, 1, \ldots, x_0)$,

зобразити у вигляді:

$$f(x_{n-1}, \ldots, x_i, \ldots, x_0) = \overline{x_i} \cdot f_0 + x_i \cdot f_1 = \overline{x_i} \cdot f_0 \oplus x_i \cdot f_1 =$$
$$(1 \oplus x_i)\, f_0 \oplus x_i \cdot f_1 = f_0 \oplus f_0 \cdot x_i \oplus x_i \cdot f_1 = f_0 \oplus (f_0 \oplus f_1)\, x_i.$$

Отриманий вираз називається *розкладанням Ріда*. Воно дає можливість будь-яку функцію *n* змінних зобразити у вигляді полінома *n*-го ступеня виду:

$$f(x_{n-1}, \ldots, x_i, \ldots, x_0) = a_0 \oplus \sum_{i=0}^{n-1} b_1\ x_i \oplus \sum_{i=0}^{n-1} b_2\ x_i\ x_k \oplus \ldots$$

де a_0, b_1, b_2, … – коефіцієнти ряду Ріда.

Так, наприклад, функція двох змінних може бути записана у вигляді:

$$f(x_1, x_0) = a_0 \oplus b_0\ x_0 \oplus b_1\ x_1 \oplus b_2\ x_1\ x_0 .$$

Якщо функція описується поліномами першого ступеня, вона називається *лінійною*. Для таких функцій коефіцієнт при логічних добутках змінних дорівнює нулю. У такому випадку для функції двох змінних можемо записати:

$$f(x_1, x_0) = a_0 \oplus b_0\ x_0 \oplus b_1\ x_1,$$

а для функції з кількістю змінних n

$$f(x_1, x_0) = a_0 \oplus b_0\ x_0 \oplus b_1\ x_1 \oplus b_2\ x_2 .... \oplus b_{n-1}\ x_{n-1} .$$

Приклад 2.9. Зобразити функцію

$$f(x_1, x_0) = \overline{x_1}\ x_0 + x_1\ \overline{x_0}$$

у вигляді лінійного полінома.

Розв'язання. Виконуючи перетворення, знаходимо:

$$f(0, x_1) = x_1 ;$$
$$f(1, x_1) = \overline{x_1} ;$$
$$f(x_1, x_0) = x_1 \oplus [\overline{x_1} \oplus x_1] \cdot x_0 = x_1 \oplus x_0 .$$

У цифрових пристроях, які реалізують ті чи інші логічні функції, сигнали здебільшого розділяються за функціональним призначенням – наприклад, інформаційні, адресні, керуючі – і, відповідно, мають свої умовні позначення. Це дає можливість за виглядом логічних функцій чітко бачити особливості взаємодії між сигналами, взаємозалежність між ними. Деякі приклади таких функцій:

$$f(a_0, x_1, x_2) = \overline{a_0}\ x_1 + a_0\ x_2 ;$$
$$f(a_1, a_0, x_3, x_2, x_1, x_0) = x_0\ a_0\ a_1 + x_1\ \overline{a_0}\ a_1 + x_2\ \overline{a_0}\ \overline{a_1} + x_3\ a_0\ \overline{a_1} .$$

Приведені функції описують роботу простих мультиплексорів. Коефіцієнти a_0 і a_1 є також логічними змінними, які мають інше функціональне призначення.

У той же час, остання функція може бути представлена у вигляді:

$$f(x_3, x_2, x_1, x_0) = b_1\ x_0 + b_2\ x_1 + b_3\ x_2 + b_4\ x_3,$$

де $b_1 \ldots b_4$ – коефіцієнти, функціонально залежні від a_0 і a_1 .

У якості логічних сигналів можуть використовуватись функції. У такому випадку описана логічна функція реалізує функціональний комутатор.

Геометрична інтерпретація логічних функцій

Однією з форм геометричної інтерпретації логічних функцій є використання карт Карно, представлених у **ДДНФ** (**ДКНФ**).

Карта Карно – це компактна форма представлення таблиці істинності логічної функції із застосуванням для позначення (кодування) комбінацій змінних циклічного коду Грея.

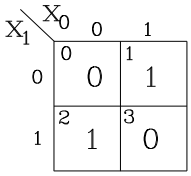


Рис. 2.2. Карта для двох логічних змінних

Карта для двох логічних змінних наведена на рис. 2.2. Особливість карти Карно полягає в тому, що по горизонталі та по вертикалі задаються координати клітинок, якими виступають аргументи логічної функції. Тому кожна клітина має свою координату – 00, 01, 10, 11 – яка може бути представлена відповідною двійковою або десятковою цифрою. Значення функції записуються в відповідних клітинах нулями та одиницями. Наприклад, логічна функція $y = \bigvee_{0}^{3} 1, 2$ зображена двома одиницями і двома нулями у відповідних клітинах карти Карно (рис. 2.2).

При використанні карт Карно для функцій з більшою кількістю логічних змінних кількість клітин у карті зростає відповідно до формули

$$N = 2^n.$$

Приклад 2.10. Функцію, що приведена в скороченій канонічній формі:

$$y = \bigvee_{0}^{15} 2, 3, 4, 5, 10, 11, 12, 13,$$

зобразити у вигляді карти Карно.

Розв'язання. Оскільки числові величини мінтермів перевищують число 7 і знаходяться в межах 15, то приведена логічна функція має чотири змінні, які представимо як x_3, x_2, x_1, x_0. Карта Карно для чотирьох змінних має вигляд, приведений на рис. 2.3.

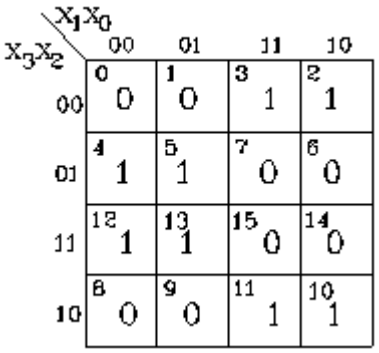


Рис. 2.3. Карта Карно для чотирьох логічних змінних

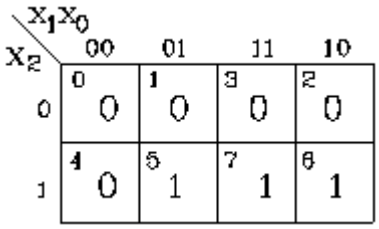


Рис. 2.4. Карта Карно для трьох логічних змінних

З карти Карно також легко перейти до табличної форми запису і навпаки.

Якщо задана функція неповністю визначена, то, подібно до таблиці, у відповідних клітинах записується знак невизначеності, що дає можливість повністю відобразити всі властивості функції.

Ще однією формою геометричної інтерпретації логічних функцій, яка досить широко використовується останнім часом, є *n-вимірні куби*, або *кубічні комплекси*.

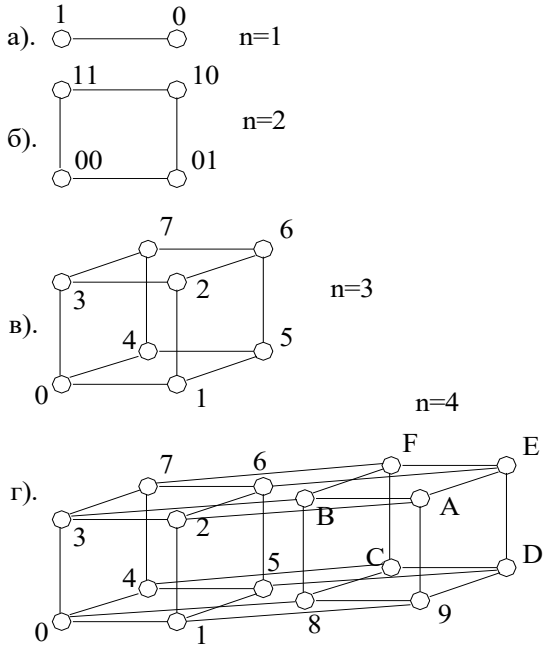


Рис. 2.5. Геометричні зображення *n*-вимірного куба для різних значень *n*

На рис. 2.5 приведені геометричні зображення *n*-вимірного куба для різних значень *n*, що відповідають словам двійкового коду. Кожна вершина відображає одне кодове слово. Наприклад, функція $y = x$ відображається на рис. 2.5, ***а*** вершиною з позначенням **1**.

Функція $y = x_1\, x_2 + \overline{x_1}\ \overline{x_2}$ на рис. 2.5, ***б*** відображається двома вершинами, з'єднаними прямою лінією (ребром кубу).

Кожна вершина куба, в якій функція має мінтерм, називається *нульовим кубом* (0-кубом). Функція, яка містить декілька мінтермів. зображається в просторі *нульовим кубічним комплексом* K_0.

Якщо два нульових куби комплексу K_0 відрізняються тільки по одній змінній, то два сусідніх мінтерми створюють *одиничний куб* (1-куб), який зображається геометрично як ребро *n*-вимірного куба. Множина одиничних кубів створюють одиничний кубічний комплекс K_1.

Аналогічно, якщо два одиночних куба комплексу K_1 відрізняються тільки по одній координаті, то вони створюють *двійковий куб* (2-куб). Геометрично таке зображення є гранню *n*-вимірного куба. Множина двійкових кубів створюють кубічний комплекс K_2.

Об’єднання кубічних комплексів K_0, K_1, K_2, …, K_m логічної функції y (x_0 … x_n) створюють її кубічний комплекс:

$$K(y) = \vee(K_0, K_1, \ldots, K_m).$$

Приклад 2.11. Для логічної функції

$$y = \bigvee_{0}^{7} 3, 4, 5, 6, 7$$

записати кубічні комплекси.

Розв’язання. Нульовий кубічний комплекс має п’ять членів за кількістю мінтермів логічної функції:

K_0 = (011, 100, 101, 110, 111).

Порівнюючи записані 0-куби, знаходимо, що сусідніми є мінтерми 3 і 7, які відрізняються лише старшим розрядом. Тому вони створюють 1-куб виду -11. Аналогічно, мінтерми 4 і 5 створюють 1-куб виду 10, а мінтерми 4 і 6 створюють 1-куб виду 1-0.

Повний аналіз показує, що одиничний кубічний комплекс матиме вигляд:

K_1 = (-11, 10-, 1-0, 11-, 1-1).

Двійковий кубічний комплекс містить один 2-куб:

K_2 = (1- -).

Графічна інтерпретація ґрунтується на використанні умовних позначень логічних елементів, які є стандартними.

Ще однією формою представлення логічних функцій є *часові діаграми*, які відображають часові співвідношення між вхідними логічними змінними і вихідною функцією (рис. 2.6). Часові діаграми зображаються з урахуванням фронтів імпульсних послідовностей з відображенням моментів початку перехідних процесів і часто мають допоміжні вказівні стрілки, які уточнюють хід перехідних процесів у схемі.

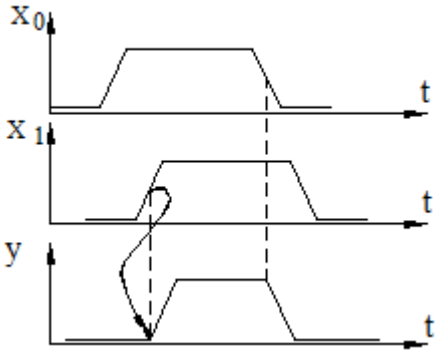


Рис. 2.6. Приклад часових діаграм для логічної схеми, що реалізує операцію цифрового компаратора

На рис. 2.6 приводиться приклад часових діаграм для логічної схеми, що реалізує операцію цифрового компаратора. У випадку, що розглядається, приводяться лише вхідні та вихідні сигнали. При визначенні інтервалів часових затримок, тривалості перехідних процесів приводиться більша деталізація часових діаграм.

Методи мінімізації логічних функцій: карти Карно (Вейча), метод Квайна, Квайна-Мак-Класкі

Метою мінімізації є зменшення вартості технічної реалізації логічних функцій незалежно від використовуваних апаратних засобів [1].

Логічні функції апаратно реалізуються за допомогою мікросхем, орієнтованих на виконання тих чи інших операцій. Мікросхеми загального використання здебільшого можуть реалізовувати декілька простих одиночних операцій. З цієї причини справедливо стверджувати, що чим простішою є аналітична форма запису логічної функції, тим менше використовується логічних елементів і, як результат, тим менше мікросхем необхідно для її реалізації. Складність логічних функцій визначається кількістю логічних змінних, що входять до їх складу в прямому і інверсному виді, та кількістю простих логічних операцій над ними. Будь-яка логічна функція може бути записана різними аналітичними виразами різного рівня складності. Серед них можна знайти такі, які містять мінімальну кількість логічних змінних і операцій

над ними. Задача знаходження таких аналітичних виразів називається *мінімізацією логічних функцій*. Звідси витікає, що *мінімізація логічної функції* – це заміна логічної функції, що представлена у вигляді логічної суми мінтермів або логічного добутку макстермів, іншою логічною функцією з мінімальною кількістю логічних змінних та операцій над ними.

Задача мінімізації – це задача неоднозначна, і різними шляхами можна отримати різні вирази мінімізованої функції, які відрізнятимуться між собою кількістю змінних і операцій над ними.

Аналітичний спосіб мінімізації. Для зменшення складності логічних функцій здебільшого використовуються операції склеювання:

$$x_1\, x_2 + x_1\, \overline{x_2} = x_1\, (x_2 + \overline{x_2}) = x_1\,;$$
$$(x_1 + x_2)\, (x_1 + \overline{x_2}) = x_1$$

та поглинання:

$$x_1 + x_1 x_2 = x_1(1 + x_2) = x_1\,;$$
$$x_1\, (x_1 + x_2) = x_1\,.$$

Мінімізація за допомогою карт Карно (діаграм Вейча). Метод, оснований на використанні карт Карно, характеризується своєю простотою і наочністю. Зображення функції в площині координат її параметрів, подібно до алгебраїчних функцій, дозволяє наглядно встановити взаємозв'язки між її змінними і, як результат, виділити ті змінні, які є домінуючими в її визначенні.

Для пояснення методу перш за все сформулюємо основні властивості карт Карно:

- клітини карти, координати яких відрізняються лише параметрами однієї змінної, називаються *сусідніми*;
- сусідні клітини, значення функцій в яких або тільки істинні, або тільки хибні, можуть об'єднуватися в групи по 2^m клітин, де m – ціле число (m = 0, 1, 2, 3, …);

- при переході до аналітичної форми запису логічної функції з карти Карно вона може записуватись незмінними координатами об’єднаних груп клітин;
- у випадку неповністю визначеної функції невизначені клітини можуть бути довизначеними, виходячи з умови одержання більшої кількості об’єднаних клітин;
- одна клітина може об’єднуватись у декілька груп.

Використання карт Карно для мінімізації логічних функцій базується на наглядному використанні операції склеювання. Дійсно, дві сусідні клітки відрізняються лише однією змінною. Тому, об’єднуючи їх, ми записуємо лише незмінні координати, тобто виносимо їх за дужки. У дужках залишаються змінні координати, які об’єднуються в одну операцією склеювання. Для отримання мінімального значення функції, представленого картою Карно, окрім правил, викладених вище, необхідно користуватись загальним правилом: одиниці або нулі повинні об’єднуватись мінімальним числом найбільших контурів.

Приклад 2.13. Мінімізувати логічну функцію, що представлена у вигляді карти Карно (рис. 2.7).

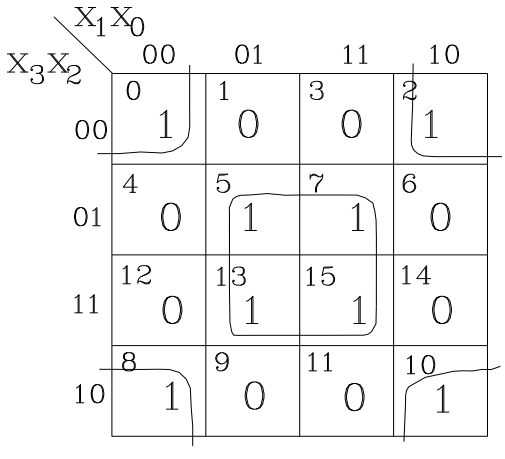


Рис. 2.7. Карта Карно

Розв'язання. Об'єднуючи сусідні клітини з істинними значеннями функції (тобто клітини 0, 2, 8, 10 та клітини 5, 7, 13, 15), записуємо їх незмінні координати:

- для центральних клітин: $y_1 = x_0\, x_2$;
- для крайніх клітин: $y_2 = \overline{x_0}\ \overline{x_2}$.

Мінімізована логічна функція має вигляд:

$y = y_1 + y_2 = x_0\, x_2 + \overline{x_0}\ \overline{x_2}$.

Аналогічний результат буде одержаний і при об'єднанні клітин з нулями. Продемонструємо це.

Для вертикальної групи клітин (клітини 1, 3, 9, 11) знаходимо: $\overline{y_3} = \overline{x_2}\ x_0$.

Для групи горизонтальних клітин (клітини 4, 6, 12, 14): $\overline{y_4} = x_2\ \overline{x_0}$

Мінімізована функція

$$\overline{y} = \overline{x_2}\ x_0 + x_2\ \overline{x_0}$$

операцією інверсії та теоремою де Моргана зводиться до отриманої раніше.

За допомогою карт Карно легко вирішуються задачі мінімізації функцій з кількістю змінних до шести включно. При більшій кількості змінних пошук мінімальних форм запису функцій значно ускладнюється і наочність карт Карно втрачається.

Мінімізація на основі використання кубічних комплексів. Кубічна форма зображення логічних функцій дає можливість просто і наглядно, за аналогією з використанням карт Карно, розв'язувати задачі мінімізації. Особливість такого способу мінімізації полягає в тому, що з кубічного комплексу $K(y)$ завжди можна виділити множину кубів $\Pi(y)$ таких, що кожний член комплексу K_0, тобто вершини куба (мінтерми), буде включеним по крайній мірі в один куб множини $\Pi(y)$. Множина кубів $\Pi(y)$ називається *покриттям комплексу* $K(y)$, або *покриттям логічної функції*. Зрозуміло, що для будь-якої логічної функції існує декілька її покриттів. У той же час, кожному покриттю $\Pi(y)$ відповідає своя диз'юнктивна нормальна форма, отримана як сума логічних добутків відповідно виділеним кубам логічної функції.

Приклад 2.14. Для кубічного комплексу з прикладу 1.31 знайти покриття логічної функції.

Розв'язання. Кубічний комплекс логічної функції має вигляд:

$$K(y) = (011;\ 100;\ 101;\ 110;\ 111;\ -11;\ 11-;\ 1-1;\ 10-;\ 1-0;\ 1--).$$

Нульовий кубічний комплекс включає всі вершини куба, тому створює покриття функції:

$$П_1(y) = K_0 = (011;\ 100;\ 101;\ 110;\ 111).$$

Усі вершини куба включаються також в одиничний кубічний комплекс K_1 , тому він також створює покриття логічної функції:

$$П_2(y) = K_1 = (-11;\ 11-;\ 1-1;\ 10-;\ 1-0).$$

Перебираючи комбінації кубів різних рангів, можна отримати покриття логічної функції:

$$П_3(y) = K_2 = (011;\ 11-;\ 10-);$$
$$П_4(y) = K_3 = (-11;\ 1-1;\ 1-0);$$
$$П_5(y) = K_4 = (011;\ 1--);$$
$$П_6(y) = K_5 = (-11;\ 1--)$$
і т.п.

Відповідні вказаним покриттям логічні функції мають вигляд:

$$y_1 = \overline{x_2} \cdot x_1 \cdot x_0 + x_2 \cdot \overline{x_1} \cdot \overline{x_0} + x_2 \cdot \overline{x_1} \cdot x_0 + x_2 \cdot x_1 \cdot \overline{x_0} + x_2 \cdot x_1 \cdot x_0 ;$$
$$y_2 = x_1 \cdot x_0 + x_2 \cdot x_1 + x_2 \cdot x_0 + x_2 \cdot \overline{x_1} + x_2 \cdot \overline{x_0} ;$$
$$y_3 = \overline{x_2} \cdot x_1 \cdot x_0 + x_2 \cdot x_1 + x_2 \cdot \overline{x_1} ;$$
$$y_4 = x_1 \cdot x_0 + x_2 \cdot x_0 + x_2 \cdot \overline{x_0} ;$$
$$y_5 = \overline{x_2} \cdot x_1 \cdot x_0 + x_2 ;$$
$$y_6 = x_1 \cdot x_0 + x_2 .$$

Складність отриманої таким шляхом логічної функції прийнято характеризувати поняттям *"ціна покриття"* (Ц$_П$). Ціна покриття дорівнює сумі цін всіх кубів, які складають дане покриття П(y):

$$Ц_П = \sum Ц_К.$$

У свою чергу, ціна одного r-куба логічної функції n змінних визначається як різниця повного числа вхідних змінних і рангу відповідного куба, тобто дорівнює числу змінних у відповідній диз'юнкції:

$$Ц_К = n - r .$$

Так, для логічної функції трьох змінних ціна 0-куба дорівнює трьом, а 2-куба – одиниці.

У відповідності до сказаного, задача мінімізації логічних функцій зводиться до пошуку покриття П(y) кубічного комплексу $K(y)$, що має мінімальну ціну.

Покриття П(y) комплексу $K(y)$, що має мінімальну ціну, називається *покриттям Квайна*, а відповідна цьому покриттю логічна функція – *мінімальною диз'юнктивною нормальною формою*.

Мінімізація неповністю визначених логічних функцій. Серед логічних функцій зустрічаються такі, досконалі форми яких містять невизначені мінтерми (макстерми). З точки зору інженерної практики це означає, що електронна апаратура, яка розробляється, повинна бути байдужою до відповідних комбінацій логічних змінних. Зустрічаються й інші ситуації, при яких в деяких комбінаціях змінних має місце неоднозначність логічної функції. В обох випадках наявність неоднозначності використовується з метою забезпечення мінімальної форми логічної функції.

Розглянемо приклад логічної функції, що приведена на рис. 2.8.

X_2 \ X_1X_0	00	01	11	10
0	0: 0	1: 1	3: 1	2: 0
1	4: 1	5: X	7: 1	6: X

Рис. 2.8. Приклад логічної функції

Невизначеність її значень, що відображена у клітинах 5 та 6, може бути використана для мінімізації. Підстановка в указані клітини значень "**1**" дає можливість об'єднати групи клітин 1, 3, 5, 7 і 4, 5, 6, 7 та одержати мінімізовану функцію:

$$y = x_0 + x_2 .$$

Слід зазначити, що для більш складних функцій можуть використовуватись декілька різних способів довизначення, в результаті можна отримувати мінімальні функції різної складності.

Недовизначені функції часто зустрічаються при реалізації перетворювачів кодів.

Сумісна мінімізація декількох логічних функцій. У практичних задачах цифрової електроніки дуже часто зустрічаються ситуації, коли використовується декілька різних функцій з однаковими змінними. Як приклад, розглянемо дві наступні функції:

$$Y_1 = \bigvee^{15} 3, 6, 7 = \overline{x_2}\, x_1\, x_0 + x_2\, x_1\, \overline{x_0} + x_2\, x_1\, x_0;$$
$$Y_2 = \bigvee^{15} 0, 1, 3 = \overline{x_2}\, \overline{x_1}\, \overline{x_0} + \overline{x_2}\, \overline{x_1}\, x_0 + \overline{x_2}\, x_1\, x_0.$$

Кожну з них можна мінімізувати окремою. При такій мінімізації маємо:

$$Y_1 = x_1\, x_0 + x_2\, x_1;$$
$$Y_2 = \overline{x_2}\, \overline{x_1} + \overline{x_2}\, x_0.$$

Обидві функцій можуть реалізовуватись окремо. Але, порівнюючи приведені функції, бачимо, що в обох з них є загальний мінтерм. Це дає підставу окремо реалізувати загальний мінтерм, а решту мінімізовувати незалежно для кожної з функцій. У такому випадку маємо:

$$Y_1 = \overline{x_2}\, x_1\, x_0 + x_2\, x_1;$$
$$Y_2 = \overline{x_2}\, \overline{x_1} + \overline{x_2}\, x_1\, x_0.$$

При побудові схеми окремо реалізуються всі терми, а потім виконується їх об’єднання через функцію **АБО**.

Мінімізація логічних функцій з великою кількістю змінних. Практичні комбінаційні схеми можуть мати десятки-сотні входів і виходів і описуватись сотнями-тисячами термів. У таких випадках для мінімізації широко використовується метод декомпозиції. Він полягає у наступному. Логічна функція з великою кількістю змінних – наприклад,

$$y = f(x_0, x_1, x_2, \ldots, x_{10}),$$

розбивається на декілька підлеглих функцій виду:

$$y = f(x_0, x_1), f_0(x_2, x_3, x_4), f_1(x_5, x_6), f_2(x_7, ..x_{10}).$$

При виборі функцій f_0, f_1, f_2 схемотехніки використовують власний досвід, але при їх вдалому виборі апаратна реалізація мінімізованої функції може бути значно економнішою, ніж будь-які способи прямої реалізації.

Слід зазначити, що інтенсивний розвиток мікросхемотехніки, у тому числі програмованих логічних інтегральних схем (ПЛІС), постійних запам'ятовуючих пристроїв (ПЗП), а також ряду функціональних мікросхем змінив акценти в задачах мінімізації. Широке використання ПЛІС привело до розвитку методів їх програмування, які одночасно використовують і алгоритмічні засоби мінімізації кількості логічних елементів, що використовуються. Одним з таких алгоритмічних методів є *метод Квайна – Мак-Класкі*.

Мінімізація логічних функцій за допомогою метода Квайна–Мак-Класкі. Метод використовується при необхідності мінімізації логічних функцій з великою кількістю змінних. Він, як і інші подібні методи, що мають властивість однозначності алгоритму, широко використовується і розвиваються розробниками програмних засобів для ПЛІС. Здебільшого це фірми-виробники програмованих логічних матриць (наприклад, ALTERA, XILLINX та інші).

Алгоритм пошуку мінімальних диз'юнктивних форм зводиться до наступного:

1. Знаходиться покриття П(y) заданої функції. Для цього формується кубічний комплекс логічної функції і в кожному i-му кубічному комплексі відмічають куби (імпліканти), які не утворили (i + 1)-й кубічний комплекс. Відмічені імпліканти, які називаються *простими*, створюють покриття заданої логічної функції;

2. Будують таблицю покриттів матриці Квайна. Рядки вказаної таблиці відповідають простим імплікантам, а стовбці – 0-кубам (конституєнтам одиниці) логічної функції. На перетині i-го рядку та j-го стовбця ставиться

мітка, якщо імпліканта і покриває конституєнту *j* (імпліканта *i* покриває конституєнту *j* в тому випадку, якщо вона відрізняється від неї незалежними аргументами);

3. Визначають покриття мінімальної вартості. Для цього:

- виділяють ядро Квайна. Якщо 0-куб заданої логічної функції покривається лише однією простою імплікантою, то остання є суттєвою та входить в ядро Квайна і, відповідно, у покриття мінімальної вартості;
- з таблиці викреслюються стовбці і рядки, які покриті імплікантами ядра Квайна. Якщо в отриманій після викреслювання таблиці містяться прості імпліканти, вони також включаються в ядро Квайна з послідуючим викресленням відповідних рядків і стовбців;
- стискається таблиця по стовбцях, для чого з неї викреслюються стовбці, які повністю включаються в будь-який з решти стовбців;
- стискається таблиця по рядках, для чого з неї викреслюються рядки, які повністю включаються в будь-яку з решти рядків.

Послідовно стискаючи таблицю по рядках і стовбцях, отримують *циклічну таблицю*, імпліканти якої повинні входити в покриття логічної функції мінімальної вартості.

Перелік посилань

1. Рябенький В.М., Жуйков В.Я., Гулий В.Д. Цифрова схемотехніка: Навч. посібник. – Львів: Новий світ – 2000, 2009, – 736 с.

Лекція №3

Мова VHDL

Розробка сучасних інтегральних мікросхем є складною інженерно-технічною проблемою, вирішення якої неможливе без широкого застосування систем автоматизованого проектування. Створення подібних систем, як відомо, потребує засобів формального опису структур і функцій об’єктів проектування. Мова VHDL (Very high speed integrated circuits Hardware Description Language) була розроблена у 1980 році в результаті реалізації в США проекту по створенню надшвидкісних інтегральних схем. У 1987 році Інститутом Інженерів з Електрики та Електроніки (IEEE) ця мова була визнана в якості стандарту США [1]. Зараз VHDL є найбільш поширеною у світі мовою такого призначення. Вона застосовується у багатьох системах автоматизованого проектування, кількість користувачів яких стрімко зростає.

Мова VHDL служить для опису моделі цифрового пристрою (приладу, системи). Опис на мові VHDL визначає зовнішні зв’язки пристрої («вид ззовні» або інтерфейс) і один або декілька «видів зсередини» (див. рис. 3.1) [1]. Вид зовні задає інтерфейс пристрою, набір сигналів, якими обмінюється з зовнішнім світом. Цей вид описує абстрактне уявлення пристрої «в цілому» і позначається англійським терміном entity, що в дослівному перекладі означає «сутність» і найбільш точно відображає зміст вистави. Однак у літературі термін «сутність» не знайшов широкого розповсюдження, для позначення зовнішнього опису об’єкта використовуються терміни «інтерфейс об’єкта», «декларативна частина» та інші. У цьому посібнику буде використовуватися термін «інтерфейс об'єкта» або просто «інтерфейс».

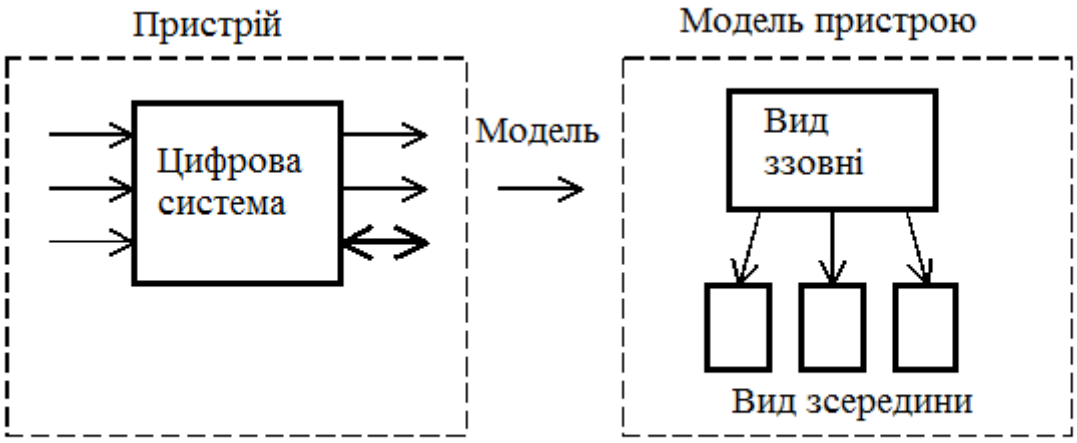


Рис. 3.1. Цифровий пристрій і його модель

Вид зсередини визначає функціональні можливості пристрою або його структуру. Внутрішня будова об’єкта визначає архітектура.

Як і в мовах програмування, мова VHDL має свої правила, в тому числі правила опису імен змінних, об’єктів, типів даних та інших параметрів [2, 3].

Ідентифікатори в мові VHDL

Ідентифікатори – це послідовність літер і цифр довільної довжини. Дозволеними символами є прописні (A ... Z), малі (a ... z), цифри (0 ... 9), знак підкреслення. Перший символ має бути буквою, а останній символ не може бути знаком підкреслення. Малі та великі літери вважаються ідентичними, наприклад, Count, COUNT і CouNT розглядаються як одне слово. Знаки підкреслення не повинні слідувати один за одним. Коментарі починаються з подвійного дефісу та слідують до кінця рядка, наприклад. Наступні ідентифікатори зарезервовані для використання в якості ключових слів і не можуть використовуватись інакше:

Abs	Access	after	alias	All
And	architecture	array	begin	Block
Body	Buffer	case	component	Configu-ration

Constant	disconnect	downto	else	Elsif
End	Entity	file	for	function
generate	Generic	guarded	if	In
inout	Is	label	library	linkage
loop	Map	mod	nand	New
next	Nor	not	null	Of
on	Open	or	others	Out
package	Port	procedure	process	Range
record	Register	rem	select	severity
signal	Subtyupe	then	to	Transport
type	Units	until	use	vriable
wait	When	while	with	Xor

Об’єкти мови VHDL

Об’єкти – це область зберігання даних певного типу. Створюються об’єкти за допомогою декларації об’єкта, наприклад:

variable COUNT: INTEGER;

У результаті породжується об’єкт з ім’ям COUNT, який зберігає цілочисельну величину. Крім того, COUNT декларується як клас variable.

Об’єкти-дані можуть бути трьох класів:

- Constant (константа) – може зберігати окреме значення певного типу. Це значення визначається об’єкту на початку моделювання і не може змінюватися в процесі моделювання.

- Variable (змінна) – об’єкт цього класу може зберігати окреме значення певного типу, проте, в процесі моделювання йому можуть присвоюватися різні значення. Для цього використовуються вирази привласнення (variable assignment statement).

- Signal (сигнал) – об’єкт даного класу має попереднє значення, має поточне значення і набір наступних значень.

Об’єкти класу signal моделюють дротові з’єднання в схемах, у той час як змінні (variable) і константи (constant) використовуються для моделювання поведінки схеми, вони аналогічні об’єктів, що використовуються в мовах програмування С та Pascal.

Декларація констант

Декларації об’єктів (object declaration) має на меті дати ім’я об'єкту, оголосити його тип, клас і навіть присвоїти значення. Приклади декларацій констант описані нижче:

constant RISE_TIME: TIME: = 10ns;

constant BUS_WIDTH: INTEGER: = 8:

У першому випадку оголошується об’єкт RISE_TIME, який зберігає значення типу TIME, об’єкту на початку моделювання присвоюється величина 10 наносекунд. У другому випадку оголошується, що BUS_WIDTH (ширина шини) типу INTEGER (ціле) і їй присвоєно значення 8.

Декларація змінних

Приклади декларацій об’єктів класу variable наведені нижче:

variable CTRL_STATUS: BIT_VECTOR (10 downto 0);

variable SUM: INTEGER range 0 to 100: = 10;

variable FOUND, DONE: BOOLEAN;

У першому випадку декларується змінна CTRL_STATUS як масив з 11 елементів, причому, кожен елемент типу BIT. У другому випадку змінна SUM декларується як цілочисельна, що лежить в діапазоні від 0 до 100, на початку моделювання змінній присвоюється значення 10. Якщо змінній на початку моделювання не задано значення, то використовується значення за замовчуванням. Їм служить «крайнє ліве» значення в наборі значень даного типу. Наприклад, змінна типу BOOLEAN має набір значень (FALSE, TRUE) і в третьому прикладі початкове значення змінних FOUND і DONE буде взято за замовчуванням, тобто FALSE.

Декларація сигналів

Декларації об’єктів класу signal схожі на декларації змінних:

signal CLOCK: BIT;

signal DATA_BUS: BIT_VECTOR (0 to 7);

signal GATE_DELAY: TIME: = 10 ns;

У першому прикладі декларується об’єкт CLOCK типу BIT, початкове значення при моделюванні буде взято за замовчуванням, тобто 0 (набір значень типу BIT: 0,1 і крайнє ліве значення 0).

Не всі об’єкти в мові VHDL створюються шляхом декларування, наприклад, вхідні/вихідні порти завжди вважаються об’єктами класу signal.

Типи даних

Кожен об’єкт в мові VHDL може зберігати значення, які відносяться до певного набору значень. Ця множига значень декларується за допомогою оголошення типу (type declaration). Тип – це ім’я, яке пов’язане з певним набором значень і набором операцій. Наприклад, BOOLEAN має набір значень FALSE, TRUE і набір операторів: and, or, nor, nand, not. У мові VHDL є можливість створювати нові типи з використанням декларацій і завдання набору операцій.

Всі можливі типи в VHDL розпадаються на чотири великі категорії:

- Scalar (скалярні), які в свою чергу поділяються на чотири типи:
  - -Enumeration (перераховуваний тип),
  - -Integer (цілочисельний тип),
  - -Physical (фізичний тип),
  - -Floating point (тип «з плаваючою комою»).
- Composite (композитні) – вони складаються з елементів одного типу (масиви) або різного типу (записи),
- Access type (типи доступу) – забезпечують доступ до даного типу через покажчики,

- File types (файл) – забезпечує доступ до об’єктів, що містять послідовності значень даного типу.

Перераховуваний тип

При декларації користувачем визначається набір певних значень, наприклад:

type MVL is ('U', '0 ', '1', 'Z);

type MICRO_OP is (LOAD, STORE, ADD, SUB, MUL, DIV);

MVL – перераховуваний тип з упорядкованим набором значень: 'U', '0 ', '1', і 'Z'. MICRO_OP має набір значень: LOAD, STORE, ADD, SUB, MUL, DIV. Порядок запису значень в декларації визначає лексику, тобто значення праворуч завжди більше значення ліворуч: STORE <DIV is true, SUB> MUL is false. Значення в перелічуваному типі мають позиційний номер. Позиційний номер крайнього лівого елемента – 0. Значення в перелічуваному типі називають enumeration literals (літерали перераховуваного типу). Наприклад, у декларації:

type CAR_STATE is (STOP, SLOW, MEDIUM, FAST);

літералами є STOP, SLOW, MEDIUM, FAST і тільки вони можуть присвоюватися змінній CAR_STATE.

Цілочисельний тип

Integer – задає тип, набір значень якого знаходиться в заданому цілочисельному діапазоні, наприклад:

type INDEX is range 0 to 15;

type WORD_LENGTH is range 31 downto 0;

subtype DATA_WORD is WORD_LENGTH range 15 downto 0;

type MY_WORD is range 4 to 6;

INDEX – це змінна цілочисельного типу, набір значень якої знаходиться у діапазоні від 0 до 15. DATA_WORD – підтип WORLD_LENGTH, набір значень знаходиться у в діапазоні від 15 до 0. На відміну від перелічуваного типу в

цілочисельному типі позиційний номер дорівнює величині значення, наприклад, для значення 31 змінної WORD_LENGTH позиція дорівнює 31.

Тип «з плаваючою комою»

Тип floating point має набір значень у заданому діапазоні дійсних чисел, наприклад:

type TTL_VOLTAGE is range 1.4 to 5.5

type REAL_DATA is range 0.0 to 31.9;

Літерали типу floating point відрізняються від цілочисельних присутністю крапки (.). У результаті 0 – це цілочисельний літерал, а 0.0 – це літерал типу з плаваючою комою.

Фізичний тип

Physical type зберігає значення, які представляють собою результати вимірювань фізичних величин: часу, довжини, напруги, струму тощо Значення цього типу виражаються цілими, помноженими на базову одиницю, наприклад:

type CURRENT is range 0 to 1 E9

units

nA; - (base unit) nano-ampere

uA = 1000 nA; micro-ampere

mA = 1000 uA; milli-ampere

Amp = 1000 mA; ampere

end units;

subtype FILTER_CURRENT is CURRENT range 10 mA to 5 mA;

Тут CURRENT визначається як фізичний тип, який має значення в діапазоні від 0 nA до 10^9 nA. Базовою одиницею є наноампер, а всі інші є похідними. Позиційний номер значення дорівнює числу базових одиниць, представлених даними значенням, наприклад, 2 mA мають позиційний номер 2000, в той час як 100 nA займають позицію 100. Фізичний літерал записується як ціле, за яким слідує назва одиниці вимірювання (пробіл обов’язковий).

Тип «масив»

Об’єкт типу array складається з елементів одного типу. Нижче наведено приклади масивів:

type ADDRESS_WORD is array (0 to 63) of BIT;

type DATA_WORD is array (7 downto 0) of MVL;

type ROM is array (0 to 125) of DATA_WORD;

ADDRESS_WORD – однорозмірний масив з 64 елементів типу BIT. DATA_WORD – однорозмірний масив з 8 елементів типу MVL. ROM – масив з 126 елементів типу DATA_WORD, тобто в даному випадку маємо справу з масивом масивів.

Доступ до елементів масиву здійснюється за допомогою індексів, наприклад, ADDRESS_WORD (26) посилається на 27-ой елемент масиву ADDRESS_WORD.

Операції мови VHDL

У мові VHDL існують операції наступних категорій:

1. Логічні операції
2. Операції відношення
3. Операції додавання/віднімання
4. Операції множення/ділення
5. Інші

Пріоритет операцій зростає від категорії 1 до категорії 5. Операції однієї категорії мають однаковий пріоритет і виконуються в послідовності зліва направо. Дужки використовуються для зміни послідовності виконання.

Логічні операції

Існує шість логічних операцій: and, or, nand, nor, xor, not. Операції застосовуються до типів BIT, BOOLEAN, до однорозмірних масивів BIT і BOOLEAN. При виконанні операцій бітові значення «0» і «1» інтерпретуються як булеві FALSE і TRUE. Результат має той самий тип, що і операнди. Операція not – унарна операція, вона має пріоритет категорії 5.

Операції відношення

У мові VHDL є наступні операції відношення: =, / =, <, <=,>,> =.

Результатом виконання будь-якої операції відношення є булевий вираз BOOLEAN. Операції рівності (=) і нерівності (/ =) застосовуються до усіх типів, за винятком типу «файл». Решта операцій застосовуються до скалярних типів (цілочисельного, перераховуваного) або над масивами дискретного типу. Коли операндами є масиви, то порівняння виконується зліва направо по одному елементу, наприклад при наступному порівнянні:

BIT_VECTOR '(0 "," 1 "," 1 ") <BIT_VECTOR' ('1 ', '0', '1 ')

отриманий результат TRUE, тому що перший елемент вектора зліва менше першого елемента вектора справа. Інший приклад, якщо декларується тип:

type MVL is ('U', '0 ', '1', 'Z');

то результат порівняння:

MVL '(' U ') <MVL' ('Z')

буде TRUE, тому що 'U' знаходиться лівіше 'Z'.

Операції підсумовування, віднімання, конкатенації

Операції мають позначення: +, -, &.

Операнди, що беруть участь в операціях додавання (+) і віднімання (-) повинні бути одного числового типу, результат отримується того ж типу. Операндами в конкатенації (&) можуть бути або окремі елементи або одномірні масиви. Результат видається у вигляді масиву, наприклад при виконанні конкатенації:

«0» & «1»

утворюється масив символів «01», або:

'C' & 'A' & 'T'

дає "CAT".

Операції множення, ділення

До цієї групи операцій відносяться: *, /, mod, rem. Операнди множення (*) і ділення (/) повинні бути одночасно або цілочисельного типу (integer) або типу

з плаваючою комою (floating point). Результат завжди того типу, що і операнди. Операція множення може мати один операнд фізичного типу, а інший – або цілочисельного, або натурального типу. Результат видається у вигляді фізичного типу. В операції ділення допустимо ділити об’єкт фізичного типу на цілочисельний або натуральний. Результат завжди фізичного типу. Ділення фізичного типу на фізичний дає цілочисельний результат.

Операції знаходження залишку від ділення (rem) та «ділення за модулем» (mod) в якості операндів можуть мати цілочисельний тип і результат цілочисельного типу.

Результат rem має знак першого операнда і визначається наступним чином:

A rem B = A - (A / B) * B

Результат mod має знак другого операнда і визначається наступним чином:

A mod B = A - B * N,

де N – деяке ціле.

Інші операції

До їх числа слід віднести: Abs, ** та інші. Операція знаходження абсолютного значення (abs) сумісна з будь-яким числовим типом операнда. Операція піднесення до степеня (**) операндом зліва має цілочисельний або тип з плаваючою комою, а в якості правого операнда (степінь) – тільки цілочисельний тип.

Послідовні оператори

До послідовних операторів (Sequential statements) відносяться оператори присвоювання (Assignment statements), умовні оператори if (if statements) та case (case statements), оператори циклу (loop - next statements) та інші.

Оператор присвоювання

Ці оператори поділяються на оператори присвоювання змінної (позначається знаком: =) і оператори присвоювання сигналу (<=). В обох

випадках спочатку обчислюється значення виразу, що стоїть праворуч від знаку рівності, а потім отримане значення присвоюється змінній або сигналу, що стоїть ліворуч від знака. Наприклад, оператор

abar: = not a;

задає нове значення змінної abar, а саме – інверсне значення a.

Оператор

z <= not (a and b);

задає нове значення сигналу z, яке виходить праворуч від знака <=.

Оператор if

У загальному випадку if представляє собою послідовність виразів, які обумовлюють умови. В якості умов застосовуються будь-які вирази, при обчислення яких результатом є булева величина (FALSE і TRUE).

if булевий_вираз then

послідовні_вирази

[elsif булевий_вираз then

послідовні_вирази]

[else

послідовні_вирази]

end if;

Вираз if обчислюється шляхом перегляду умов однієї за іншою, поки не буде знайдено істинна. Потім обчислюється набір послідовних виразів, пов’язаних з цією умовою. Умов виду elsif у виразі if може бути від 0 і більше. Може використовуватися і умова else. Вирази if можуть вкладатися один в один без обмежень.

Розглянемо простий приклад

```
if CTRL = '1 'then
MUX_OUT <= "10";
else
MUX_OUT <= "01";
```

```
end if;
```

Якщо керуючий сигнал CTRL дорівнює «1», то вихідний сигнал MUX_OUT приймає значення «10», інакше MUX_OUT прийме значення «01». На цьому оператор if завершується.

Розглянемо складніший приклад

```
if CTRLI = '1 'then
if CTRL2 = 0 "then
MUX_OUT <= "0010";
else
MUX_OUT <= "0001";
end if;
else
if CTRL2 = "0" then
MUX_OUT <= "1000";
else
MUX_OUT <= "0100";
end if;
end if;
```

Якщо керуючий сигнал CTRL1 дорівнює «1», тоді за умови (відкривається вкладене if) CTRL2 = 0 вихідний сигнал MUX_OUT <=«0010», інакше (тобто при будь-яких інших значеннях сигналу CTRL2) MUX_OUT <=«0001» . Тут завершується внутрішній (вкладений) оператор if. Інакше (тобто при будь-яких інших значеннях сигналу CTRL1), якщо (відкривається новий вкладений оператор if) CTRL2 = «0», тоді MUX_OUT <= «1000», інакше (тобто при будь-яких інших значеннях сигналу CTRL2) MUX_OUT <= «1000». Тут завершується внутрішній (вкладений) оператор if, а також завершується і зовнішній оператор if.

Оператор case

Оператор case має наступний формат:

case вираз is

when варіант вибору => послідовні_вирази

when варіант вибору => послідовні_вирази

- довільне число варіантів вибору.

when others => послідовні_вирази]

end case;

При обчисленні виразу case вибирається одна з гілок у відповідності зі значенням виразу. Вираз може мати значення перераховуваного типу або типу одновимірного масиву. Розглянемо приклад:

type WEEK_DAY is (MON, TUE, WED, THU, FRI, SAT, SUN);

type DOLLARS is range 0 to 10;

variable DAY: WEEK_DAY;

variable POCKET_MONEY: DOLLARS;

case DAY is

when TUE => POCKET_MONEY: = 6; - гілка 1

when MON I WED => POCKET_MONEY: = 2; - гілка 2

when FRI to SUN => POCKET_MONEY: = 7; - гілка 3

when others => POCKET_MONEY: = 0; - гілка 4

end case;

Змінна WEEK_DAY має значення перераховуваного типу (дні тижня). Змінна DOLLARS має цілочисельні значення в діапазоні від 0 до 10. Гілка 1 вибирається, коли день тижня TUE. Гілка 2 вибирається у випадку коли дні тижня MON або WED (вертикальна риса означає АБО). Гілка 3 покриває значення від FRI до SUN, тобто FRI, SAT і SUN. Гілка 4 покриває всі значення, що залишилися, тобто THU.

Оператор loop

Оператор loop використовується для задання ітерації набору послідовних виразів.

[Ярлик для loop:] ітераційна схема loop

послідовні_вирази

end loop [ярлик для loop];

Існує три ітераційних схеми. Перша має форму:

for ідентифікатор in діапазон

Приклад використання схеми:

FACTORIAL: = 1;

for NUMBER in 2 to N loop

FACTORIAL: = FACTORIAL * NUMBER;

end loop;

Тіло loop виповнюється N-1 раз, при цьому ідентифікатор NUMBER в кінці кожної ітерації збільшується на 1. Мається на увазі, що ідентифікатор цілочисельного типу і значення його лежать в діапазоні від 2 до N.

Другою ітераційною схемою є наступна:

while булевий_вираз

Приклад використання схеми:

J: = 0; SUM: = 10;

WH-LOOP: while J <20 loop - loop має ярлик WH_LOOP

SUM: = SUM * 2;

J: = J +3;

end loop;

Вирази в тілі loop виконуються один за іншим і ця послідовність повторюється, поки умова J <20 є істинною.

Третьою є конструкція, в якій ітераційна схема не задається і вихід з loop здійснюється з використанням виразів: exit, next або return, наприклад:

SUM: = 1; J: = 0;

```
L2: loop - loop має ярлик
J: = J +21;
SUM: = SUM * 10;
exit when SUM> 100;
end loop L2;
```

У цьому прикладі вираз exit змушує виходити з петлі L2 коли SUM стає більше 100. При відсутності появи exit loop буде виконуватися нескінченно.

Паралельні оператори

До паралельних операторів (Concurrent statements) відносяться: оператор process, оператор паралельного виклику процедури, оператор конкретизації компонента, оператор генерації (generate) та інші [1]. Паралельні оператори визначають паралельну поведінку схем, порядок їх виконання не залежить від їхньої появи всередині блоку.

У загальному вигляді оператор process може бути записаний таким чином:

```
[ім’я процесу:] [postponed] process [(список)]
розділ декларацій
begin
оператори
end process [ім’я процесу];
```

Ім’я процесу і ключове слово [postponed] є необов’язковими і часто відсутні. Список, що знаходиться після ключового слова process, хоча і є необов’язковим, на практиці використовується досить часто для вказівки сигналів запуску. Перед ключовим словом можуть знаходитись різного роду декларації типів, констант атрибутів і т. д.

Хоча process є паралельним оператором, він може містити послідовні оператори. Усередині process не можуть декларуватися сигнали.

Опис інтерфейсу пристрою

Для опису інтерфейсу пристрою мовою VHDL використовується декларація інтерфейсу entity [1]. Вона задає ім’я опису, імена інтерфейсних портів, напрямок передачі, тип портів. Порт – це сигнальна лінія (шина), за допомогою якої пристрій (модель) взаємодіють з навколишнім середовищем.

Кожен інтерфейсний порт може функціонувати у наступних режимах:

in – значення порту тільки зчитується для використання всередині моделі;

out – значення порту може тільки оновлюватися моделлю, але не зчитується;

inout – двонаправлених порт, значення прочитується і оновлюється моделлю;

buffer – буферний порт, значення прочитується і оновлюється моделлю, але джерелом сигналу може бути або буфер, або одиночний джерело.

Наприклад, для напівсуматора (рис. 3.2) декларація буде виглядати наступним чином:

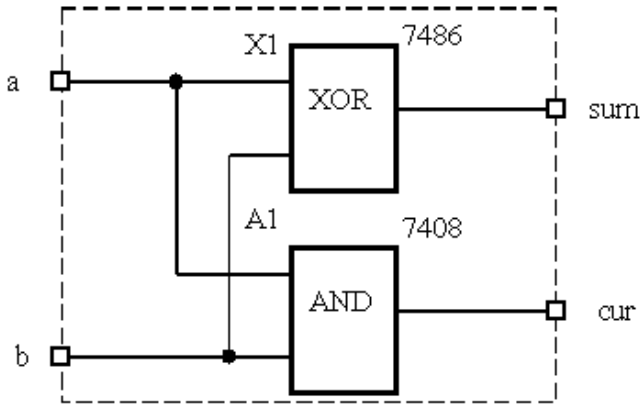


Рис. 3.2. Напівсуматор

```
entity half_adder is
port (a, b: in BIT; sum, cur: out BIT);
end half_adder;
```

Тут опис з ім’ям half_adder має два вхідних порти: a і b (in – означає вхідний порт), і два вихідних порти: sum і carry (out – означає вихідний порт). Тип портів визначений як BIT – означає, що сигнали на лініях портів можуть приймати значення: «0» або «1».

Іншим прикладом може послужити дешифратор, показаний на рис. 3.3.

```
entity dec2x4 is
port (a, b, enable: in BIT; z: out BIT_VECTOR (0 to 3));
end dec2x4;
```

Тут опис з ім’ям dec2x4 має три вхідних порти і чотири вихідних. Вихідні порти описані як масив. BIT_VECTOR – це одновимірний масив, діапазон задається параметром Z (0 to 3).

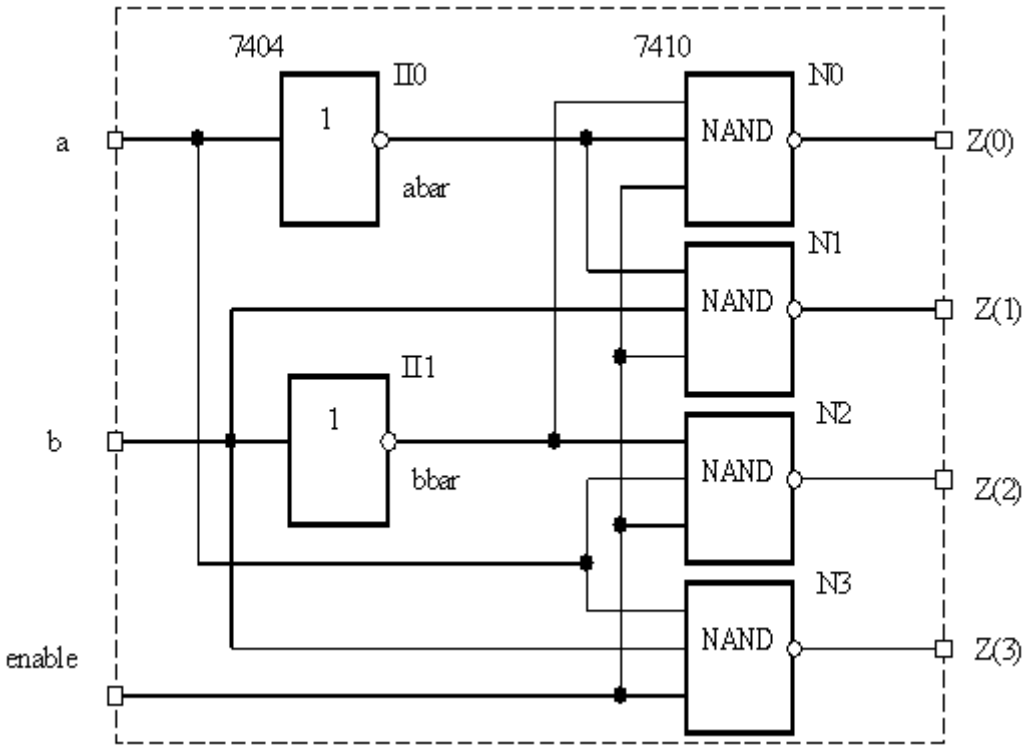


Рис. 3.3. Дешифратор 2x4

Опис архітектури пристрою

Архітектура (architecture body) моделює погляд на пристрій «зсередини». Пристрій може розглядатися і описуватися по-різному. Він може бути представлений або як композиція більш простих модулів (структурний стиль моделювання), або як набір паралельно виконуваних алгоритмів (потоковий стиль – dataflow), або описуватись як процес послідовного виконання операцій (поведінковий стиль – behavioral), або може описуватись поєднанням зазначених стилів.

З одним інтерфейсом типу entity може бути пов’язано кілька архітектур.

Поведінковий опис

Функціонування пристрою розглядається як процес послідовного обчислення виразів, що входять до process. Поштовхом до запуску процесу є зміна (event-подія) будь-якого сигналу, з вхідних в список «сприйнятих» сигналів (sensitivity list). Цей список зовні дуже схожий на список параметрів в інших мовах програмування високого рівня. Розглянемо поведінкову модель дешифратора dec2x4 (рис. 3.2). Декларація інтерфейсу його вже наводилася, а тепер вона буде доповнена описом архітектури поведінкового стилю.

```
entity dec2x4 is
port (a, b, enable: in BIT; z: out BIT_VECTOR (0 to 3));
end dec2x4;
architecture dec_seq of dec2x4 is
begin
process (a, b, enable)
variable abar, bbar: BIT;
begin
abar: = not a; - вираз 1
bbar: = not b; - вираз 2
if enable = '1 'then - вираз 3
z (3) <= not (a and b); - вираз 4
z (2) <= not (a and bbar); - вираз 5
z (1) <= not (abar and b); - вираз 6
z (0) <= not (abar and bbar); - вираз 7
else z <= "1111";
end if;
end process;
end dec_seq;
```

Архітектура має своє ім’я dec_seq. Про приналежність архітектури до опису інтерфейсу dec2x4 говорить вираз: dec_seq of dec2x4. Список сигналів, якими запускається процес, подано в дужках: process (a, b, enable). До початку процесу, якщо необхідно, декларуються змінні variable abar, bbar: BIT. Зона дії їх обмежена даним процесом (від begin до end process). Припустимо, що в момент T відбулася зміна сигналу a. Нове значення змінної abar (вираз 1) обчислюється і присвоюється без затримки – миттєво (в якості знака присвоювання в даному випадку використовується =). Без затримки будуть обчислені вирази 2 і 3.

Далі в ході процесу один за іншим обчислюються вирази від 4 до 7, а нові значення сигналам z(0), z(1), z(2), z(3) будуть присвоєні (в якості символу оператора присвоєння використовується <=) лише через деяку, нехай навіть фіктивну (нескінченно малу), затримку D. Це суттєва відмінність процедур присвоєння для змінних і для сигналів.

Розглянемо приклад мультиплексора 4x1 (рис. 3.3). Чотири вхідних сигналb A, B, C і D виводяться через один порт Z. Комутація здійснюється сигналом CTRL.

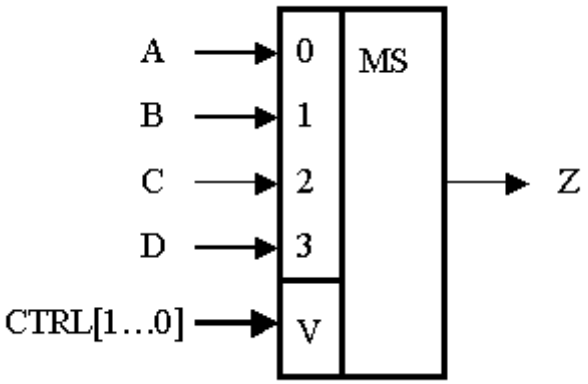


Рис. 3.3. Мультиплексор 4x1

```
entity MUX is
port (A, B, C, D: in BIT; CTRL: in BIT_VECTOR (0 to 1);
Z: out BIT);
```

```
end MUX;
architecture MUX_BEHAVIOR of MUX is
begin
PMUX: process (A, B, C, D, CTRL)
variable TEMP: BIT;
begin
case CTRL is
when "00" => TEMP: = A:
when "01" => TEMP: = B;
when "10" => TEMP: = C;
when "11" => TEMP: = D;
end case;
Z <= TEMP;
end process PMUX;
end MUX_BEHAVIOR;
```

Архітектура має назву MUX_BEHAVIOR, процес має ярлик PMUX. Змінній TEMP в залежності від значення сигналу CTRL присвоюється значення одного з вхідних портів: A, коли CTRL має значення "00"; B, коли CTRL має значення "01"; A, коли CTRL має значення "10"; D, коли CTRL має значення "11".

Для опису синхросигналів зазвичай використовується атрибут EVENT. Він приймає значення TRUE коли відбувається «подія» - зміна сигналу. Наприклад, поведінка D-тригера, показаного на рис. 3.4, можна описати так:

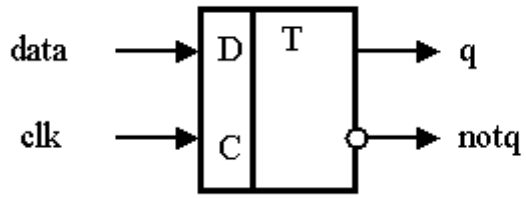


Рис. 3.4. D-тригер

```
entity dff is
port (data, clk: in BIT
q, notq: out BIT);
end dff;
architecture behav of dff is
begin
process (clk)
begin
if (clk'event and clk = '1 ') then
q <= data;
notq <= not data;
end if;
end process;
end behav;
```

Вираз (clk'event and clk = '1 ') задає умову: якщо сигнал clk змінюється з "0" на "1".

Потоковий опис

Опис в стилі dataflow використовує в архітектурі паралельно обчислювані вирази. Кількість паралельно обчислюваних виразів може бути будь-якою. Так як обчислення відбуваються паралельно, то порядок запису виразів не має значення. Поштовхом до початку обчислень є зміна будь-якого із сигналів, що входять у вирази. Розглянемо модель однорозрядного суматора, показаного на рис. 3.5.

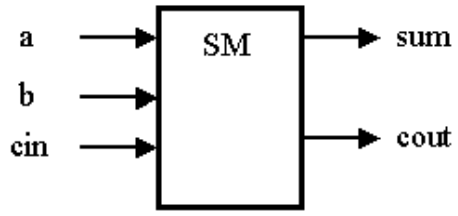


Рис. 3.5. Однорозрядний сумматор

```
entity full_adder is
port (a, b, cin: in BIT; sum, cout: out BIT);
end full_adder;
architecture full_ad_conc of full_adder is
begin
sum <= (a xor b) xor cin after 15 ns;
cout <= (a and b) or (b and cin) or (cin and a) after 10ns;
end full_ad_conc;
```

Для опису потоку даних тут використовуються два вирази. Завжди, коли відбуваються зміни сигналів a, b або cin, обидва вирази обчислюються і сигналам sum і cout присвоюються нові значення через 15ns і 10ns, відповідно. Знак <= – оператор присвоювання. Вираз after 15ns означає затримку, внесену логікою, що реалізує обчислення виразу для sum. Якщо затримка не введена явно, то передбачається наявність нескінченно малої затримки D. Такий прийом дозволяє впорядкувати події опису dataflow. Розглянемо схему рис. 3.6 і її модель.

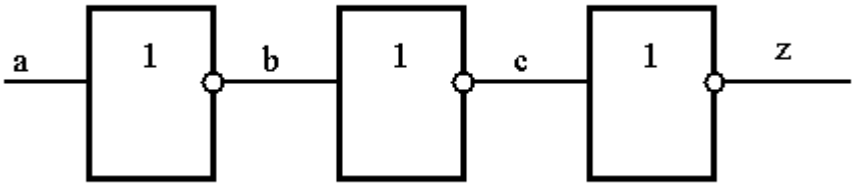


Рис. 3.6. Ланцюжок інверторів

```
entity fast_inverter is
port (a: in BIT; z: out BIT);
end fast_inverter;
architecture delta_delay of fast_inverter is
signal b, c: BIT;
begin
z <= not c; - вираз 1
```

c <= not b; - вираз 2

b <= not a; - вираз 3

end delta_delay;

У виразах 1, 2 і 3, які присвоюють значення змінним z, c, b, мається на увазі затримка D. Наприклад, в момент T відбувається зміна сигналу a. Запускається процес обчислення виразу 3 і через затримку D змінній b буде присвоєно нове значення (в момент T + D). У свою чергу зміна сигналу b запустить процес обчислення виразу 2 і значення сигналу c буде оновлено в момент T +2 D. Зміна сигналу c призведе до запуску процесу обчислення виразу 1 і значення сигналу z буде змінене в момент T +3 D. Часові співвідношення у схемі ілюструє рис. 3.7.

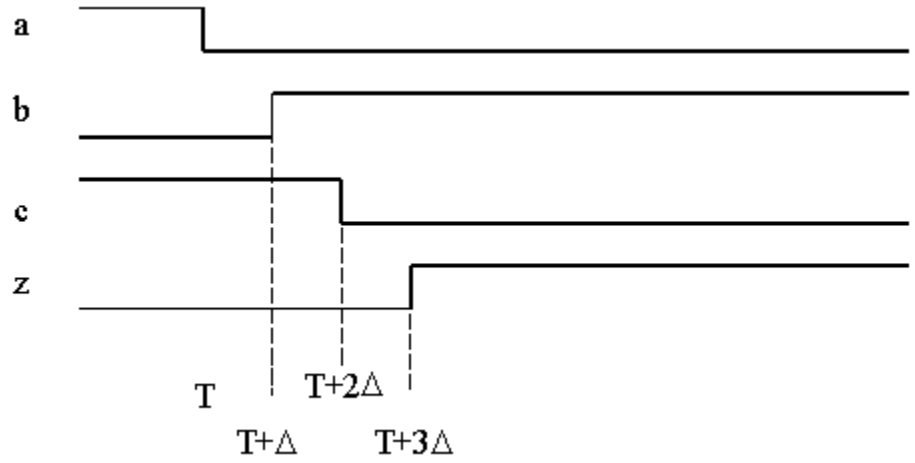


Рис. 3.7. Часові співвідношення в ланцюжку інверторів

Структурний опис

Структурний опис інтерпретує пристрій як набір компонентів, пов’язаних між собою сигналами [1]. Грубо кажучи – це таблиця з’єднань (netlist). Розглянемо просту схему керування (рис. 3.8) і її структурну модель.

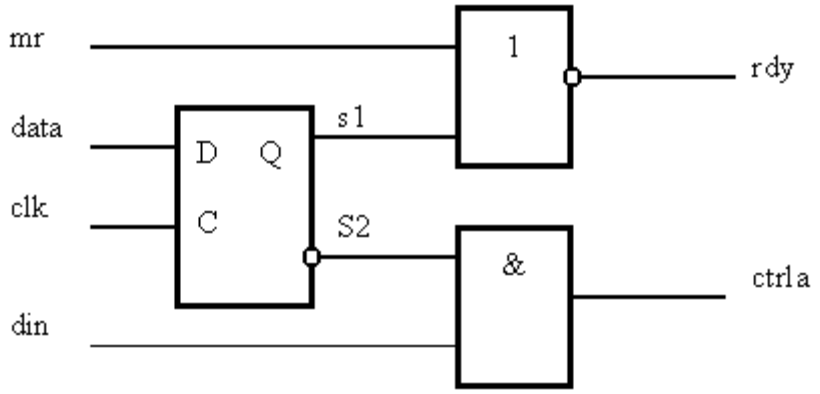


Рис. 3.8. Пристрій місцевого керування

```
entity ctr_lck is
port (data, mr, clk, din: in BIT; rdy, ctrla: out Bit);
end ctr_lck;
architecture str_view of ctr_lck is
component AND2 - декларується компонент AND2
port (x, y in BIT; z: out BIT);
end component;
component DFF - декларується компонент DFF
port (d, clock: in BIT; Q, NOTQ: out BIT);
end component;
component NOR2 - декларується компонент NOR2
port (a, b: in BIT; z: out BIT);
end component;
signal s1, s2: BIT;
begin
D1: DFF port map (data, clk, s1, s2); - вираз 1
A1: AND2 port map (s2, din, ctrla); - вираз 2
N1: NOR2 port map (s1, mr, rdy); - вираз 3
end str_view;
```

Тут декларуються три компоненти: AND2, DFF і NOR2. У тіло архітектури екземпляри компонентів вводяться за допомогою виразів 1, 2 і 3.

Компоненти пов’язані між собою сигналами s1 і s2. У декларації задається ім’я компонента і його інтерфейс, що дуже схоже на декларацію інтерфейсу пристрою. Вираз для примірника компонента повинен в першу чергу задати ім’я примірника або ярлик. Наприклад, D1 – це ярлик для примірника тригера DFF. Далі слідує список зв’язків (association list). Він встановлює зв’язок між портами компонента і портами і сигналами пристрою. Існує два варіанти зв’язку: позиційний та поіменний. У позиційному списку перший порт декларації (зліва направо) компонента відповідає першому порту примірника компонента (зліва направо), другий – другому і так далі. Саме такий варіант списку був використаний в наведеному вище прикладі. Замість цього можна було використовувати поіменний варіант списку зв’язків, наприклад, вираз 3 можна було б записати так:

```
N1: NOR2 port map (a => s1, b => mr, z => rdy);
```

Розглянемо приклад структурного опису моделі напівсуматора (рис. 3.2).

```
entity half_adder is
port (a, b: in BIT; sum, cur: out BIT);
end half_adder;
architecture h_a_str of half_adder is
component a_7486
port (a_2: in BIT;
a_3: in BIT;
a_1: out BIT);
end component;
component a_7408
port (a_2: in BIT;
a_3: in BIT;
a_1: out BIT);
end component;
begin
```

```
X1: a_7486 port map (a, b, sum);
A1: a_7408 port map (a, b, cur)
end h_a_str;
```

Назви компонентів і їх портів наведені такими, якими вони дані в бібліотеці фірми ALTERA – відомого виробника програмованих логічних схем.

Декларації компонентів, використані в архітектурі h_a_str півсуматора half_adder, можна упакувати в окремий файл, наприклад, maxplus2.vhd:

```
package maxplus2 is
component a_7486
port (a_2: in BIT;
a_3: in BIT;
a_1: out BIT);
end component;
component a_7408
port (a_2: in BIT;
a_3: in BIT;
a_1: out BIT);
end maxplus2;
```

Файл maxplus2 при компіляції може бути поміщений в бібліотеку, наприклад, з назвою altera. Посилання на бібліотеку дозволить не декларувати компоненти в архітектурі:

```
library altera;
use altera.maxplus2.all;
entity half_adder is
port (a, b: in BIT; sum, cur: out BIT);
end half_adder;
architecture h_a_str of half_adder is
begin
X1: a_7486 port map (a, b, sum);
```

A1: a_7408 port map (a, b, cur);

end h_a_str;

Комбінований опис архітектури

В єдиному тілі архітектури можна комбінувати всі розглянуті вище стилі опису [2]. В якості прикладу розглянемо однорозрядний повний суматор (рис. 3.9).

Суматор представлений виразом 1, що створює екземпляр X1 компонента a_7486 (з бібліотеки altera.maxplus2), і задає відображення портів і сигналу сутності (суматора) на порти компонента. Це структурний стиль. Вираз 2 – це process, поведінковий стиль, і вираз 3 – це стиль dataflow.

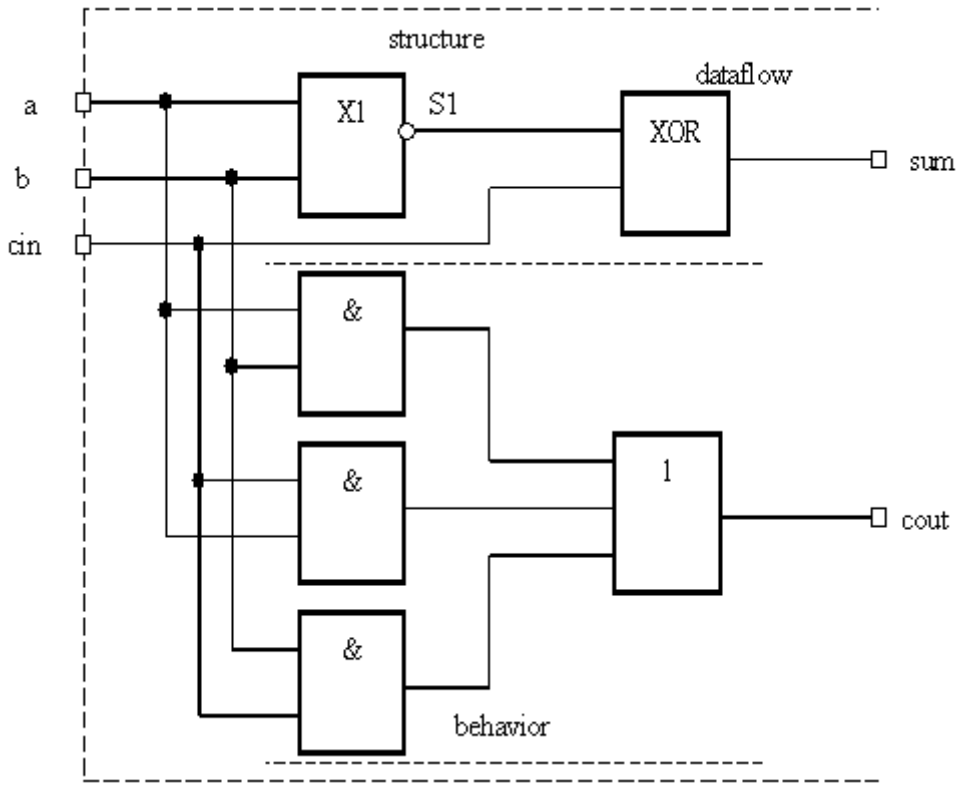


Рис. 3.9. Однорозрядний суматор

library altera;

use altera.maxplus2.all;

entity full_adder is

port (a, b, cin: in BIT; sum, cout: out BIT);

end full_adder;

```
architecture fa_mix of full_adder is
signal s1: BIT;
begin
X1: a_7486 port map (a, b, s1); - вираз 1
process (a, b, cin) - вираз 2
variable t1, t2, t3: BIT;
begin
t1: = a and b;
t2: = b and cin;
t3: = a and cin;
cout <= t1 or t2 or t3;
end process;
sum <= s1 xor cin; - вираз 3
end fa_mix;
```

Особливості синтезу схем за описами мовою VHDL

Між процесами моделювання (імітації поведінки схеми) та синтезу схем з використанням мови VHDL є істотні відмінності [2]. На відміну від системи моделювання система проектування перетворює проектну інформацію в заданий формат. Серед них виділяються широко поширений формат EDIF, спеціалізовані формати різних фірм-виробників мікросхем, а також мови Verilog, VHDL та інші. Формат EDIF є своєрідним стандартом де-факто, він є в більшості сучасних систем проектування і може використовуватися для обміну інформацією. Серед спеціалізованих форматів можна виділити формат XNF фірми Xilinx, який широко використовувався в попередніх версіях САПР цієї фірми. При перетворенні вихідного модуля в проміжний текст мовою VHDL зазвичай здійснюється перетворення в структурні конструкції цієї мови, в яких використовуються схемотехнічні особливості використовуваної елементної бази.

Система моделювання звичайно використовує всі конструкції мови VHDL, в той час, як система проектування використовує не всі його можливості. Зазвичай при синтезі не підтримуються операції над типом real (в цьому випадку при синтезі видається помилка), ігноруються ключове слово after, не підтримується також ряд інших другорядних конструкцій мови. У системах проектування атрибут event може використовуватися тільки для вказівки фронтів синхросигналів в умовних операторах, переважно в операторах if.

У сучасних системах проектування для опису схем використовується тип std_logic, який замінює тип bit. Тип std_logic має наступні значення:

0 – логічний нуль;

1 – логічна одиниця;

U – значення не ініціалізоване;

X – невідоме значення;

Z – високий вихідний опір;

W – невідоме значення при слабкому джерелі сигналу;

L – логічний нуль при слабкому джерелі сигналу;

H – логічна одиниця при слабкому джерелі сигналу;

'-' – невизначене значення.

Перелік посилань

1. Neal S. Widmer, Gregory L. Moss, Ronald J. Tocci. Digital systems. Principles and applications. Pearson, 2016. – 1004 p.

2. Robert Boylestad. Introductory Circuit Analysis. Pearson, 2016. – 1224 p.

Лекція №4

Комутатори

Комутатором (у загальному випадку на n входів та m виходів – скорочено $n \times m$) називають цифровий пристрій, який забезпечує передачу сигналів з i-го вхідного на j-ий вихідний виводи.

У цифровій схемотехніці найбільшого поширення набули такі комутуючі пристрої, як *мультиплексори, селектори-мультиплексори* (з організацією $n \times 1$) та *пристрої зсуву* ($n \times n$).

Мультиплексори

При передачі цифрової інформації виникає велика кількість ситуацій, коли цифровий двійковий код необхідно перетворити з паралельного формату в послідовний з метою його запису або передачі по двопроводній лінії. Як приклад, пристрої запису інформації – накопичувачі на компакт-дисках, гнучкі та жорсткі диски – використовують послідовний формат. З послідовним форматом працюють COM-порти ПЕОМ, модеми, USB-шини [1].

Розглянемо умови і принцип перетворення двійкового коду з паралельного формату у послідовний.

Припустимо, що по однобайтовій шині даних до пристрою підводиться двійкове слово $d_7\,d_6\,d_5\,d_4\,d_3\,d_2\,d_1\,d_0 = \text{A3}_{16}$ (рис. 4.1), яке пристрій повинен зчитати і передати в послідовному форматі на вихід Y.

Щоб виконати таку операцію, необхідно послідовно звертатись (адресуватись) до кожного біта даних і по черзі приєднувати його до виходу Y. Зрозуміло, що для забезпечення такої адресації необхідно мати три розряди адресної шини. У механічному еквіваленті адресна шина визначає короткочасне положення бігунка перемикача, який по черзі з'єднує вхід шини даних з виходом Y пристрою.

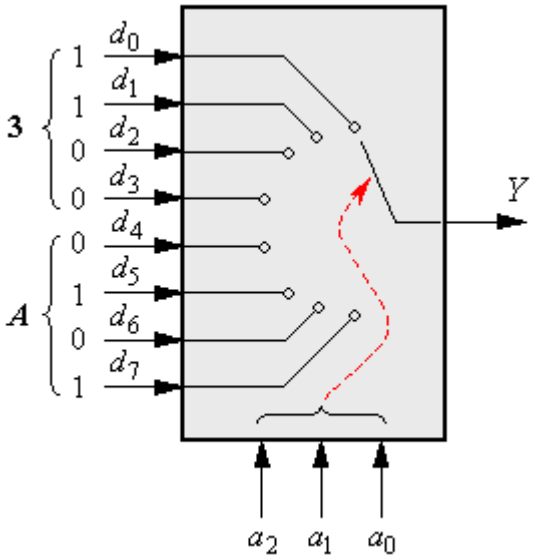


Рис. 4.1. Пристрій для перетворення паралельного формату у послідовний

Таблиця 4.1. Таблиця станів пристрою

N	a_2	a_1	a_0	Y	Код слова
0	0	0	0	d_0	1
1	0	0	1	d_1	1
2	0	1	0	d_2	0
3	0	1	1	d_3	0
4	1	0	0	d_4	0
5	1	0	1	d_5	1
6	1	1	0	d_6	0
7	1	1	1	d_7	1

Роботі пристрою, що зображений на рис. 4.1, відповідає таблиця станів (табл. 4.1).

Якщо вихідний код Y розглядати як функцію вхідних сигналів пристрою, то отримаємо наступну залежність:

$$Y = \overline{a_2}\ \overline{a_1}\ \overline{a_0}\ d_0 + \overline{a_2}\ \overline{a_1}\ a_0\ d_1 + \overline{a_2}\ a_1\ \overline{a_0}\ d_2 + \overline{a_2}\ a_1\ a_0\ d_3 +$$

$$+ a_2\ \overline{a_1}\ \overline{a_0}\ d_4 + a_2\ \overline{a_1}\ a_0\ d_5 + a_2\ a_1\ \overline{a_0}\ d_6 + a_2\ a_1\ a_0\ d_7\ .$$

Якщо зміну сигналів a_2, a_1, a_0 забезпечити послідовно в часі з тактовим інтервалом T, то можемо побудувати часову діаграму (рис. 4.2), яка встановлює послідовність передачі сигналів на входах $d_7 \ldots d_0$ на вихід Y.

Звідси витікає, що *мультиплексор* – це цифровий комбінаційний пристрій, який забезпечує перетворення інформації, що задана на входах $d_7 \ldots d_0$ у паралельному форматі, в послідовний формат на виході Y.

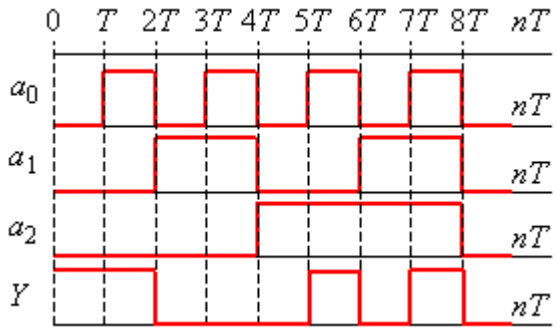


Рис. 4.2. Часова діаграма, яка встановлює послідовність передачі сигналів на входах $d_7 \ldots d_0$ на вихід Y

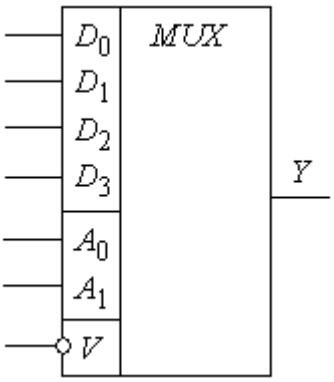


Рис. 4.3. Умовне позначення мультиплексора

Умовне позначення мультиплексора в електронних схемах приводиться на рис. 4.3. Входи A називають адресними (адресна шина), а входи D – входами даних (шина даних). Допоміжний вхід V називають керуючим або *дозволяючим*. З його урахуванням робота мультиплексора, приведеного на рис. 4.3, описуватиметься наступною логічною функцією:

$$Y = \overline{v} \cdot (\overline{a_1}\ \overline{a_0}\ d_0 + \overline{a_1}\ a_0\ d_1 + a_1\ \overline{a_0}\ d_2 + a_1\ a_0\ d_3)\ . \qquad (4.1)$$

Дозволяючий вхід V розширює можливості пристрою. Він дає можливість синхронізувати роботу мультиплексора з іншими схемами, а також використовується для нарощування розрядності адресних сигналів. Логічна схема мультиплексора, реалізована у базисі елементів **І-АБО-НІ** у відповідності до формули (4.1), приведена на рис. 4.4.

Кількість входів шини даних мультиплексора відображається у його назві. Наприклад, розглянута схема відповідає назві “з чотирьох в один”, або просто “**4:1**”. В залежності від кількості інформаційних входів, вони бувають “**8:1**”, “**16:1**” і т. д.

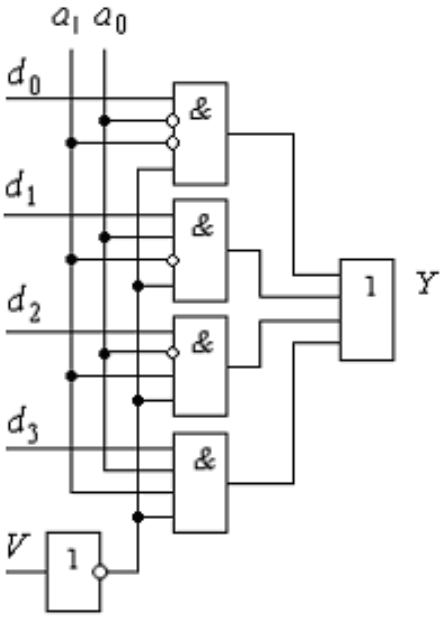


Рис. 4.4. Логічна схема мультиплексора, реалізована у базисі елементів І-АБО-НІ

Оскільки мультиплексор є пристроєм логіки, то на його базі можуть реалізовуватись різні логічні функції. Як приклад, скористаємось формулою (4.1). Припустимо, що необхідно реалізувати логічну функцію:

$$y = x_0 \cdot x_1 + \overline{x_0} \cdot \overline{x_1}\,. \qquad (4.2)$$

Порівнюючи формули (4.1) і (4.2), бачимо, що, присвоївши адресним сигналам a_0, a_1 значення x_0, x_1 відповідно і забезпечивши виконання умов $d_1 = d_2 = 0$, $d_0 = d_3 = 1$, $v = 0$, одержимо схему пристрою (рис. 4.5), який реалізує функцію (4.2).

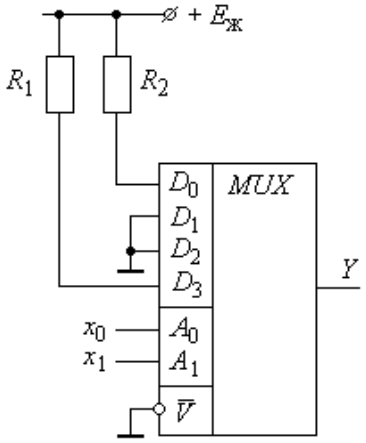


Рис. 4.5. Схема пристрою, який реалізує функцію (4.2)

Нескладні перетворення логічних функцій дають можливість реалізовувати на мультиплексорах функції з кількістю змінних, що перевершує кількість адресних входів. Виконується наступна послідовність перетворень:

- у мінімізованій диз'юнктивній нормальній формі функції виділяються змінні, які мають найвищий ранг, тобто повторюються в найбільшій кількості диз'юнкцій. Наприклад, вибирається *m* змінних;
- виконується перетворення функції так, щоб забезпечити присутність виділених змінних в усіх диз'юнкціях. Для цього кожна диз'юнкція домножається на $1 = x + \bar{x}$;
- проводиться перегрупування змінних логічної функції таким чином, щоб виділені змінні були винесені за дужки у відповідних групах диз'юнкцій.

У результаті виконаних перетворень одержується формула вихідної функції, яка реалізується на мультиплексорі "з $M = 2^m$ в 1", якщо на його адресні входи подати виділені змінні, а на інформаційних входах забезпечується виконання тих логічних виразів, які залишились в дужках. Якщо вирази в дужках є функціями декількох змінних, то їх реалізація може бути забезпечена або за допомогою іншого мультиплексора, або на основі типових логічних елементів.

Приклад 4.1. Функцію п'яти змінних: $y = x_0 \cdot x_1 \cdot \bar{x}_3 \cdot \bar{x}_4 + \bar{x}_0 \cdot \bar{x}_3 \cdot x_4 + x_0 \cdot \bar{x}_2 \cdot x_3 + x_2 \cdot \bar{x}_3 + \bar{x}_1 \cdot x_2$ реалізувати з використанням мультиплексора **"з 4-х в 1"**.

Розв'язання. У використовуваному мультиплексорі наявні два адресні входи – отже, вибираємо дві змінні з найбільшими рангами. Ранги змінних приведені в табл. 4.2.

Таблиця 4.2. Ранги змінних

x_0	x_1	x_2	x_3	x_4
3	2	3	4	2

Вибираємо змінні x_2 та x_3. Виконуємо перетворення функції:

$$y = x_0 \cdot x_1 \cdot \bar{x}_3 \cdot \bar{x}_4 \cdot (x_2 + \bar{x}_2) + \bar{x}_0 \cdot \bar{x}_3 \cdot x_4 \cdot (x_2 + \bar{x}_2) + x_0 \cdot \bar{x}_2 \cdot x_3 + x_2 \cdot \bar{x}_3 + \bar{x}_1 \cdot x_2 \cdot (x_3 + \bar{x}_3) =$$
$$= \bar{x}_2 \cdot \bar{x}_3 \cdot (x_0 \cdot x_1 \cdot \bar{x}_4 + \bar{x}_0 \cdot x_4) + \bar{x}_2 \cdot x_3 \cdot (x_0) + x_2 \cdot x_3 \cdot (\bar{x}_1) + x_2 \cdot \bar{x}_3 \cdot (x_0 \cdot x_1 \cdot x_4 + \bar{x}_0 \cdot x_4 + 1 + \bar{x}_1) =$$
$$= \bar{x}_2 \cdot \bar{x}_3 \cdot (x_0 \cdot x_1 \cdot \bar{x}_4 + \bar{x}_0 \cdot x_4) + \bar{x}_2 \cdot x_3 \cdot (x_0) + x_2 \cdot x_3 \cdot (\bar{x}_1) + x_2 \cdot \bar{x}_3 \cdot (1)$$

Логічна сума в перших дужках також реалізується за допомогою мультиплексора **4:1**, якщо на адресні входи його подати змінні x_0 та x_4.

Принципова схема пристрою, що реалізує потрібну логічну функцію, приведена на рис. 4.6.

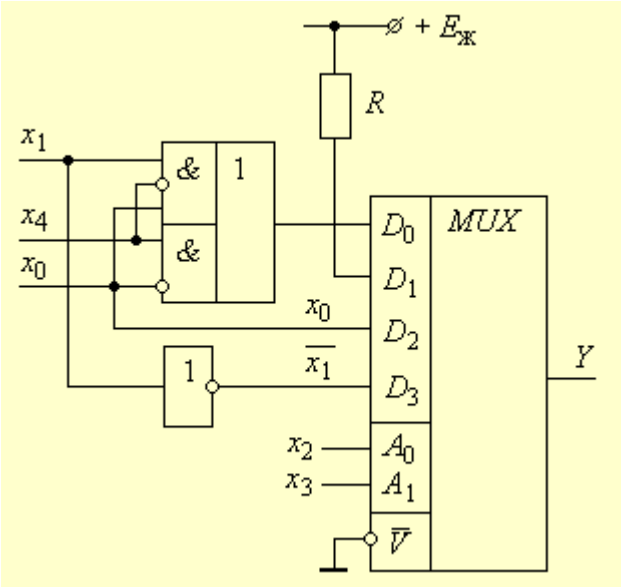


Рис. 4.6. Принципова схема пристрою, що реалізує потрібну логічну функцію

Практично задача може бути розв'язана при використанні мультиплексора **8:1**.

Вибравши логічні змінні x_0, x_2, x_3 як змінні з найвищими рангами (див. табл. 4.2), функцію перетворимо до наступного вигляду:

$$y = \bar{x}_0 \cdot \bar{x}_2 \cdot \bar{x}_3 \cdot (x_4) + x_0 \cdot \bar{x}_2 \cdot \bar{x}_3 \cdot (x_1 \bar{x}_4) + \bar{x}_0 \cdot x_2 \cdot \bar{x}_3 \cdot (1) +$$
$$+ x_0 \cdot x_2 \cdot \bar{x}_3 \cdot (1) + \bar{x}_0 \cdot \bar{x}_2 \cdot x_3 \cdot (0) + x_0 \cdot \bar{x}_2 \cdot x_3 \cdot (1) + \bar{x}_0 \cdot x_2 \cdot x_3 \cdot (\bar{x}_1) + x_0 \cdot x_2 \cdot x_3 \cdot (\bar{x}_1).$$

У табл. 4.3 зведені значення необхідних сигналів, які подаються на відповідні інформаційні входи.

На рис. 4.7 приведена схема, що реалізує необхідну функцію на мультиплексорі К1533КП7 (його зарубіжні аналоги – мікросхеми MC54151W,

SN74151N). Необхідні допоміжні логічні функції реалізуються з використанням одного корпусу мікросхеми К1533ЛА3 (зарубіжні аналоги – 7400PC, MH7400, SN7400N).

Табл. 4.3. Значення необхідних сигналів, які подаються на відповідні інформаційні входи

N	x_3	x_2	x_0	D_i
0	0	0	0	x_4
1	0	0	1	$x_1 \cdot \overline{x_4}$
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	$\overline{x_1}$
7	1	1	1	$\overline{x_1}$

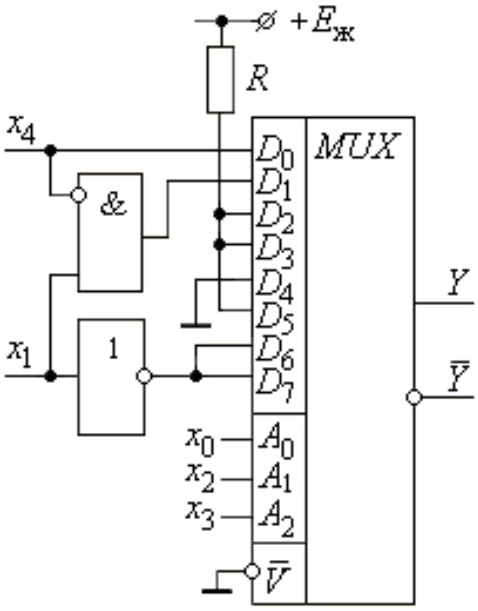


Рис. 4.7. Схема, що реалізує необхідну функцію на мультиплексорі К1533КП7

У більшості серій мікросхем мультиплексори, що виготовляються у вигляді самостійних виробів, мають кількість інформаційних входів, не перевищуючу шістнадцяти. Збільшення їх кількості досягається різними прийомами з'єднання окремих мультиплексорів. Наприклад, на рис. 4.8 приведена схема паралельного нарощування.

Для збільшення кількості інформаційних входів необхідно підвищувати кількість адресних провідників (ємність адресного простору, адресної шини). У схемі, приведеній на рис. 4.8, розширення адресного простору досягається використанням входу V, який, об'єднаний в обох мультиплексорах через інвертор DD3, виступає старшим розрядом адресної шини. При $a_2 = 0$ зміна сигналів на a_0, a_1 дозволить вибирати входи $d_0 \ldots d_3$ шини даних і через елемент

АБО (DD4) передавати на вихід *Y*. Мультиплексор DD2 при цьому відключений високим рівнем сигналу на виході DD3. При $a_2 = 1$ зміною a_0, a_1 забезпечується комутація входів $d_4 \ldots d_7$ на вихід *Y*. При наявності в мультиплексорах інверсних виходів їх об'єднання забезпечується елементом **І-НІ** (DD5).

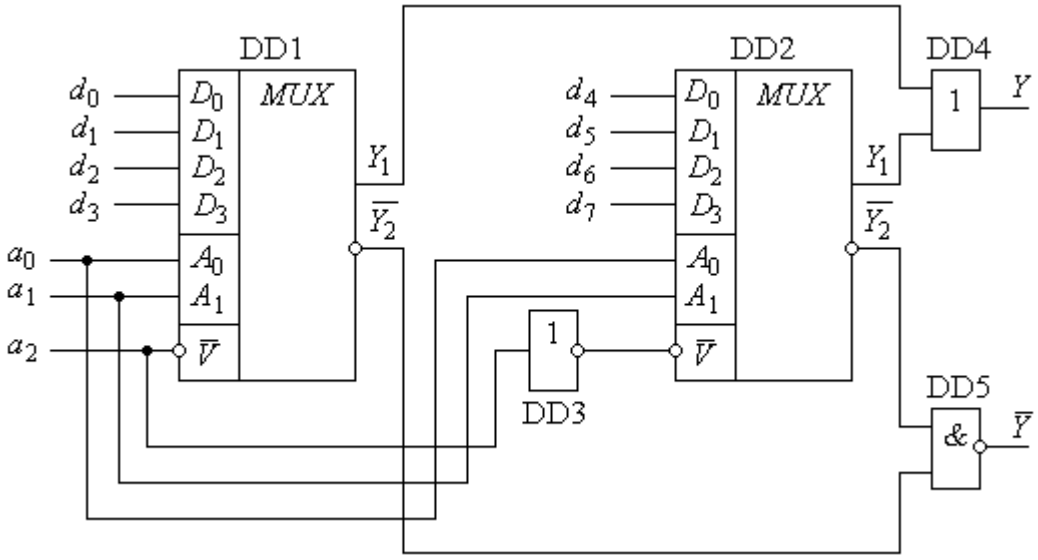


Рис. 4.8. Схема паралельного нарощування

При необхідності суттєвого нарощування кількості комутованих інформаційних входів використовується пірамідальний спосіб нарощування. Приклад його реалізації приведений на рис. 4.9. Ця схема в реалізації є більш гнучкою. Адресні входи мультиплексорів нижнього рівня DD1…DD4 з'єднуються паралельно і керуються молодшими розрядами адресного простору. Їх кількість визначається тільки кількістю інформаційних входів кожного мультиплексора. Мультиплексор верхнього ступеня задає старші розряди адресного простору, завдяки яким він забезпечує комутацію виходу кожного з мультиплексорів $Y_0 \ldots Y_3$ на загальний вихід *Y*.

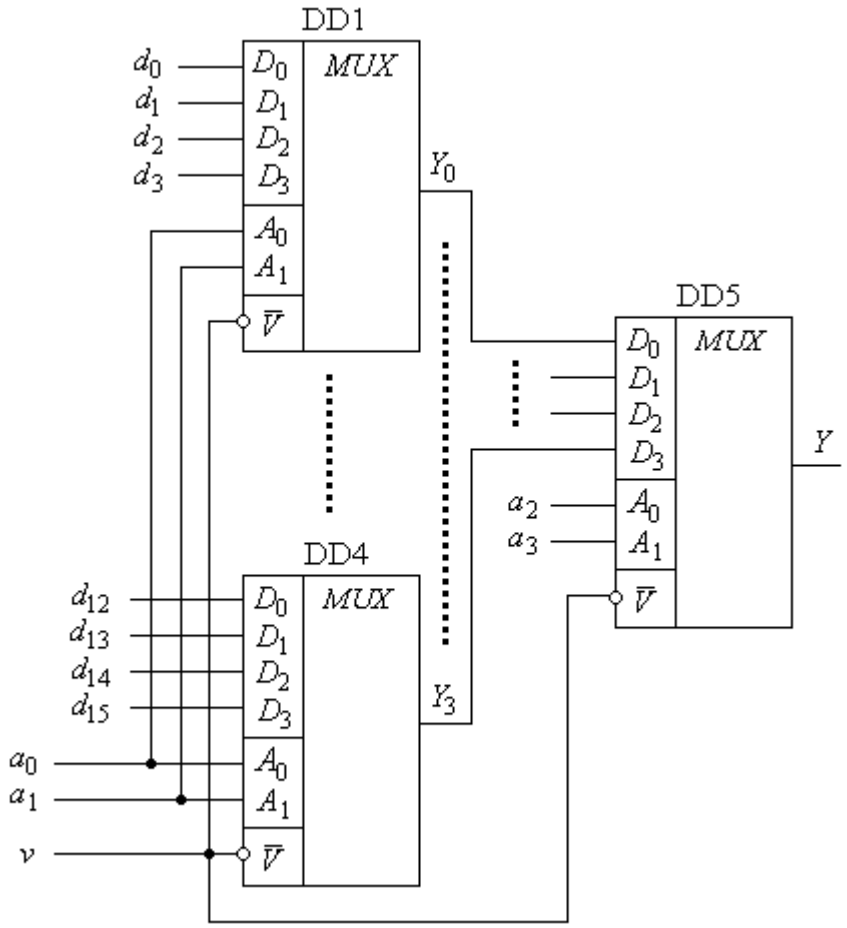


Рис. 4.9. Схема пірамідального нарощування

Пристрої зсуву

Пристрої зсуву – це комбінаційні пристрої, призначені для забезпечення передачі вхідного коду на вихід зі зсувом на декілька розрядів у залежності від керуючого коду [1]. Забезпечення зсуву двійкових слів необхідно при побудові арифметичних пристроїв, у задачах узгодження шин даних при формуванні заданих часових послідовностей сигналів та ін.

Припустимо, що необхідно спроектувати пристрій, на вхід якого подаються дані по лініях $d_3\, d_2\, d_1\, d_0$, котрі необхідно передавати на вихідні лінії $y_3\, y_2\, y_1\, y_0$ зі зсувом від **0** до **3** розрядів у залежності від значення адресного двійкового коду A (рис. 4.10). Оскільки при одночасному виконанні зсуву можливі лише чотири комбінації, то для їх забезпечення достатньо мати лише дві нитки адресного слова $A = a_1\, a_0$.

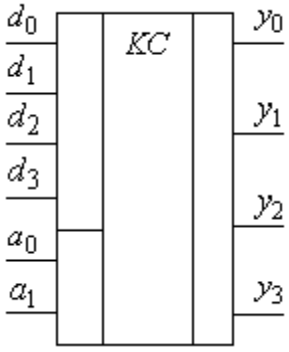


Рис. 4.10. Схема пристрою, на вхід якого подаються дані, котрі необхідно передавати на вихідні лінії в залежності від значення адресного двійкового коду

Тоді функціональні залежності для кожного з виходів y_i матимуть наступний вигляд:

$$\begin{aligned} y_0 &= d_0 \overline{a_1}\,\overline{a_0} + d_1 \overline{a_1}\, a_0 + d_2\, a_1 \overline{a_0} + d_3\, a_1\, a_0; \\ y_1 &= d_1 \overline{a_1}\,\overline{a_0} + d_2 \overline{a_1}\, a_0 + d_3\, a_1 \overline{a_0} + d_0\, a_1\, a_0; \\ y_2 &= d_2 \overline{a_1}\,\overline{a_0} + d_3 \overline{a_1}\, a_0 + d_0\, a_1 \overline{a_0} + d_1\, a_1\, a_2; \\ y_3 &= d_3 \overline{a_1}\,\overline{a_0} + d_0 \overline{a_1}\, a_0 + d_1\, a_1 \overline{a_0} + d_2\, a_1\, a_2. \end{aligned} \qquad (4.3)$$

З формул (4.3) витікає, що схема зсуву складається з чотирьох мультиплексорів, адресні входи яких з’єднані паралельно, а розряди шини даних приєднані до загальної шини даних зі зсувом одна відносно іншої на один розряд (рис. 4.11).

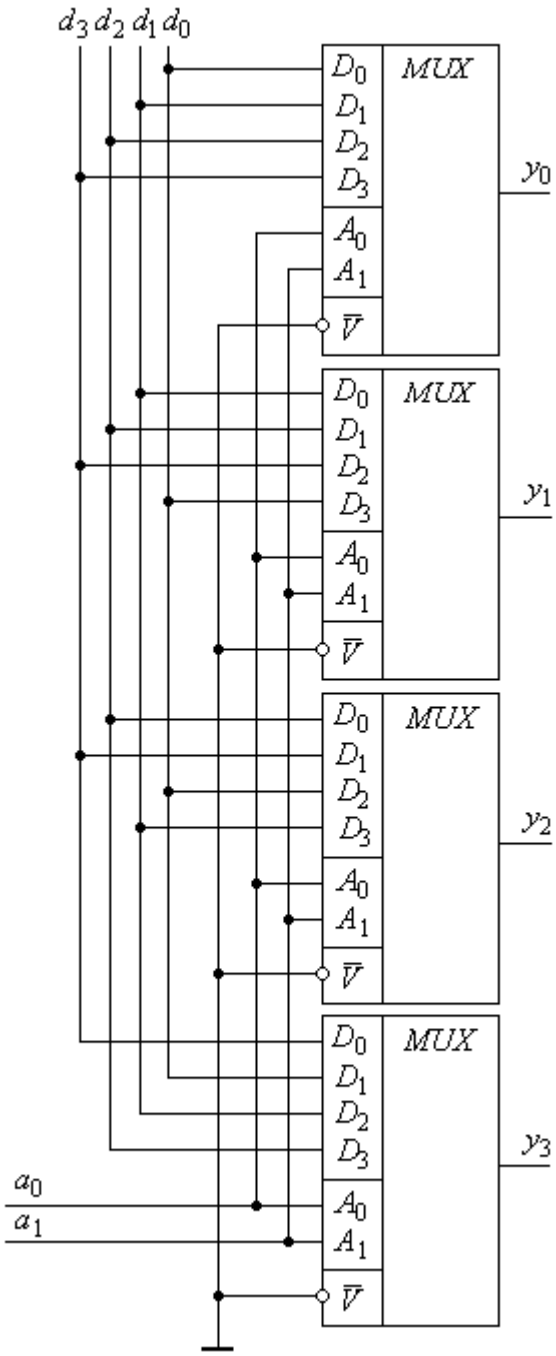


Рис. 4.11. Схема зсуву, яка складається з чотирьох мультиплексорів

Селектори-мультиплексори

Селектори-мультиплексори – прилади, які дають можливість забезпечувати не тільки побітну цифрову передачу інформації, а й вибирати цифрові потоки інформації з будь-якого входу на вихід [1]. Ще один приклад використання таких приладів – це застосування їх у якості демультиплексорів. Робота у такому режимі полягає в тому, що двійковий цифровий код може

подаватись на вхід *Y*, а зніматись побітово з виходів $d_0 \div d_3$, забезпечуючи тим самим перетворення послідовного формату в паралельний.

Кодуючі та декодуючі пристрої

Найпростішим перетворювачем кодів можна вважати кодер, який, незалежно від коду, що подається на його вхід (прямий чи зворотний), на виході завжди видає прямий код (рис. 4.12) [1].

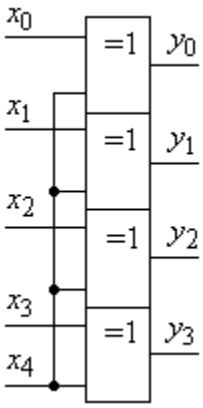


Рис. 4.12. Найпростіший перетворювач кодів

Схема в кожному біті перетворювача реалізує операцію:

$$y_i = x_i \cdot \overline{x_4} + \overline{x_i} \cdot x_4,$$

у зв'язку з чим при наявності у знаковому розряді x_4 одиниці (від'ємне число) код порозрядно інвертується, а при $x_4 = 0$ передається без змін.

У багатьох серіях такий перетворювач може бути реалізований на мікросхемі ЛП5.

Розглянемо декілька прикладів розробки перетворювачів кодів.

Приклад 4.2. Створити перетворювач двійкового коду в прямий код семисегментних індикаторів для відображення цифр десяткової системи числення.

Розв'язання**.** Умовне розміщення світлодіодів матриці семисегментного індикатора наведене на рис. 4.13.

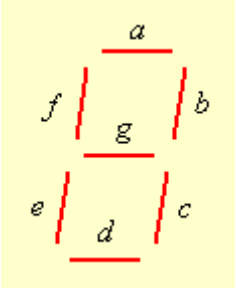


Рис. 4.13. Умовне розміщення світлодіодів матриці семисегментного індикатора

Таблиця 4.4. Таблиця істинності перетворювача

N	x_3	x_2	x_1	x_0	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

Відповідно до нього, створюється таблиця істинності перетворювача (табл. 4.4), приписуючи значення логічної **“1”** тому світлодіодові, що засвічується при відображенні цифри, та логічного **“0”** – у протилежному випадку.

Використовуючи табл. 4.4, можемо записати логічні функції роботи кожного світлодіода у ДДНФ:

$$a = \vee\, 0, 2, 3, 5, 6, 7, 8, 9 = \wedge 1, 4\,;$$

$$b = \vee\, 0, 1, 2, 3, 4, 7, 8, 9 = \wedge 5, 6\,;$$

$$c = \vee\, 0, 1, 3, 4, 5, 6, 7, 8, 9 = \wedge 2\,;$$

$$d = \vee\, 0, 2, 3, 5, 6, 8, 9 = \wedge 1, 4, 7\,;$$

$$e = \vee\, 0, 2, 6, 8 = \wedge 1, 3, 4, 5, 7, 9\,;$$

$$f = \vee\, 0, 4, 5, 6, 8, 9 = \wedge 1, 2, 3, 7\,;$$

$$g = \vee\, 2, 3, 4, 5, 6, 8, 9 = \wedge 0, 1, 7\,.$$

Даний перетворювач кодів може бути реалізований в базисі ЛЕ **І-АБО-НІ**.

Проектованим комбінаційним пристроєм не використовуються комбінації вхідних сигналів від **1010** до **1111**, яким відповідають десяткові числа від **10** до **15**, тобто логічні функції, які описують роботу світлодіодів семисегментного індикатора, є неповністю визначеними. Довизначивши їх одиницями і скориставшись даними табл. 4.4, представляємо кожну логічну функцію у вигляді карти Карно (рис. 4.14) та мінімізуємо їх.

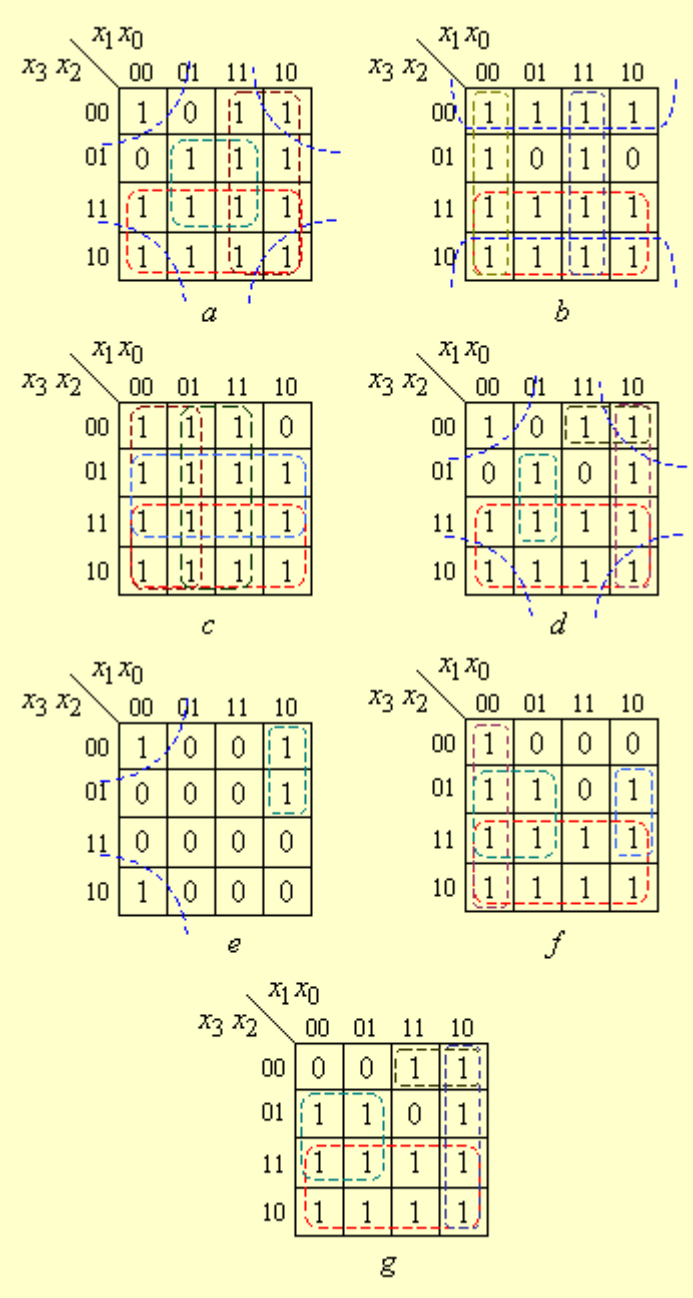


Рис. 4.14. Карти Карно логічних функцій семисегментного індикатора

Після мінімізації знаходимо:

$$a = x_1 + x_3 + \overline{x_2} \cdot \overline{x_0} + x_2 \cdot x_0$$
$$b = \overline{x_2} + x_3 + \overline{x_1} \cdot \overline{x_0} + x_1 \cdot x_0$$
$$c = x_3 + x_2 + \overline{x_1} + x_0$$
$$d = x_3 + \overline{x_2} \cdot \overline{x_0} + x_1 \cdot \overline{x_0} + \overline{x_3} \cdot \overline{x_2} \cdot x_1 + x_2 \cdot \overline{x_1} \cdot x_0$$
$$e = \overline{x_2} \cdot \overline{x_1} \cdot \overline{x_0} + \overline{x_3} \cdot x_1 \cdot \overline{x_0}$$
$$f = x_3 + \overline{x_1} \cdot \overline{x_0} + x_2 \cdot \overline{x_1} + x_2 \cdot x_1 \cdot \overline{x_0}$$
$$g = x_3 + x_1 \cdot \overline{x_0} + x_2 \cdot \overline{x_1} + \overline{x_3} \cdot \overline{x_2} \cdot x_1$$

Табл. 4.4 відповідає логіці роботи дешифратора К564ИД5 (CD4056BE, HCC4056BD, TC4056BF). Умовне графічне позначення дешифратора К564ИД5 приведено на рис. 4.15.

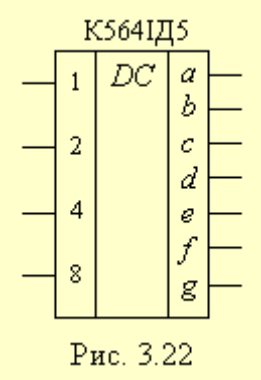


Рис. 4.15. Умовне графічне позначення дешифратора К564ИД5

Для побудови перетворювачів кодів перш за все необхідно мати таблицю істинності його роботи, яка складається з двох частин. Перша з них – це таблиця перебору змінних перетворюваного коду, а друга – це таблиця відповідності змінних отримуваного коду.

Приклад 4.3. Розробити логічну схему перетворювача прямого двійкового коду в доповнюючий.

Розв’язання. Доповнюючий код отримується інвертуванням кожного розряду прямого двійкового коду і додаванням до отриманого результату одиниці шляхом двійкової арифметики.

Карти Карно для вихідних функцій перетворювача прямого двійкового коду в доповнюючий приведені на рис. 4.16.

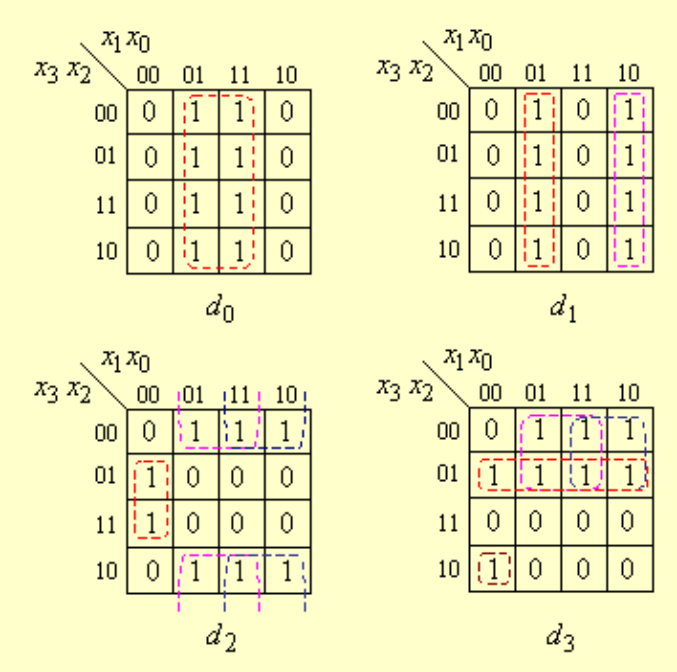


Рис. 4.16. Карти Карно для вихідних функцій перетворювача прямого двійкового коду в доповнюючий

Мінімізовані значення логічних функцій:

$$d_0 = x_0 = 0 \oplus x_0 \ ;$$

$$d_1 = \overline{x_0} \cdot x_1 + x_0 \cdot \overline{x_1} = x_0 \oplus x_1 \ ;$$

$$d_2 = x_0 \cdot \overline{x_2} + x_1 \cdot \overline{x_2} + \overline{x_0} \cdot \overline{x_1} \cdot x_2 = (x_0 + x_1) \oplus x_2 \ ;$$

$$d_3 = x_2 \cdot \overline{x_3} + x_0 \cdot \overline{x_3} + x_1 \cdot \overline{x_3} + \overline{x_0} \cdot \overline{x_1} \cdot \overline{x_2} \cdot x_3 = (x_0 + x_1 + x_2) \oplus x_3 \ .$$

З аналізу приведених формул можна знайти загальний вигляд функції для *i*-го розряду вихідної функції:

$$d_i = (x_0 + x_1 + \ldots + x_{i-1}) \oplus x_i .$$

Як витікає з одержаних формул, з підвищенням розрядності формула для знаходження вихідної функції d_i значно ускладнюється, і її використання фактично стає недоцільним. Тому для знаходження розрядів доповнюючого коду більш економним для багаторозрядних чисел виявляється перетворення їх у зворотний код з послідуючим додаванням одиниці до молодшого розряду за допомогою суматора.

У залежності від характеру перетворення інформації, ці операції називають *шифруванням* або *дешифруванням* (широко використовується також

термінологія *кодування* та *декодування*). Шифрування полягає у перетворенні m-розрядного коду з k_m байдужих наборів вхідних змінних в однозначно відповідний йому n-розрядний код з меншим числом розрядів ($n < m$) і байдужих наборів. Під байдужими наборами в будь-якому коді маються на увазі ті набори, які не використовуються при перетворенні. Пристрої, які виконують вказані операції, називаються відповідно *шифраторами* та *дешифраторами* (*кодерами* та *декодерами*).

Приклад 4.4. Розробити схему шифратора для перетворення десяткового коду з клавіатури калькулятора в відповідний йому двійково-десятковий код.

Розв'язання. Приймемо, що при натисканні відповідної клавіші (0...9) низький рівень відповідного сигналу перетвориться в чотирьохрозрядний двійковий код від **0000** до **1001**. Таблиця шифратора "з 10 в 4" (і, відповідно, дешифратора "з 4 в 10") приводиться нижче (табл. 4.5). З табл. 4.5 витікає, що для кожної клавіші вхідного коду характерною є лише одна з можливих комбінацій **1** та **0**. До решти комбінацій проектований кодер повинен бути байдужим.

Аналогічно, у вихідних комбінаціях значень $y_3\,y_2\,y_1\,y_0$ не використовуються набори від **1010** до **1111**.

Таблиця 4.4. Таблиця істинності шифратора

N	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	y_3	y_2	y_1	y_0
0	0	1	1	1	1	1	1	1	1	1	0	0	0	0
1	1	0	1	1	1	1	1	1	1	1	0	0	0	1
2	1	1	0	1	1	1	1	1	1	1	0	0	1	0
3	1	1	1	0	1	1	1	1	1	1	0	0	1	1
4	1	1	1	1	0	1	1	1	1	1	0	1	0	0
5	1	1	1	1	1	0	1	1	1	1	0	1	0	1
6	1	1	1	1	1	1	0	1	1	1	0	1	1	0
7	1	1	1	1	1	1	1	0	1	1	0	1	1	1
8	1	1	1	1	1	1	1	1	0	1	1	0	0	0
9	1	1	1	1	1	1	1	1	1	0	1	0	0	1

Реалізація такого перетворювача може бути виконана з використанням формул:

$$y_0 = \overline{x_1} + \overline{x_3} + \overline{x_5} + \overline{x_7} + \overline{x_9};$$
$$y_1 = \overline{x_2} + \overline{x_3} + \overline{x_6} + \overline{x_7};$$
$$y_2 = \overline{x_4} + \overline{x_5} + \overline{x_6} + \overline{x_7};$$
$$y_3 = \overline{x_8} + \overline{x_9}.$$

Табл. 4.5 близька до таблиці станів шифратора К555ИВ3 з тією лише різницею, що виходи останнього $y_0 \ldots y_3$ – інвертовані.

До складу кодуючих пристроїв часто входить пристрій для виділення старшої одиниці. Робота цього пристрою полягає в тому, що одиниця на виході ***i***-го розряду при наявності одиниці на його вході можлива лише тоді, коли на входах всіх старших розрядів мають місце нулі. Тобто для чотирьохрозрядного перетворювача слова $x_3\, x_2\, x_1\, x_0$ в $y_3\, y_2\, y_1\, y_0$ повинні бути справедливими наступні співвідношення:

$$y_3 = x_3;$$
$$y_2 = x_2 \cdot \overline{x_3};$$
$$y_1 = x_1 \cdot \overline{x_2} \cdot \overline{x_3};$$
$$y_0 = x_0 \cdot \overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3}.$$

Для організації такого перетворювача як модуля з можливістю нарощування розрядності його доповнюють вхідним дозволяючим сигналом ***e*** і вихідним дозволяючим сигналом ***q***.

Технічна реалізація модуля на базі ТТЛ-елементів **І-НІ** приведена на рис. 4.17.

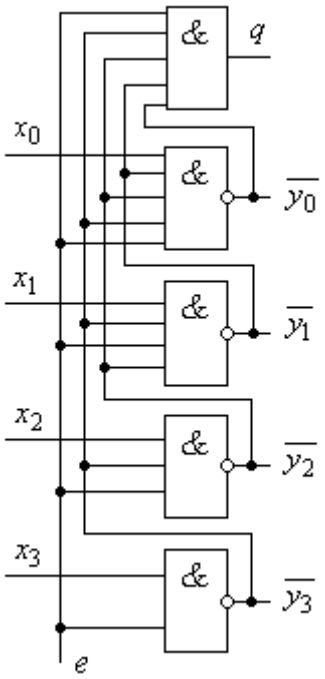


Рис. 4.17. Технічна реалізація пристрою для виділення старшої одиниці

Логіка роботи схеми описується формулами:

$$y_3 = x_3 \cdot e$$

$$y_2 = x_2 \cdot e \cdot \overline{y_3} = e \cdot x_2 \cdot \overline{x_3}$$

$$y_1 = x_1 \cdot e \cdot \overline{y_2} \cdot \overline{y_3} = e \cdot x_1 \cdot \overline{x_2} \cdot \overline{x_3}$$

$$y_0 = x_0 \cdot e \cdot \overline{y_1} \cdot \overline{y_2} \cdot \overline{y_3} = e \cdot x_0 \cdot \overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3}$$

$$q = e \cdot \overline{x_0} \cdot \overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3}$$

Якщо використовуються слова, розрядність яких перевищує розрядність модуля, то вони розбиваються на групи, які поєднуються завдяки виходу ***q*** старшої групи і входу ***e*** молодшої. Підключення до виходу приведеної схеми шифратора, що відповідає, наприклад, табл. 3.6, дає можливість створити схему, яка перетворює біт старшої одиниці у відповідний двійковий код. Іншими словами, така схема визначатиме у двійковому коді номер біту найстаршої одиниці.

Пріоритетні шифратори знаходять використання для визначення номеру пристрою, що подає сигнал запиту на обслуговування в мікропроцесорних системах, визначення станів та номерів вимикачів у схемах автоматики та інше.

Дешифратори широко використовуються для організації звернення до тих чи інших пристроїв – наприклад, до конкретних пристроїв пам'яті.

Дешифратори-демультиплексори

Демультиплексори – це цифрові комбінаційні пристрої, в яких сигнали з одного інформаційного входу x розподіляються на 2^m виходів y_i, які комутуються m адресними входами, тобто фактично вони є генераторами мінтермів [1]. Завдяки цій властивості легко створювати канонічні суми, тобто реалізовувати за допомогою логічних операцій **АБО** логічні функції вхідних змінних.

Якщо прийняти кількість адресних входів $m = 2$ (рис. 4.18), то роботу пристрою описуватимуть булеві рівняння (4.4):

$$\begin{aligned} y_0 &= \bar{v} \cdot x \cdot \overline{a_0} \cdot \overline{a_1} \\ y_1 &= \bar{v} \cdot x \cdot a_0 \cdot \overline{a_1} \\ y_2 &= \bar{v} \cdot x \cdot \overline{a_0} \cdot a_1 \\ y_3 &= \bar{v} \cdot x \cdot a_0 \cdot a_1 \end{aligned} \quad . \qquad (4.4)$$

Прикладом табличної форми запису роботи демультиплексора, згідно з рівняннями (4.4), є табл. 4.5.

Таблиця 4.5. Таблична форма запису роботи демультиплексора

V	A_1	A_0	x	y_0	y_1	y_2	y_3
0	0	0	1/0	1/0	0	0	0
0	0	1	1/0	0	1/0	0	0
0	1	0	1/0	0	0	1/0	0
0	1	1	1/0	0	0	0	1/0
1	0	0	x	$\bar{V}$	0	0	0
1	0	1	x	0	$\bar{V}$	0	0
1	1	0	x	0	0	$\bar{V}$	0
1	1	1	x	0	0	0	$\bar{V}$

Особливістю більшості серійних мікросхем є їх багатофункціональність. Приведена на рис. 4.18 мікросхема DD1 дешифратора при $a_3 = 0$ виконуватиме функцію перетворювача трьохрозрядного двійкового коду у вісімковий. Значення $a_3 = 1$ при цьому буде заборонено. Якщо на вхід a_3 подавати

послідовний двійковий код, а входи a_0, a_1, a_2 використовувати як адресні, то одержимо демультиплексор **“з 1 на 8”**.

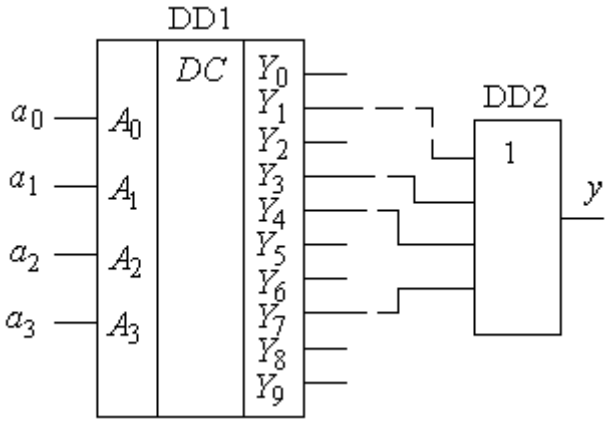


Рис. 4.18. Мікросхема дешифратора

Приклад 4.5. Використовуючи дешифратор К564ИД1, розробити пристрій для перетворення двійкового однобайтного коду, заданого в послідовному форматі, в код, представлений у паралельному форматі.

Розв’язання. Дешифратор К564ИД1 – це перетворювач чотирьохрозрядного двійкового коду в прямий десятковий. Він має чотири входи розрядів двійкового коду (1, 2, 4, 8) і 10 виходів розрядів десяткового коду (0 ÷ 9) (рис. 4.19). Табл. 4.6 є таблицею істинності мікросхеми і пояснює її роботу.

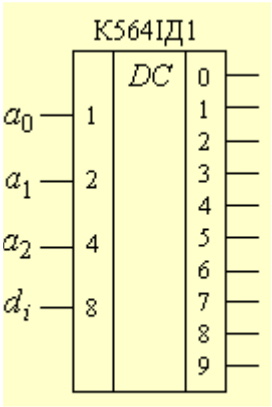


Рис. 4.19. Дешифратор К564ИД1

Таблиця 4.6. Таблиця істинності мікросхеми

Вхідний код Входи				Вихідний код Виходи									
8	**4**	**2**	**1**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1
1	0	1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0

Для використання дешифратора К564ИД1 у якості демультиплексора входи 1, 2, 4 необхідно використовувати як адресні a_0, a_1, a_2, а вхід 8 як вхід даних послідовного формату. На вихідних шинах 0 ÷ 7 отримуватимемо рознесені в часі значення двійкового коду, який подається на вхід 8 в інвертованому виді.

На рис. 4.20 зображені часові діаграми роботи пристрою при подачі на вхід 8 послідовного двійкового коду d_i.

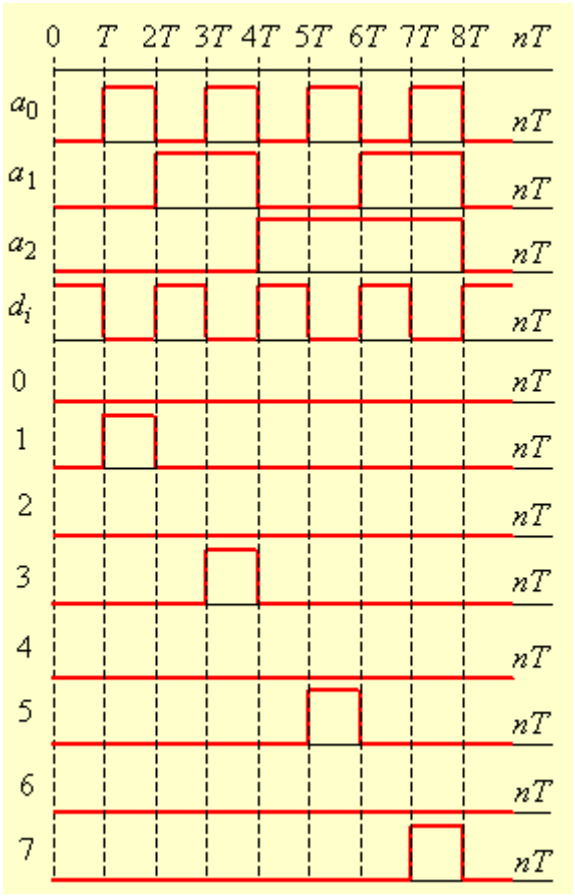


Рис. 4.20. Часові діаграми роботи пристрою при подачі на вхід 8 послідовного двійкового коду d_i

Дешифратори широко використовуються для реалізації логічних функцій. Наприклад, для реалізації функції:

$$y = \vee 1,3,4,7$$

достатньо об'єднати виходи y_1, y_3, y_4, y_7 за допомогою чотирьохвходового елемента **АБО** (DD2 на рис. 4.18). Якщо виходи $y_0 \div y_{n-1}$ дешифратора інвертовані, то об'єднання виходів відповідних диз'юнкцій забезпечується елементами, що реалізують операцію **І-НІ**.

Використання дешифраторів як генераторів мінтермів корисно в тих випадках, коли необхідно реалізовувати систему логічних функцій. При використанні дешифраторів для реалізації логічних функцій відпадає необхідність їх мінімізації. Більш того, їх необхідно приводити до досконалої диз'юнктивної форми.

Приклад 4.6. Комбінаційна схема визначена за допомогою наступних рівнянь:

$$y_1 = x_1\, x_2 + \bar{x}_1\, \bar{x}_2\, \bar{x}_3\,;$$

$$y_2 = x_1 + x_2 + \bar{x}_3\,;$$

$$y_3 = \bar{x}_1\, x_2 + x_1\, \bar{x}_2\, x_3\,.$$

Розробити схему, що реалізує ці три рівняння за допомогою дешифратора та зовнішніх логічних елементів.

Розв'язання. Для вирішення цієї задачі перетворимо функцію до вигляду:

$$\bar{y}_1 = \overline{x_1\, x_2\, x_3 + x_1\, x_2\, \bar{x}_3 + \bar{x}_1\, \bar{x}_2\, \bar{x}_3} = \overline{x_1\, x_2\, x_3} \cdot \overline{x_1\, x_2\, \bar{x}_3} \cdot \overline{\bar{x}_1\, \bar{x}_2\, \bar{x}_3}\,;$$

$$\bar{y}_2 = \overline{x_1 + x_2 + \bar{x}_3} = \bar{x}_1\, \bar{x}_2\, x_3\,;$$

$$\bar{y}_3 = \overline{\bar{x}_1\, x_2\, x_3} \cdot \overline{\bar{x}_1\, x_2\, \bar{x}_3} \cdot \overline{x_1\, \bar{x}_2\, x_3}\,.$$

Із приведених функцій бачимо:

1) функція y_2 може бути отримана за допомогою звичайного інвертора, підключеного до виходу, відповідного $x_3\, \bar{x}_2\, \bar{x}_1 = 100_2 = 4_{10}$;

2) функція y_1 отримується об'єднанням через елемент 3**І-НІ** виходів, відповідних $x_3\, x_2\, x_1 = 111_2 = 7_{10}$; $\overline{x_3}\, x_2\, x_1 = 011_2 = 3_{10}$; $\bar{x}_3\, \bar{x}_2\, \bar{x}_1 = 000_2 = 0_{10}$;

3) аналогічно, функція y_3 отримується шляхом об'єднання виходів, номери яких відповідають $x_3\, x_2\, \bar{x}_1 = 110_2 = 6_{10}$; $\bar{x}_3\, x_2\, \bar{x}_1 = 010_2 = 2_{10}$; $x_3\, \bar{x}_2\, x_1 = 101_2 = 5_{10}$, за допомогою елемента 3**І-НІ**.

Використаємо мікросхему К1533ЛЕ4 (або її зарубіжні аналоги – 7427PC, DM7427N, ECG7427, FLH621, MC7427N, SN7427N, SN66N). Принципова схема, яка реалізує вказані функції, зображена на рис. 4.21.

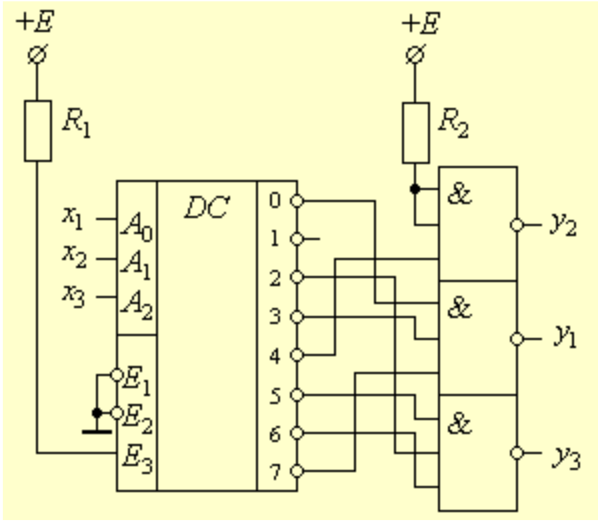


Рис. 4.21. Принципова схема, яка реалізує вказані функції

Нарощування дешифраторів забезпечується на тих самих принципах, що і нарощування мультиплексорів. Наприклад, на рис. 4.22 наведена схема перетворювача **“з 4 в 16”**. Наведена схема має можливість виконувати різні види кодових перетворень. У залежності від необхідного вихідного коду, необхідно використовувати ті виходи $y_0 \div y_{15}$ схеми, які відповідають вибраному коду. При пірамідальному нарощуванні перетворювачів кодів організація схеми залежить від характеру їх використання.

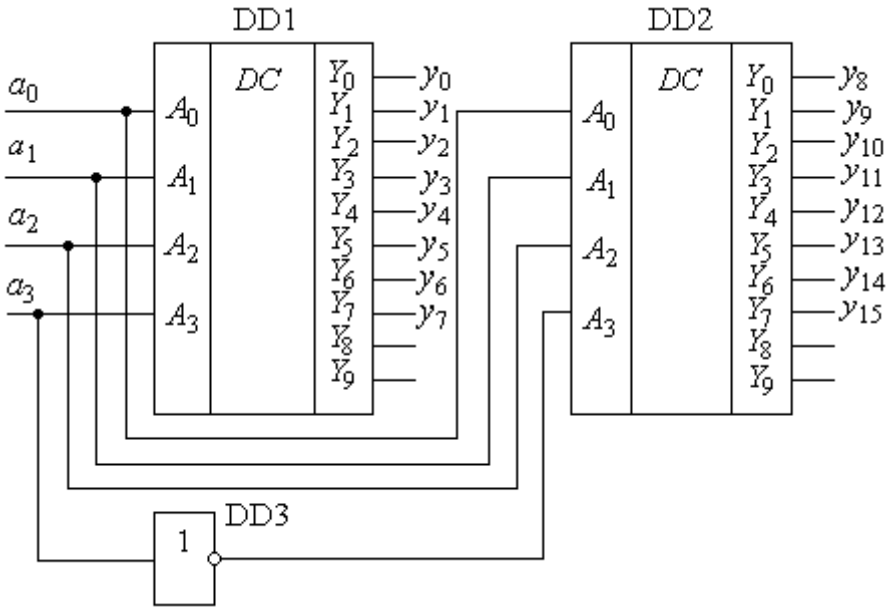


Рис. 4.22. Схема перетворювача **“з 4 в 16”**

На рис. 4.23 приводиться приклад створення демультиплексора на 32 виходи на базі мікросхеми К1533ИД4 (зарубіжні аналоги – 74155PC, UCY74155N). Схемотехнічно мікросхема складається з двох секцій, *Y* та *E*, кожна з яких має інверсний вхід *V*, об'єднаний з прямим та інверсним інформаційними входами Y та E через вхідний елемент **І**. Обидві секції мають об'єднані адресні входи A_0, A_1.

При забезпеченні на входах V_1 і V_2 низького логічного рівня сигнали, що поступають на входи Y та E, передаватимуться на свої виходи $Y_0 \div Y_3$ та $E_0 \div E_3$ у відповідності до зміни адресів. Якщо входи Y та E об'єднати, як показано на рис. 3.33 в мікросхемах DD2, DD5, то з'єднання входів Y надасть нам можливість організувати схему демультиплексора **"з 1 на 8"**. При забезпеченні $x = 0$ схема, представлена на рис. 4.23, перетворюється в дешифратор **"з 5 на 32"**. Якщо в мікросхемі DD1 об'єднати входи *Y* та *E* для створення адресної шини a_5, то при нульових сигналах на входах V_1 та V_2 і відповідному нарощенні вихідних мікросхем одержимо дешифратор **"з 6 на 64"**.

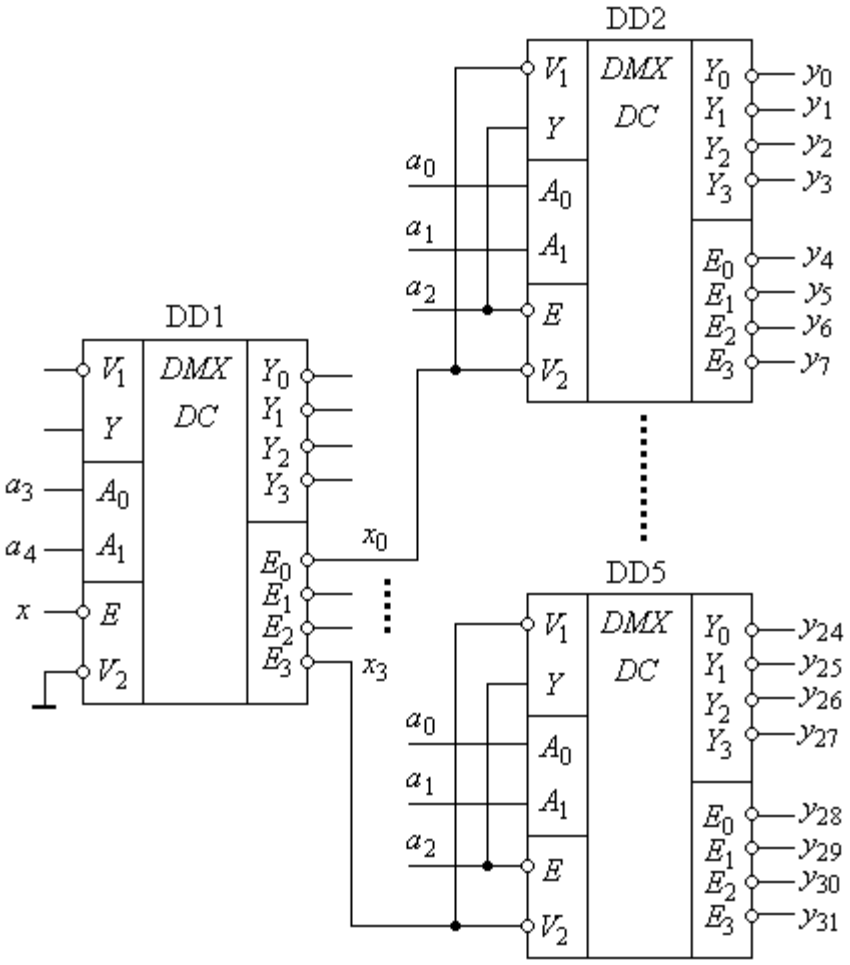


Рис. 4.23. Приклад створення демультиплексора на 32 виходи на базі мікросхеми К1533ИД4

Окрім розглянутих областей використання, кодові перетворювачі виступають складовою частиною керуючих автоматів. Широке використання вони знаходять у цифрових пристроях криптографічного захисту інформації.

На рис. 4.24 приведені два варіанти засекречення інформації.

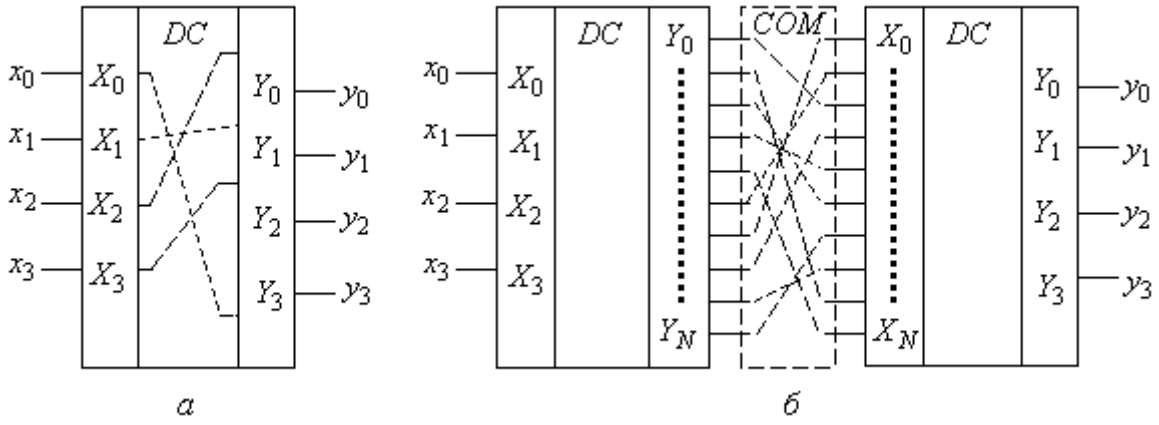


Рис. 4.24. Варіанти засекречення інформації

Спосіб перестановки з допомогою внутрішніх комутаторів (рис. 4.24, *а*) надає можливість для ***n***-входового перетворювача створити ***n!*** можливих варіантів. Спосіб підстановки (рис. 4.24, *б*) розширює кількість варіантів, адже перестановка виконується після перетворення вхідного двійкового коду. Кількість можливих варіантів з'єднань з допомогою комутатора **COM** підвищується до $2^n!$. Зворотнє перетворення інформації забезпечується блоком зі зворотнім законом перестановки. Використання подібних пристроїв дає можливість відкрито передавати та зберігати конфіденційну та секретну інформацію.

Перелік посилань

1. Рябенький В.М., Жуйков В.Я., Гулий В.Д. Цифрова схемотехніка: Навч. посібник. – Львів: Новий світ – 2000, 2009, – 736 с.

Лекція №5

Суматори

Найпростішою схемою є *напівсуматор*, який виконує операцію знаходження суми двох однорозрядних слів [1].

Роботу напівсуматора описує табл. 5.1, з якої знаходимо логічні рівняння для суми і для переносу:

$$S = a_0 \cdot \overline{b_0} + \overline{a_0} \cdot b_0 ;$$
$$P = a_0 \cdot b_0 .$$

Таблиця 5.1. Опис логіки роботи напівсуматора

a_0	b_0	S	P
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

При знаходженні суми багаторозрядних слів напівсуматор може використовуватись лише в молодшому розряді. Для знаходження суми старших розрядів схема повинна мати три входи – два інформаційні і третій – вхід переносу з молодших розрядів. Схема, що відповідає цим вимогам, називається *повним суматором*, а логіку її роботи описує табл. 5.2.

Таблиця 5.2. Опис логіки роботи повного суматора

N	a_i	b_i	p_{i-1}	S_i	P_{i+1}
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	0
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	1
6	1	1	0	0	1
7	1	1	1	1	1

З табл. 5.2 знаходимо наступні рівняння:

$$S_i = \vee 1,\ 2,\ 4,\ 7 = p_{i-1}\,(\overline{a_i} \cdot \overline{b_i} + a_i \cdot b_i) + \overline{p}_{i-1}\,(a_i \cdot \overline{b_i} + \overline{a_i} \cdot b_i) ;$$

$$P_i = \vee 3,\ 5,\ 6,\ 7 = p_{i-1}\,(\overline{a_i} \cdot b_i + a_i \cdot \overline{b_i} + a_i \cdot b_i) + \overline{p}_{i-1}\, a_i \cdot b_i = a_i \cdot b_i + p_{i-1}(a_i \oplus b_i)$$

для побудови логічної схеми однорозрядного повного суматора.

Знаходження суми багаторозрядних слів може виконуватись як послідовно, так і паралельно. Для послідовного виконання операції використовується один повний суматор, загальний для всіх розрядів. Для виконання операції суми слова подаються на інформаційні розряди послідовно, починаючи з молодших розрядів (рис. 5.1).

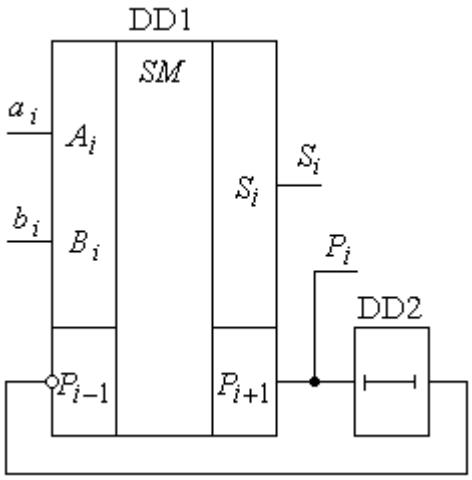


Рис. 5.1. Схема для послідовного виконання операції додавання

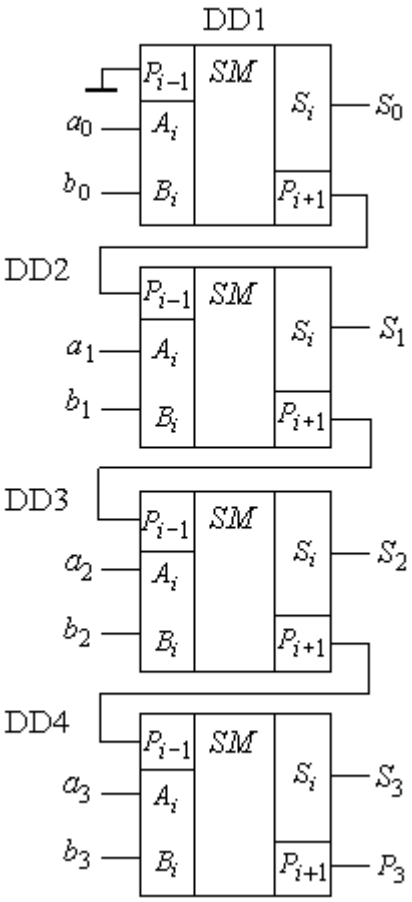


Рис. 5.2. Схема організації чотирьохрозрядного паралельного суматора з послідовним переносом

Для врахування результату операції з попереднім розрядом вхід та вихід переносу об’єднується через елемент затримки DD2. Цей елемент повинен забезпечити затримку та зберігання сигналу P_{i+1} на один такт виконання операції. Синхронно з інтервалом в один такт повинні подаватись і вхідні дані a_i, b_i.

Приведена схема виконання операції суми сама по собі нескладна, але вимагає складного апаратного забезпечення. До того ж, послідовне порозрядне виконання операції суттєво знижує швидкодію.

На рис. 5.2 приведена схема організації чотирьохрозрядного паралельного суматора з послідовним переносом. Для кожного розряду в ній використовуються окремі однорозрядні повні суматори, які з'єднані між собою послідовно по виходах-входах переносу. Незважаючи на паралельну одночасну подачу всіх розрядів слів, операція в ***i***-му розряді починає виконуватись тільки після того, як поступить сигнал переносу з (***i***-1)-го розряду. Швидкодія суматора визначається сумою затримок передачі сигналу переносу з молодшого розряду на вихід суматора старшого розряду.

За схемою, приведеною на рис. 5.2, виготовляються мікросхеми К1533ИМ3 (зарубіжні аналоги – 7483PC, CDB483E, FLH241, MC7483N, N7483N, SN7482AN, UCY7483N), умовне зображення якої приведено на рис. 5.3 (мікросхема DD2). Мікросхема може використовуватись для операцій як з додатними, так і з від'ємними числами. При виконанні операції додавання на вхід p_0 необхідно подати сигнал низького рівня.

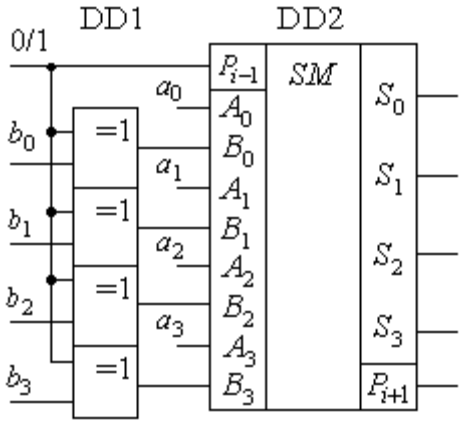


Рис. 5.3. Умовне зображення мікросхеми К1533ИМ3

Нарощування розрядності суматорів виконується на схемі, приведеній на рис. 5.2. Таблиця станів для суматорів за своєю сутністю є таблицею суми двох чисел (див. табл. 5.1 – табл. 5.2). Вона має простий вигляд і зручна в

користуванні для чисел невеликої розрядності. Суматор К1533ИМ3, як і всі комбінаційні схеми, має широкі функціональні можливості. Перш за все, він може використовуватись для знаходження суми чисел з меншою розрядністю. Схема входів-виходів при цьому може бути різноманітною. Його можливо використовувати і для віднімання двох чисел. Ця операція виконується шляхом додавання зменшуваного і зменшуючого, представленого у доповнюючому коді. Для виконання цієї операції зменшуюче $b_3\, b_2\, b_1\, b_0$ з допомогою інверторів перетворюється в зворотній код, а потім до нього по входу p_0 додається одиниця.

Приклад 5.1. Розглянути мікросхему К1533ИМ3. Пояснити, як використовувати цей чотирьохрозрядний суматор для додавання:

а) двох чотирьохрозрядних чисел;

б) двох дворозрядних та двох однорозрядних чисел;

в) двох восьмирозрядних (байтових) чисел.

Розв’язання. При додаванні двох чотирьохрозрядних слів необхідно вхід переносу P_0 заземлити (рис. 5.4). В дворозрядному суматорі входами є A_1, A_2 та B_1, B_2, а виходами S_1, S_2, S_3. Останній є виходом переносу. В однорозрядному суматорі використовуються входи A_4, B_4, а виходами є S_4 та P_4 . При додаванні двох однобайтових слів необхідно наростити два однотипних суматори. При цьому вихід P_4 суматора напівбайтів молодших розрядів з’єднується з входом P_0 суматора напівбайтів старших розрядів слів, що складаються.

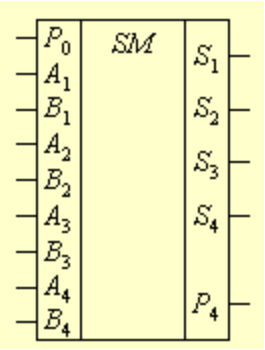


Рис. 5.4. Умовне зображення схеми суматора для додавання двох чотирьохрозрядних слів

Операції знаходження суми та різниці можна поєднати в одному пристрої, як показано на рис. 5.3. Для виконання операції додавання на вхід P_{i-1} переносу подається сигнал низького рівня. Відповідно на входах елементів **ВИКЛ. АБО** число $b_3\, b_2\, b_1\, b_0$ буде представлене в прямому коді. При подачі на вхід P_{i-1} сигналу високого рівня, число $b_3\, b_2\, b_1\, b_0$ буде представлене у доповнюючому коді.

При виконанні операції віднімання результат на виході суматора залежить від співвідношення між числами, що беруть участь в операції. При

$$a_3\, a_2\, a_1\, a_0 > b_3\, b_2\, b_1\, b_0$$

результат одержується в прямому коді, а в розряді переносу P_{i+1} з'являється **1**.

Коли ж

$$a_3\, a_2\, a_1\, a_0 < b_3\, b_2\, b_1\, b_0,$$

результат отримується в зворотньому коді, а в розряді переносу P_{i+1} одержується **0**.

Приклад 5.2. Розробити пристрій двійкового множення на суматорах та логічних елементах **І**, що дозволяє множити чотирьохрозрядне число ***A*** на трьохрозрядне число ***B***.

Розв'язання. Після множення трьохрозрядного і чотирьохрозрядного кодів отримується семирозрядний код. Результат операції утворюється за рахунок паралельного множення множеного на кожен розряд множника і складання проміжних добутків зі зсувом на один розряд. При підрахуванні добутку виконуються наступні дії

			A_4	A_3	A_2	A_1
			×	B_3	B_2	B_1
			A_4B_1	A_3B_1	A_2B_1	A_1B_1
		A_4B_2	A_3B_2	A_2B_2	A_1B_2	
	A_4B_3	A_3B_3	A_2B_3	A_1B_3		
M_7	M_6	M_5	M_4	M_3	M_2	M_1

Тут **$M_1 = A_1B_1$; $M_2 = A_1B_2 + A_2B_1$; $M_3 = A_1B_3 + A_2B_2 + A_3B_1$; $M_4 = A_2B_3 + A_3B_2 + A_4B_1$; $M_5 = A_3B_3 + A_4B_2$; $M_6 = A_4B_3$;** ***M_7 – біт переносу із попереднього розряду; знак "+" – знак арифметичного додавання.***

Однорозрядне множення забезпечується елементами **I**. Суми можуть бути реалізовані суматорами К1533ИМ3. Для реалізації цієї операції на рис. 5.5 приведена принципова схема.

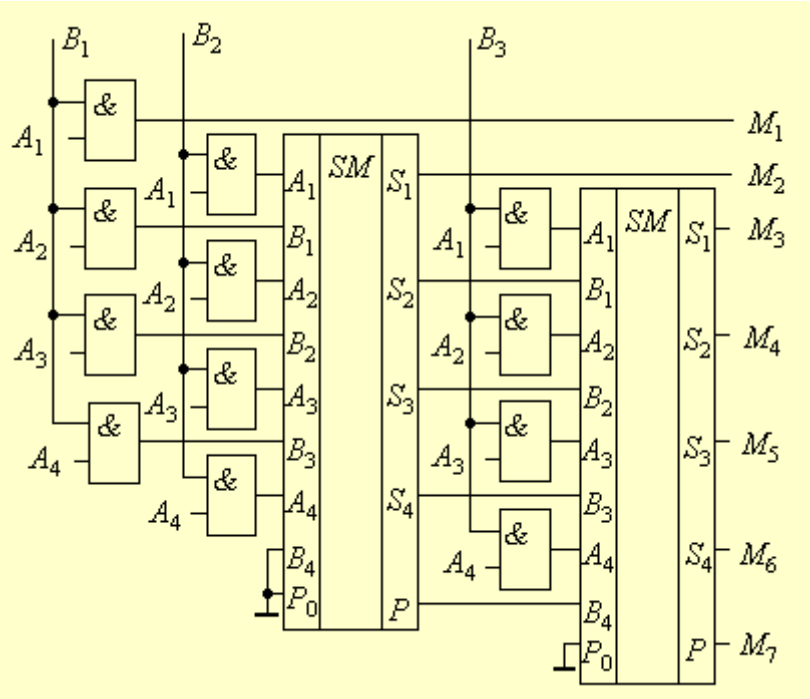


Рис. 5.5. Принципова схема однорозрядного множення

У зв'язку з затримкою переносу, швидкодія суматора К1533ИМ3 обмежена. Для підвищення швидкодії в ряді серій мікросхем виготовляють спеціальні вузли, які називаються *пристроями прискореного переносу*. Принцип прискореного переносу полягає у тому, що для кожного двійкового розряду знаходять два допоміжних логічних сигнали: $g_i = a_i b_i$ (сигнал генерації переносу) та $h_i = a_i + b_i$ (сигнал розповсюдження).

Сигнал $g_i = 1$, коли складові ***i***-го розряду такі, що перенос в старший розряд $P_{i+1} = 1$ незалежно від значення вхідного переносу P_{i-1} .

Сигнал h_i іноді називають *сигналом прозорості*, тому що відповідно до його визначення при $h_i = 1$ сигнал переносу з попереднього розряду p_i вільно

пройде наступний розряд P_{i+1}. За допомогою g_i та h_i можна представити роботу вузла переносу одного розряду суматора $P_{i+1} = g_i + p_i h_i$.

Будуючи структуру зв'язків між переносами, починаючи зі старшого розряду до нульового, можна створити логічну схему, яка дасть можливість організувати паралельний суматор. Серед мікросхем ТТЛ паралельний перенос має суматор К1533ИМ6, який має затримку розповсюдження сигналу t_3 від входів до виходу P_{i+1} приблизно у 2 рази меншу, ніж К1533ИМ3. В мікросхемах КМОН суматори також виконуються з вузлом паралельного переносу.

Для прискорення переносу в суматорах з великою кількістю розрядів використовується принцип групового переносу. Суматор розділяється на групи, які мають невелику розрядність і входи-виходи переносу. Сутність групового переносу полягає в тому, що, додатково до вузла переносу, в групі будують вузол переносу між групами більш високого рівня, який формує сигнали групових переносів, котрі подаються на входи міні-суматорів. Паралельний перенос між групами суттєво прискорює роботу багаторозрядних суматорів.

Компаратори

Компаратори – це пристрої, що виконують порівняння двох чисел [1].

Мікросхеми компараторів визначають не тільки рівність, але й нерівність двох чисел. Для цього мікросхема має три виходи: «$\boldsymbol{A} > \boldsymbol{B}$», «$\boldsymbol{A} < \boldsymbol{B}$» і «$\boldsymbol{A} = \boldsymbol{B}$», на яких у залежності від співвідношення величин $A = a_3\, a_2\, a_1\, a_0$, $B = b_3\, b_2\, b_1\, b_0$ з'являється активний рівень сигналу.

Логічна функція, що реалізує алгоритм роботи компаратора для одного розряду:

$$y_i = a_i \cdot b_i + \overline{a_i} \cdot \overline{b_i} = \overline{a_i \oplus b_i} \qquad (5.1)$$

Для багаторозрядних чисел функція (5.1) повинна виконуватись для кожного розряду. Нерівність $\boldsymbol{A} > \boldsymbol{B}$ має місце тоді, коли в числі $\boldsymbol{A}$ одиниці

дорівнює більш старший розряд, ніж у числі $\boldsymbol{B}$. Для чотирьохрозрядного компаратора така нерівність може бути записана у вигляді:

$$(A > B) = a_3\overline{b_3} + \overline{(a_3 \oplus b_3)} \cdot a_2\overline{b_2} + \overline{(a_3 \oplus b_3)} \cdot \overline{(a_2 \oplus b_2)} \cdot a_1\overline{b_1} +$$

$$+ \overline{(a_3 \oplus b_3)} \cdot \overline{(a_2 \oplus b_2)} \cdot \overline{(a_1 \oplus b_1)} \cdot a_0\overline{b_0}\,.$$

Побудова багаторозрядного компаратора можлива на базі суматора. Його реалізація можлива при виконанні операції віднімання.

Дійсно, в відповідності до законів арифметики при $\boldsymbol{A} = \boldsymbol{B}$ на виходах всіх розрядів суматора s_0, s_1, s_2, s_3 буде **0** при $P_{i+1} = 1$.

При $\boldsymbol{A} > \boldsymbol{B}$ значення **1** буде як на виході переносу $P_{i+1} = 1$, так і хоча б на одному виході s_i. Тому ознакою $\boldsymbol{A} > \boldsymbol{B}$ може бути функція

$$y_> = p_{i+1} \cdot (\sum s_i)\,.$$

При $\boldsymbol{A} < \boldsymbol{B}$ результат наявності переносу $P_{i+1} = 0$.

Відповідно до сказаного, логічна схема компаратора на базі суматора К1533ИМ3 (зарубіжні аналоги – 7483PC, CDB483E, MC7483N, N7483N, SN7483AN, UCY7483N) має вигляд, приведений на рис. 5.6, *а*. Прикладом серійних мікросхем є компаратор К564ИП2 (зарубіжні аналоги – CD4585BD, CD4585BH, CD4585BK, CD4585BKF3) (КМОН) (рис 5.6, *б*); К555СП1 (зарубіжні аналоги – 74LS85N, DL083D, SN74LS85N, UCY74LS85N) (ТТЛ).

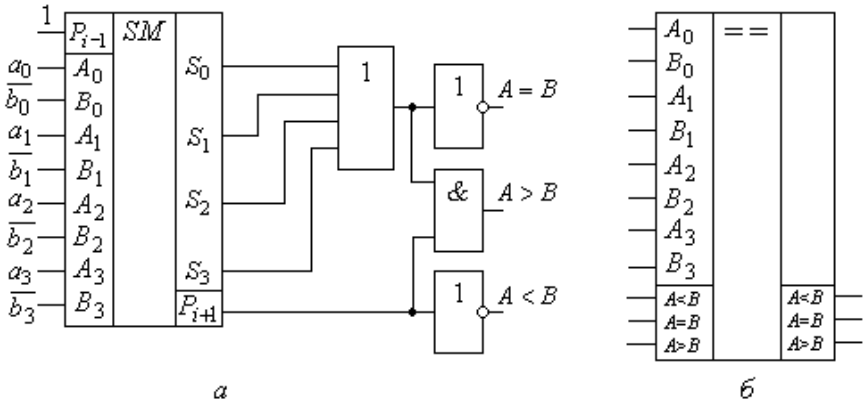


Рис. 5.6. Логічна схема компаратора на базі суматора К1533ИМ3

Мікросхеми мають розширення по входу (входи «$\boldsymbol{A} > \boldsymbol{B}$», «$\boldsymbol{A} < \boldsymbol{B}$» і «$\boldsymbol{A} = \boldsymbol{B}$»), що дає можливість як послідовного нарощування, так і паралельного.

На рис. 5.7 приведена схема послідовного нарощування двох компараторів К561ИП2.

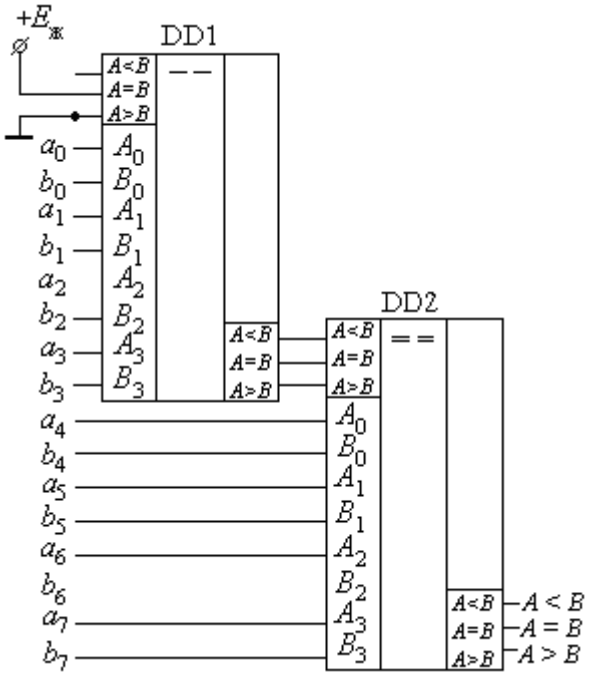


Рис. 5.7. Схема послідовного нарощування двох компараторів К561ИП2

Входи розширення першої мікросхеми необхідно з’єднати відповідно до рис. 5.7, а виходи з’єднуються безпосередньо з входами наступної мікросхеми. При такому способі нарощування компараторів затримки додаються, а стан виходів другого і послідуючих компараторів залежить не тільки від побітних значень порівнюваних слів, а й від результату порівняння молодших розрядів цих слів. Тому значення виходів кожного з таких компараторів можуть бути описані рівняннями:

$$(A > B)_{\text{вих}} = (A > B)_{n-m} + (A = B)_{n-m} \cdot (A > B)_{\text{вх}} ;$$

$$(A = B)_{\text{вих}} = (A = B)_{n-m} \cdot (A = B)_{\text{вх}} ;$$

$$(A < B)_{\text{вих}} = (A < B)_{n-m} + (A = B)_{n-m} \cdot (A < B)_{\text{вх}} ;$$

де n, m – діапазон розрядів порівнюваних слів, які приєднані до компаратора.

При пірамідальному способі з’єднання тривалість затримок зменшується.

При нарощуванні мікросхем компараторів необхідно уважно використовувати таблиці станів мікросхем, які відрізняються за характером

сигналів на входах нарощування навіть для близьких мікросхем – наприклад, К561ИП2 (зарубіжні аналоги – CD4585BD, CD4585BE, CD4585BK) і К564ИП2.

Компаратори широко використовуються в інформаційних системах для виділення необхідного слова в потоці цифрової інформації, для відмітки часу в часових пристроях, для виконання умовних переходів в обчислювальних пристроях. У пристроях автоматики компаратори використовуються для контролю виходу величин за межі допусків і т.п.

Арифметично-логічні пристрої

Арифметично-логічні пристрої (АЛП, ALU – Arithmetic-Logic Unit) – це спеціалізовані мікросхеми, які виконують арифметичні та логічні операції у відповідності до двійкового коду, який подається на керуючі входи мікросхеми [1]. Умовне позначення АЛП в електронних схемах приведене на рис. 5.8.

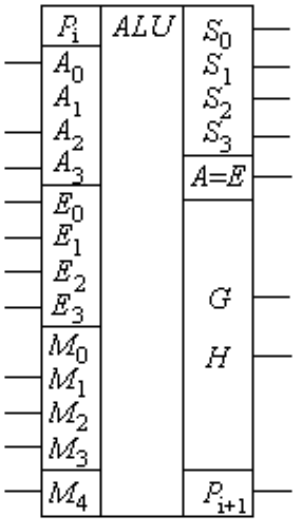


Рис. 5.8. Умовне позначення АЛП

Мікросхеми АЛП, що належать до різних серій, – наприклад, ТТЛ (К1533ИП3 (зарубіжні аналоги – 74181PC, SN74181N)), КМОН (К564ИП3 (зарубіжні аналоги – 4581BDC, 4581BDM, 4581BFC, GD4581BC, GD4581BDC, HD14581B, MC14581BAL, MC14581BCL, MC14581BCPDS, SCL4581B)), – за своїми функціональними властивостями подібні. Вони призначені для роботи з двома чотирьохрозрядними двійковими словами. Конкретний вид операції

задається п’ятирозрядним керуючим кодом, що подається на входи M_0, M_1, M_2, M_3, M_4. Як результат, цей пристрій може виконувати 32 різноманітні команди. Вхід M_4 задає режим АЛП на виконання 16 логічних ($M_4 = 1$) або 16 арифметичних ($M_4 = 0$) операцій. Допоміжні входи ***G*** та ***H*** призначені для використання АЛП в операціях над багаторозрядними словами при одночасному використанні спеціально для цієї мети призначеної мікросхеми К1533ИП4 (зарубіжні аналоги – DM74182N, FLH411, MC74182N, SN74182N, SN84182N, TL84182N, UCY74182N) і К564ИП4 (4582BDC, 4582BDM, 4582BFC, GD4582BC, GD4582BDC, HD14582B, MC14582BAL, MC14582BCL, MC14582BCP, SCL4582B, SCL4582BC). АЛП призначений для роботи з сигналами як позитивної, так і негативної логіки, що відображається у таблиці станів мікросхеми (табл. 5.3). У табл. 5.3 ***L*** та ***H*** позначені відповідно низький і високий рівні сигналів, незалежно від типу логіки. Знаки (+) та (–) – знаки арифметичних операцій. Результати арифметичних дій виражені в доповнюючому коді. Так як між зворотнім та доповнюючими кодами існує однозначний зв’язок, то скрізь, де вказана арифметична дія “**–1**”, результат на виходах зчитується у зворотному коді.

На відміну від суматорів, сигнал переносу на вході і виході є інверсним по відношенню до сигналів на входах ***A*** та ***E***. Сигнали переносу діють лише при виконанні арифметичних операцій. При використанні логічних операцій АЛП всі зазначені в табл. 5.3 функції виконує порозрядно. Функція переносу діє тільки при виконанні арифметичних дій, але в результатах цих операцій символи ***AE*** означають операцію **I**.

Якщо АЛП використовується в якості компаратора (операція $A \vee \overline{E}$), то при $A = E$ на відповідному виході формується сигнал високого рівня. При $A \neq E$ відповідний сигнал формується на виході P_{i+1} .

У зв’язку з інтенсивним розвитком мікропроцесорної техніки розглянута схема АЛП у реальній схемотехніці майже не використовується. Її вивчення

має скоріш методичне значення, адже дає можливість практично вивчати ідеологію побудови мікропрограмних автоматів і мікропроцесорів.

Таблиця 5.3. Таблиці станів мікросхеми

Входи вибору функції				Вхід-вихід (негативна логіка)		Вхід-вихід (позитивна логіка)	
M_3	M_2	M_1	M_0	Логічні функції M_4 = H	Арифметичні дії M_4 = L; P_i = L	Логічні функції M_4 = H	Арифметичні дії M_4 = L P_i = H
L	*L*	*L*	*L*	$\overline{A}$	*A – 1*	$\overline{A}$	*A*
L	*L*	*L*	*H*	$\overline{A} \vee \overline{E}$	*AE – 1*	$\overline{A}\,\overline{E}$	$A \vee E$
L	*L*	*H*	*L*	$\overline{A} \vee E$	$A\overline{E} - 1$	$\overline{A}E$	$A \vee \overline{E}$
L	*L*	*H*	*H*	*1*	*-1*	*0*	*-1*
L	*H*	*L*	*L*	$\overline{A}\,\overline{E}$	$A + (A \vee \overline{E})$	$\overline{A}E$	$A + A\overline{E}$
L	*H*	*L*	*H*	$\overline{E}$	$A + (A \vee \overline{E})$	$\overline{E}$	$(A \vee E) + A\overline{E}$
L	*H*	*H*	*L*	$\overline{A}(\cdot)\overline{E}$	*A-E-1*	*A(+)E*	*A-E-1*
L	*H*	*H*	*H*	$A \vee \overline{E}$	$A \vee \overline{E}$	$A\overline{E}$	$A\overline{E} - 1$
H	*L*	*L*	*L*	$\overline{A}E$	$A + (A \vee E)$	$\overline{A} \vee E$	*A+AE*
H	*L*	*L*	*H*	*A(+)E*	*A+E*	$\overline{A}(\cdot)\overline{E}$	*A+E*
H	*L*	*H*	*L*	*E*	$A\overline{E} + (A \vee E)$	*E*	$(A \vee \overline{E}) + AE$
H	*L*	*H*	*H*	$A \vee \overline{A}E$	*AE*	*AE*	*AE-1*
H	*H*	*L*	*L*	*0*	*A+A*	*1*	*A+A*
H	*H*	*L*	*H*	$A\overline{E}$	*AE+A*	$A \vee \overline{E}$	$(A \vee E) + A$
H	*H*	*H*	*L*	*AE*	$A\overline{E} + A$	$A \vee \overline{E}$	$(A \vee \overline{E}) + A$
H	*H*	*H*	*H*	*A*	*A*	*A*	*A-1*

Буферні елементи

При побудові складних цифрових і мікропроцесорних систем створюється шинна форма обміну інформацією між окремими пристроями [1]. *Шина* – це паралельна група провідників, по якій передаються інформаційні сигнали в обох напрямках і до якої може бути приєднана досить велика група пристроїв, що передають та приймають інформаційні сигнали. В загальному плані в залежності від характеру сигналів, що передаються і приймаються, шини класифікуються на *адресні, командні* та *шини даних*.

Оскільки кожен з пристроїв повинен мати право доступу до шин, то для забезпечення необхідного порядку при роботі з шинами існує велика кількість мікросхем, які спеціально призначені для обслуговування різноманітних пристроїв при їх роботі з шинами. Такі мікросхеми здебільшого не виконують логічних функцій. Вони призначені для:

- формування цифрових сигналів для забезпечення необхідного струму, напруги, потужності;
- забезпечення тимчасової передачі шин від одних пристроїв до інших;
- розділення в часі інформації, що передається в обох напрямках;
- підключення (чи відключення) до шин необхідних пристроїв;
- виконання допоміжних функцій – таких, як тимчасове запам’ятовування інформації, що передається по шині, її інверсія і т. п.

Всі вказані функції забезпечуються спеціальними *буферними елементами*. Основою будь-якого буферного елемента є двонаправлених підсилювач, який складається з двох ідентичних, зустрічно-паралельно з’єднаних підсилювачів потужності, кожен з яких вмикається лише при передачі інформації від його входу до виходу. Для забезпечення такої комутації підсилювачів використовується спеціальна логіка. Кількість пар підсилювачів відповідає кількості провідників шини. Якщо, наприклад, шина даних має розрядність 8 біт, то мікросхема шинного підсилювача повинна мати 8 пар

зустрічно-паралельно з'єднаних підсилювачів, керованих єдиним логічним вузлом [1].

У мікросхемах обслуговування широко використовуються логічні елементи з *Z*-станом. Переведення мікросхеми у *Z*-стан дає можливість тимчасово відділити шини від цифрових пристроїв, внаслідок чого ними можуть користуватися інші пристрої.

У технічній літературі буферні елементи часто називають *"шинними драйверами"*. До них відносяться мікросхеми К1533АП3 (SN74ALS240), К1533АП4 (SN74ALS241), а також мікросхеми груп ЛН, ЛП, ХЛ.

Перехідні процеси в логічних схемах

Логічні ключі характеризуються часом затримки $t_з$, який пов'язаний з перехідними процесами зміни стану. Тривалість перехідних процесів не є стабільною величиною і залежить від багатьох факторів. При використанні мікросхем, в яких внутрішні затримки малі, при проектуванні цифрових пристроїв слід обов'язково враховувати тривалість перехідних процесів у лініях зв'язку та вплив параметрів ліній зв'язку на тривалість перехідних процесів в мікросхемах. Ці задачі досить складні, і їх розв'язання можливе лише з використанням спеціальних методів моделювання.

При використанні мікросхем з тривалістю зміни станів, не меншою 20 нс (ТТЛ, КМОН), перехідні процеси в лініях зв'язку не враховуються, а характер лінії зв'язку, величина та характер навантаження враховуються максимальною тривалістю затримки. Це значно спрощує процес проектування цифрових пристроїв, адже затримки можуть бути враховані відразу на стані логічного проектування. В таких ситуаціях, коли мають місце значні затримки в лініях зв'язку, вони також можуть враховуватись як дискретні затримки.

На жаль, у паспортних даних на мікросхеми подається тільки максимальна тривалість затримки. Реально ж вона може змінюватись в широких межах, а мінімальна величина затримки паспортно не визначена. Звідси витікає, що рівень виходу елемента в цифровій схемі в інтервалі часу від

0 до $t_з$ не визначений. Він називається *станом невизначеності* і позначається символом $x_н$. Такий стан елемента в схемі впливає на інші елементи, на виходах яких можуть мати місце як визначені стани **“1”**, **“0”**, так і невизначений стан $x_н$. Як результат, поведінка логічних елементів в перехідних процесах описується законами трійкової логіки з такими логічними співвідношеннями:

$$\overline{x_н} = x_н\ ;\qquad x_н + 0 = x_н\ ;\qquad x_{н1} \cdot x_{н2} = x_н\ ;$$

$$x_н \cdot 1 = x_н\ ;\qquad x_н \oplus 0 = x_н\ ;\qquad x_{н1} + x_{н2} = x_н\ ;$$

$$x_н \cdot 0 = 0\ ;\qquad x_н + 1 = 1\ ;\qquad x_{н1} \oplus x_{н2} = x_н\ .$$

$$x_н \oplus 1 = x_н\ ;$$

Приведені формули широко використовуються при аналізі перехідних процесів, в тому числі при побудові часових діаграм. Останні зображаються одним з двох способів, які приведені на рис. 5.9, *а* для елемента **2І** $(y = x_0 \cdot x_1)$.

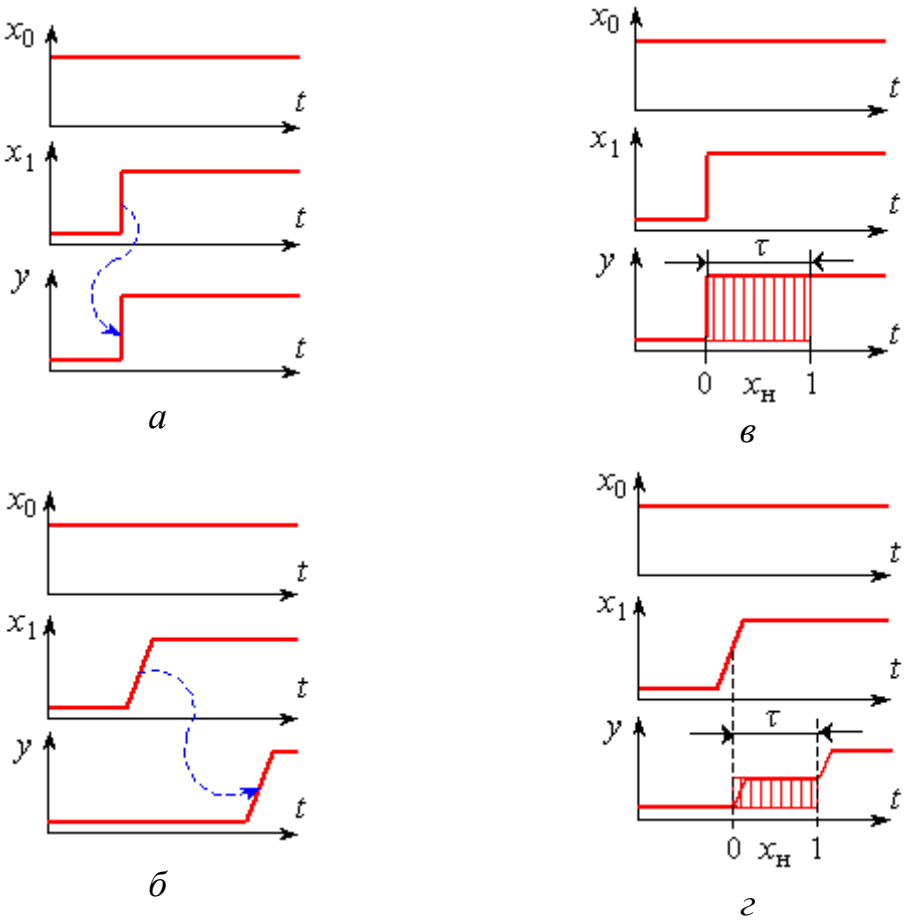


Рис. 5.9. Способи зображення часових діаграм перехідних процесів

У практиці аналізу схем використовуються чотири типи зображення перехідних процесів, які використовуються в залежності від необхідного рівня деталізації. Найпростіша з них – діаграма рис. 5.9, *а* – пояснює лише логічні зв'язки між вхідними сигналами та вихідними станами без урахування вихідних затримок.

Часова діаграма, приведена на рис. 5.9, *б*, враховує часові затримки, причому їх максимальне значення. Вона дає можливість оцінити максимальні значення тривалості перехідних процесів. Лінії зі стрілками показують взаємозв'язки між вхідними сигналами і вихідними станами мікросхеми.

Діаграми рис. 5.9, *в*, *г* враховують стани невизначеності елементів. Вони дають можливість змоделювати поведінку схеми при будь-яких комбінаціях і часових співвідношеннях внутрішніх затримок. Інтервал τ невизначеного стану $x_{\text{н}}$ дорівнює величині $\boldsymbol{t}_{\text{З}}$ конкретної мікросхеми.

Врахування станів невизначеності дає можливість виявити важливий ефект, який обов'язково необхідно враховувати при аналізі цифрових схем.

Розглянемо схему, що приведена на рис. 5.10, *а*, на вхід якої подається сигнал x_0 у вигляді одиночного імпульсу тривалістю $\boldsymbol{T}$.

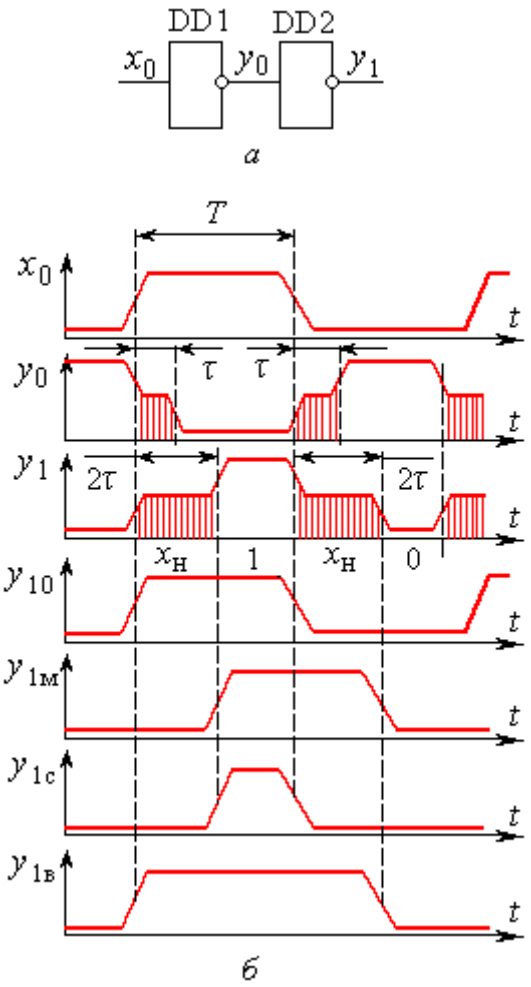


Рис. 5.10. Схема та часові діаграми при подачі одиночного імпульсу

Якщо тривалості інтервалів невизначеності **τ** при зміні станів мікросхем з **“0”** в **“1”** і з **“1”** в **“0”** однакові, то на виході y_1 матимемо сигнал, тривалість і початок якого суттєво залежить від величини затримки **τ** та співвідношення між затримками при зміні станів з **“0”** в **“1”** та навпаки. Якщо, наприклад, тривалість затримок $\tau = 0$, то вихідний імпульс y_{10} повторюватиме вхідний. Якщо затримки на зміну станів з **“1”** в **“0”** і навпаки однакові і максимальні, то сигнал $y_{1м}$ матиме затримку по відношенню до вхідного на величину **2τ**.

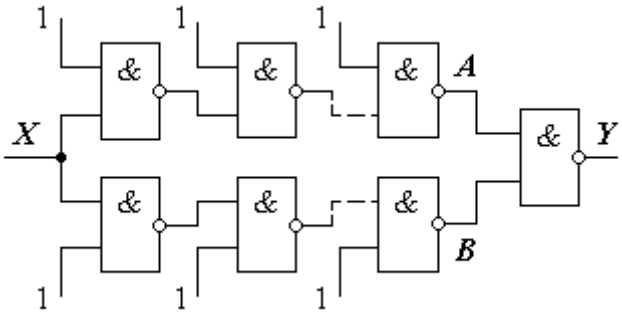
При різних співвідношеннях між затримками вихідний сигнал може бути як скороченої довжини $y_{1с}$, так і видовженої $y_{1в}$ (рис. 5.17, *б*).

Звичайно, що реально сигнал y_1 може мати випадкові співвідношення між тривалістю імпульсу та паузою, випадкову затримку початку і кінця

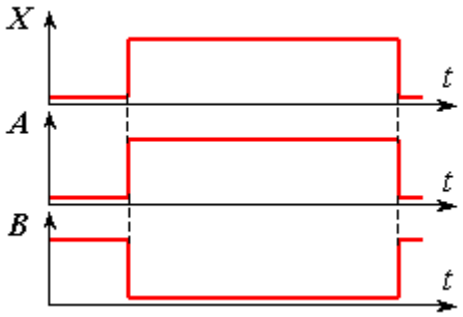
імпульсу. Як результат, імпульс, який буде пропущений через два ідентичні канали, на їх виходах може суттєво відрізнятись. Якщо ж один канал має ***m*** елементів, а другий ***n***, то, відповідно, матимемо затримки на $m\tau$ і $n\tau$, і вихідні імпульси можуть навіть не співпадати в часі. Якщо при проектуванні схеми, що має послідовно з'єднані ***k*** елементи, необхідно одержати імпульс тривалістю $t_{\text{і}}$, то в результаті вихідний імпульс з двостороннім допуском буде мати тривалість $t_{\text{і}} \pm k\tau$.

Найбільші проблеми наявність затримок створює при передачі імпульсів через паралельні канали. Розглянемо схему, що приведена на рис. 5.11, *а*. Вхідний імпульс подається на два паралельних канали з елементами **І-НІ**, один з яких назвемо ***A*** – з парною кількістю елементів, а другий ***B*** – з непарною. При наявності логічних одиниць на одному з входів кожного логічного елемента обидва канали є прозорими для сигналу ***x***. При відсутності часових затримок в каналах, їх вихідні сигнали ***A*** і ***B*** на вході об'єднуючого елемента **І-НІ** будуть строго протифазні і, як результат, $y = 0$ (рис. 5.11, *б*).

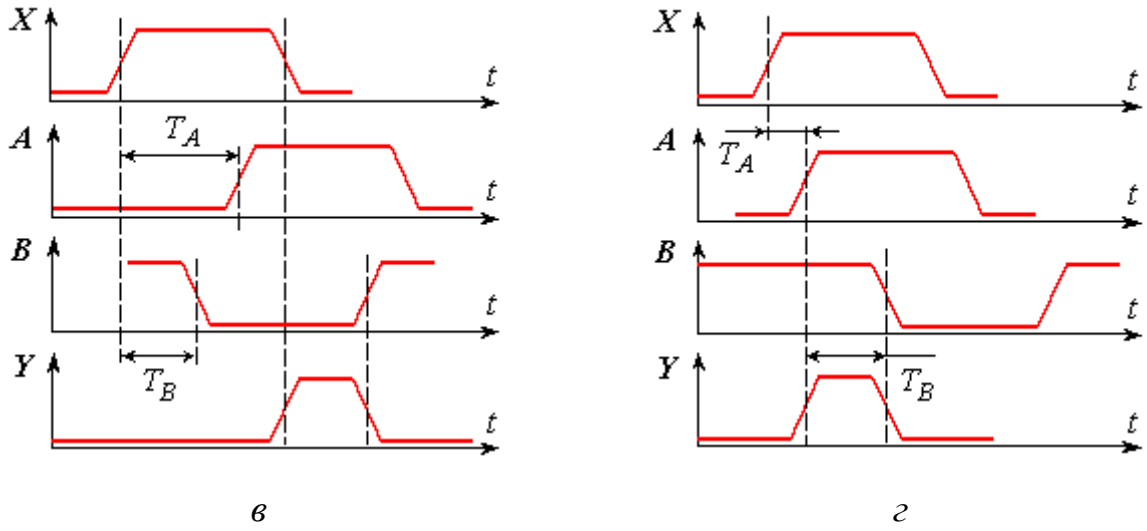
На рис. 5.11, *в, г* приведені приклади часових діаграм в тих ситуаціях, коли, відповідно, затримки $T_A > T_B$ і $T_B > T_A$. В обох випадках на виході ***y*** з'явиться сигнал, який неможливо передавати на основі алгебри Буля без врахування часових затримок. Одержаний сигнал є сигналом перешкоди, що може привести до непередбачених наслідків в послідуючих схемах.



а



б



в *г*

Рис. 5.11. Схема та часові діаграми при передачі імпульсів через паралельні канали

Розглянуте явище називається *„гонками”* або *„змаганнями”*. Головна проблема, яка пов’язана з гонками, полягає у тому, що проектант не має реальних можливостей визначити момент появи можливої перешкоди та її тривалість. Гонки називаються *критичними* або *недопустимими*, якщо хоча б один вихідний сигнал під час перехідного процесу змінюється більш ніж один раз. Критичні змагання суттєво впливають на роботу цифрових пристроїв та їх проектування.

Розглянемо схему, що приведена на рис. 5.12. При $x=1$ на обох виходах $y_{\tau 1}$ і $y_{\tau 2}$ матимемо сигнали низького рівня. Зміна ***x*** з **“1”** в **“0”** приведе до того, що на виході з меншим значенням τ (наприклад, $\tau_2 < \tau_1$) з’явиться одиничний сигнал раніше, ніж на іншому. Він забезпечить підтримку одиничного значення на вході DD1, внаслідок чого $y_{\tau 1}=0$. Оскільки значення τ_1 і τ_2 невідомі, то такий перехід приведе до невизначеності станів $y_{\tau 1}$ і $y_{\tau 2}$.

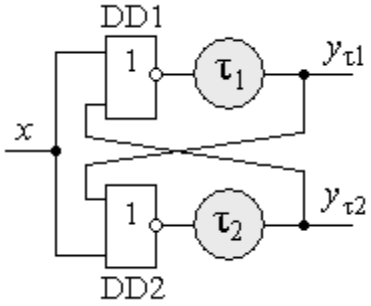


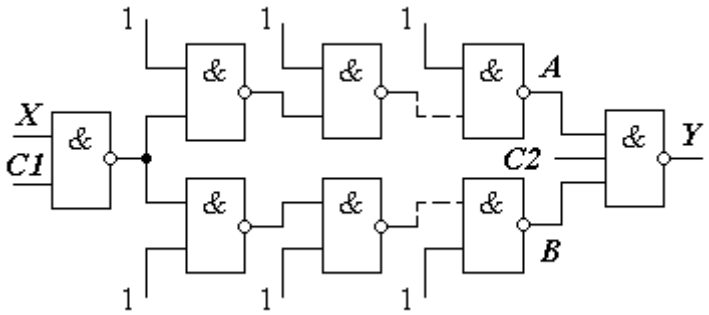
Рис. .5.12. Схема при «гонках»

На практиці використовуються три наступні способи боротьби з гонками:

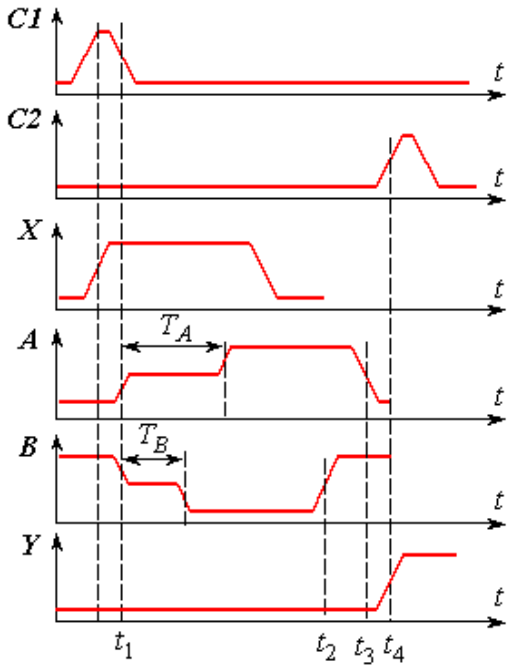
- *синхронізація;*
- *побудова протигоночних систем;*
- *врахування мінімального часу затримки.*

Синхронізація є найбільш універсальним засобом боротьби з гонками. Її суть полягає у наступному: по всьому цифровому пристрою створюється єдина система синхронізуючих сигналів. У залежності від типу елементної бази, використовуються однофазна або двофазна системи синхронізації. В останньому випадку синхронізація забезпечується двома послідовностями імпульсів ***C1*** та ***C2***, одна з яких має затримку по відношенню до іншої на половину періоду.

Для розглянутої вище схеми з паралельними каналами (див. рис. 5.11, *а*) синхронна схема з двофазною синхронізацією приведена на рис. 5.13, *а*.



а



б

Рис. 5.13. Синхронна схема з двофазною синхронізацією та часові діаграми її роботи

За синхросигналом ***C1*** сигнал ***x*** одночасно подається на два канали. Проходячи канали з різними затримками T_A і T_B, в асинхронній схемі (рис. 5.11, *а*) на виході ***Y*** в інтервалі часу $t_2 \div t_3$ з'являється перешкода. Але у синхронній схемі інформація на виході ***Y*** з'явиться лише після подачі синхроімпульсу ***C2*** в момент t_4, коли всі перехідні процеси в каналах завершаться. Важливе значення в цьому випадку має інтервал $t_1 \div t_4$ між двома синхроімпульсами, який вибирається таким, щоб перехідні процеси найбільшої тривалості могли завершитись до подачі ***C2***.

У залежності від того, що виступає приймачем сигналу ***Y***, залежить назва сигналу ***C2***. В розглянутому випадку, коли сигнал ***Y*** не запам'ятовується, а ***C2*** використовується лише для того, щоб в ***Y*** не з'являлась перешкода, він називається *стробом*, а процес відсікання перешкод – *стробуванням*. Якщо ж необхідно запам'ятати в схемі пам'яті, то сигнал ***C2*** подається на цю схему, що робить її синхронною, а ***C2*** – відповідно *синхросигналом*.

Слід зазначити, що синхронізація дає можливість суттєво спростити процес проектування цифрових схем, адже значно спрощує вирішення проблеми гонок.

Протигоночні системи – це другий практичний спосіб боротьби з гонками. Вони будуються так, що в них відсутній ризик появи на виході сигналів, не передбачених логікою роботи схеми. Прикладом подвійної схеми може бути два паралельних канали з однаковою кількістю елементів, об'єднаних елементом **АБО**; який би канал не виграв гонку, результат буде однаковим, зміниться лише момент його появи.

Для усунення гонок необхідно створювати паралельні канали передачі сигналів. Задача створення мінімально необхідної кількості паралельних каналів досить чітко формалізується при використанні карт Карно (діаграм Вейча). Сутність такої формалізації полягає в тому, щоб будь-які сусідні клітини карти Карно були охоплені щонайменше одним багатовимірним кубом. Мінімізована таким чином логічна функція буде вільною від гонок.

Практично асинхронні схеми достатньо складні в проектуванні і поступово витісняються синхронними.

Приклад 5.3. Логічна функція задана картою Карно (рис. 5.14). Мінімізувати функцію для отримання диз'юнктивної нормальної форми, вільної від гонок.

x_3x_2 \ x_1x_0	00	01	11	10
00	$^{0}0$	$^{1}1$	$^{3}1$	$^{2}0$
01	$^{4}0$	$^{5}1$	$^{7}1$	$^{6}1$
11	$^{12}1$	$^{13}1$	$^{15}0$	$^{14}1$
10	$^{8}1$	$^{9}1$	$^{11}0$	$^{10}0$

Рис. 5.14. Логічна функція, що задана картою Карно

Розв'язання. Для забезпечення диз'юнктивної форми можна об'єднати наступні групи клітинок: група № 1 – **1, 3, 5, 7**; група № 2 – **8, 9, 12, 13** і група № 3 – **6, 14**. В результаті отримаємо функцію:

$$y = \overline{x_3}\, x_0 + x_3\, \overline{x_1} + x_2\, x_1\, \overline{x_0}\,.$$

Функція буде мінімальною, але не вільною від гонок, оскільки такі сусідні клітини, як **6, 7**; **1, 9**; **12, 14** не охоплюються багатовимірними кубами. Виконання умови відсутності гонок приводить до необхідності об'єднання вказаних клітин, що може бути забезпечено двома **2**-кубами **(7, 6)** і **(12, 14)** і одним **4**-кубом **(1, 5, 13, 9)**. В результаті отримаємо функцію:

$$y = \overline{x_3}\, x_0 + x_3\, \overline{x_1} + x_2\, x_1\, \overline{x_0} + \overline{x_3}\, x_2\, x_1 + x_3\, x_2\, \overline{x_0} + \overline{x_1}\, x_0\,,$$

яка буде вільною від гонок.

Третій спосіб – *урахування мінімального часу затримок* – знаходить використання при проектуванні таких цифрових схем, які потім виготовлятимуться на одному кристалі. Це пов'язано з тим, що проектант паралельно зі схемотехнікою закладає й технологічні особливості виготовлення схеми з необхідними параметрами. В результаті у проект можна закласти необхідні часові затримки в окремих каналах або схеми. Використання таких прийомів у дискретній цифровій схемотехніці вимагає великого досвіду і уваги.

У будь-якому випадку, після проектування цифрових схем їх аналізують на предмет можливої появи перешкод в результаті гонок. На практиці з цією метою використовуються спеціальні комп'ютерні програми.

Ще одним прикладом появи перешкод завдяки гонкам є прийом сигналів з недостатньо крутими фронтами. Така ситуація має місце тоді, коли подвійний сигнал подається на паралельні канали з різними типами мікросхем. У такому випадку в деякому інтервалі рівнів вхідного сигналу для мікросхеми одного типу сигнал відповідатиме рівню логічного нуля, а для іншого – як логічної одиниці. Це приведе до появи хибних сигналів. Це явище називається *„гонками по виходу"*. Для боротьби з ним використовуються спеціальні прийоми попередньої обробки вхідних сигналів.

Перелік посилань

1. Рябенький В.М., Жуйков В.Я., Гулий В.Д. Цифрова схемотехніка: Навч. посібник. – Львів: Новий світ – 2000, 2009, – 736 с.

Лекція № 6

Тригери. Тригерні схеми

У загальному вигляді тригерна схема має вигляд, що відповідає рис. 6.1.

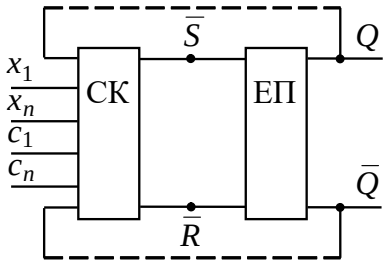


Рис. 6.1. Тригерна схема

У цифрових пристроях використовується велика кількість різноманітних тригерів, але всі вони мають у своєму складі *елемент пам'яті* (*ЕП*) та *схему керування* (*СК*), яка має два виходи – прямий Q та інверсний $\overline{Q}$, котрі можуть приймати значення **1** або **0**. Керування елементом пам'яті відбувається за допомогою входів S (*Set* – установлення стану $Q = 1$, $\overline{Q} = 0$) та R (*Reset* – повернення до початкового стану $Q = 0$, $\overline{Q} = 1$). Логічні значення сигналів залежать від значень вхідних логічних змінних $x_1 \ldots x_n$, сигналів синхронізації $c_1 \ldots c_n$, станів виходів елемента пам'яті Q та $\overline{Q}$ і логіки роботи схеми керування. У залежності від призначення схеми змінюється її логічна функція, кількість входів x_n, кількість та характер входів синхронізації.

У зарубіжній інженерній практиці всі тригерні схеми розділяються на дві групи.

Перша з них – *flip-flop* – характеризується тим, що вибірка вхідних сигналів і відповідна зміна виходів визначається в моменти дії тактових часових сигналів (синхронні тригери).

Особливість другої групи схем – *latch* – полягає в тому, що вони змінюють свій стан при зміні вхідних сигналів незалежно від наявності чи відсутності часових тактових сигналів.

Загальна характеристика тригерних схем

Особливість схемотехніки тригерних схем, порівняно з комбінаційною схемотехнікою, полягає в тому, що їх виходи з'єднані зі входами. Ця ситуація часто вимагає застосування допоміжних заходів для протидії електричним перешкодам $U_П$, що виникають на виходах різноманітних електронних пристроїв. Виникнувши на лініях зв'язку і на відповідних входах тригера, перешкода може викликати зміну його стану. Таким чином, тригер може перетворювати випадкові перешкоди у постійно діючий сигнал [1]. Для виключення подібної ситуації виходи тригера підключаються до навантаження або лінії зв'язку через буферні елементи, а для підвищення швидкодії цих елементів їх з'єднують також за схемою тригера (рис. 6.2).

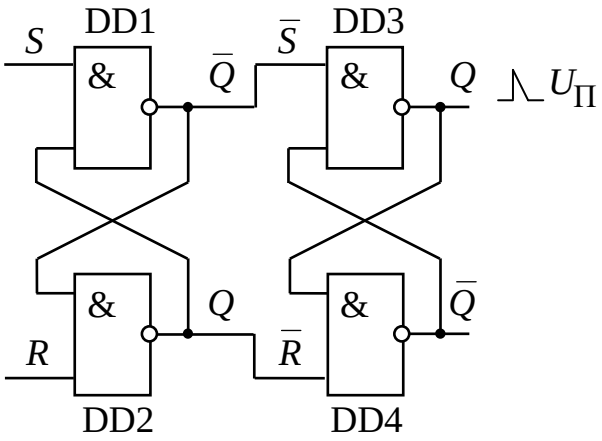


Рис. 6.2. З'єднання елементів за схемою тригера

У такій схемі на входах буферних елементів DD3, DD4 постійно діє один з потенційних сигналів з виходів DD1, DD2. Тому, якщо навіть імпульсна перешкода $U_П$ і призведе до тимчасової зміни станів DD3, DD4, то після її зникнення вхідний сигнал поверне мікросхеми DD3, DD4 у початковий стан.

Незважаючи на простоту, *RS*-тригери в чистому вигляді не використовуються для збереження даних у системах обробки інформації, адже для запису інформації необхідно мати дві послідовності сигналів: спочатку подавати *R*-сигнали для переведення тригера до нульового стану, а потім інформаційний та синхронізуючий – для запису даних.

Вхідні сигнали в залежності від призначення поділяють на: *інформаційні*, *дозволяючі*, *командні* або *синхронізуючі*.

Відповідно до типів сигналів, поділяють і входи складних схем тригерів, які позначають літерами абетки згідно з табл. 6.1.

Через те, що функціональні властивості тригера залежать від вхідної логіки, назви інформаційних входів переносяться на всю схему. Тому, крім *RS*-тригерів, широке використання знаходять *D*-, *DV*-, *JK*-, *T*-, *TV*-тригери, особливості яких розглядаються нижче.

Табл. 6.1. Умовні позначення та призначення входів та виходів тригерних схем

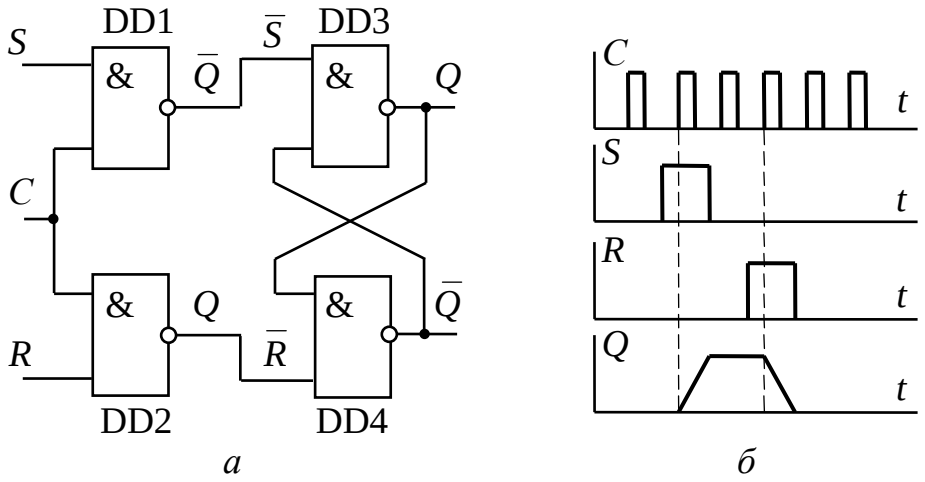
Позначення входу	***Призначення***
S	Вхід установки тригерів у стан *Q* = 1
R	Вхід установки тригерів у стан *Q* = 0
J	Вхід установки *JK*-тригерів у стан *Q* = 1
K	Вхід установки *JK*-тригерів у стан *Q* = 0
D	Вхід завантаження інформації у *D*-тригер
T	Тактовий вхід *T*-тригерів
V	Підготовчий вхід дозволу прийому інформації
C	Вхід синхронізації. Виконавчий вхід прийому інформації

Незалежно від функціональних властивостей, всі тригери за способом введення інформації поділяють на дві групи – *асинхронні* та *синхронні*.

Асинхронні тригери, подібно до розглянутого вище *RS*-тригера, змінюють свій стан безпосередньо після зміни інформаційних сигналів.

Для спрацювання *синхронних тригерів* наявності інформаційних сигналів недостатньо. Допоміжно необхідна ще наявність синхроімпульсу (тактового імпульсу) на спеціальному тактовому *C*-вході (рис. 6.3, ***а***). Тактові імпульси створюються за допомогою тактових генераторів і використовуються всім цифровим пристроєм. У синхронних схемах інформаційні сигнали (тобто дані, що записуються в тригер) готуються і подаються на відповідні входи раніше,

ніж синхросигнали (рис. 6.3, ***б***). При цьому для надійного запису інформації необхідними є відповідні часові співвідношення.



а *б*

Рис. 6.3. Схема синхронного *RS*-тригера

У довідковій літературі на будь-який тригер даються два параметри:

- час затримки розповсюдження сигналу на вмикання $t_{З\,01}$;
- час затримки розповсюдження сигналу на вимикання $t_{З\,10}$.

Інтервали $t_{З\,01}$ і $t_{З\,10}$ – це мінімальні інтервали часу, що визначають мінімальну тривалість вхідних імпульсів, які повинні подаватися на тригер.

При наявності окремих інформаційних входів і входів синхронізації задаються часові параметри окремо для асинхронних входів установки $\overline{S}$ і скидання $\overline{R}$ і окремо по входу синхронізації *C*. Час затримки по входах *J, K, D* не задається, оскільки вважається, що інформація на них повинна бути попередньо підготовлена.

Для синхронних тригерів часові інтервали характеризуються ще двома параметрами, які є характерними не тільки для тригерних схем.

Перший з них – *час підготовки* $t_{П}$ (в англомовній літературі – *Setup time*, позначається як t_{SU}). Це мінімальний інтервал часу, протягом якого інформаційний сигнал на вході тригера повинен залишатися незмінним до подачі синхросигналу. За цей час всі перехідні процеси установлення інформаційного сигналу повинні завершитися.

Другий часовий параметр – *час затримки* (*підтримки*) $t_З$ (в англомовній літературі – *Hold time,* t_H). Це мінімальний інтервал часу з моменту подачі синхросигналу, протягом якого інформаційний сигнал повинен залишатися незмінним. Цей інтервал часу тісно пов'язаний з вищезгаданими інтервалами часу $t_{З\,01}$ і $t_{З\,10}$, оскільки необхідно, щоб виконувалися наступні нерівності:

$$t_З > t_{З\,01};\ t_З > t_{З\,10}. \qquad (6.1)$$

В окремих випадках в синхронних цифрових схемах використовуються дві зміщені в часі послідовності синхроімпульсів.

RS-тригер є здебільшого лише елементом пам'яті для різних типів тригерних систем. Головна ж роль у формуванні властивостей системи належить схемі керування, яка перетворює вхідні сигнали $x_0 \ldots x_n$, $C_0 \ldots C_n$ у сигнали керування елементом пам'яті.

Недоліком асинхронних тригерів, який обмежує їх використання у швидкодіючих пристроях, є незахищеність перед небезпечними "змаганнями" ("гонками") сигналів, сутність яких полягає в тому, що сигнали, які подаються на інформаційні входи тригера, проходять по різних колах, через різну кількість логічних елементів. У зв'язку з цим можливі часові зміщення сигналів, величини яких можуть змінюватись у широких межах. Це може призвести до запису хибної інформації в тригери. Синхронізація дозволяє цей недолік ліквідувати. До того ж, синхронізація дає можливість підвищити перешкодостійкість тригера, адже значно зменшується інтервал часу доступу до його інформаційного входу (входів).

Слід зазначити, що тактовий сигнал є керуючим імпульсом, адже вважається, що до його приходу зміна сигналів на інформаційних входах повинна бути завершена.

Синхронні тригери поділяють на схеми зі *статичним* та з *динамічним керуванням*. Характерним для статичних тригерів є те, що при активному стані тактового входу (наявність одиничного рівня потенціалу) вони поводять себе як асинхронні. Ця властивість у багатьох випадках є суттєвим недоліком

синхронних тригерів зі статичним тактовим входом, адже може призводити до порушень у роботі при наявності перешкод.

Цей недолік усунений у *тригерах з динамічним тактовим входом*, а також у *тригерах з двоступінчатим керуванням*. Тригери з динамічним керуванням, відповідно до схеми виконання, реагують на перепад напруги від нуля до одиниці (за фронтом) або від одиниці до нуля (за спадом (зрізом) імпульсу). Останні, які звуться *тригерами з внутрішньою затримкою*, дуже широко застосовуються в системах обробки інформації, адже дозволяють за фронтом тактових імпульсів зчитувати інформацію, а за зрізом – записувати.

***RS*-тригери**

Схема *RS*-тригера, зібраного на логічних елементах **2АБО-НІ**, приведена на рис. 6.4, ***а***. Завдяки симетричному вигляду, вона часто називається *симетричним RS-тригером* (але зовнішня симетрія не означає симетрію електричних режимів роботи логічних елементів DD1, DD2). На рис. 6.4, ***б*** приведено його умовне позначення.

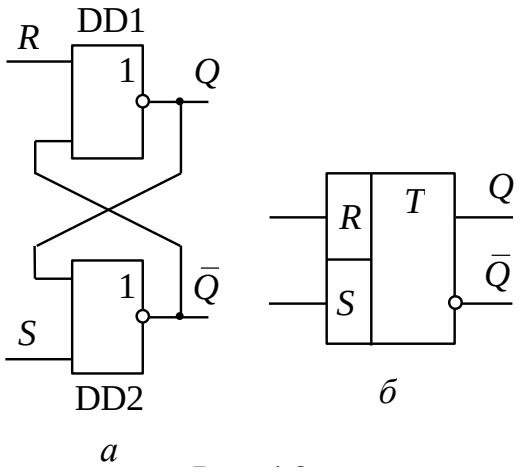


Рис. 6.4. Схема *RS*-тригера, зібраного на логічних елементах 2АБО-НІ, та його умовне позначення

Стан тригера часто ототожнюється з сигналом на прямому виході *Q*.

Особливість тригерних схем, на відміну від комбінаційних, полягає в тому, що будь-який із станів є стійким при відсутності вхідних сигналів.

Припустимо, що $R = S = 0$, а $\overline{Q} = 1$. Тоді на виході елемента DD1 зберігатиметься значення $Q = 0$, яке забезпечує значення виходу $\overline{Q} = 1$ елемента DD2, тобто підтверджує нульовий стан тригера.

Припустимо, що у цьому стані на входи тригера подано комбінацію потенційних сигналів $R = 0$, $S = 1$. Тоді вихід елемента DD2 прийме значення $\overline{Q} = 0$, а вихід елемента DD1 прийме значення $Q = 1$. Тобто вказаною комбінацією вхідних сигналів ми запишемо сигнал $S = 1$ у тригер як один біт інформації. Змінити стан тригера на попередній можливо протилежною комбінацією вхідних сигналів $R = 1$, $S = 0$. Якщо ми повторимо подачу цієї комбінації вхідних сигналів при $Q = 0$, $\overline{Q} = 1$, то стан тригера не зміниться. Таким чином, стан виходів тригера в момент часу, наступний за моментом подачі вхідних сигналів, залежить не тільки від комбінації вхідних сигналів, але й від попереднього стану його виходів. Тому для аналізу схеми необхідне розподілення станів входів та виходів тригера у часі. Для цього попередні стани позначаються індексом n: S_n, R_n, Q_n. Наступний стан тригера, в який той переходить у результаті дії комбінації вхідних сигналів з урахуванням значень виходів у n-й момент часу, позначається як Q_{n+1}, $\overline{Q}_{n+1}$. Завдяки такому розподіленню станів та вхідних сигналів у часі з'являється можливість використовувати логічні функції для запису алгоритму роботи тригерів.

Повна таблиця станів *RS*-тригера (Табл. 4.1) описує особливості його роботи. Звернемо увагу на те, що при $S_n = R_n = 1$, незалежно від Q_n, стан виходів тригера є невизначеним. Дійсно, якщо подати на обидва входи таку комбінацію сигналів, то на обох виходах з'являться логічні нулі ($Q_{n+1} = \overline{Q}_{n+1} = 0$). Але якщо ці сигнали одночасно зняти, задавши $Q_{n+1} = R_{n+1} = 0$, то стан виходів буде невизначеним. Це пов'язано з тим, що після зняття сигналів у кожному з логічних елементів буде протікати перехідний процес зміни станів, і результат його залежатиме від швидкодії елементів. Остання є величиною невизначеною. Тому розглянута комбінація вхідних сигналів

називається *невизначеною комбінацією*, а перехід від такої комбінації до нульової – *забороненим переходом*. Фактично це означає, що при проектуванні цифрових пристроїв необхідно приймати міри для виключення подібних ситуацій.

Табл. 6.2. Повна таблиця станів *RS*-тригера

R_n	S_n	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	x
1	1	1	x

Представивши табл. 6.2 у формі карти Карно (рис. 6.4), помічаємо наступну особливість логічної функції: при значеннях $S_n = R_n = 0$ маємо $Q_{n+1} = Q_n$, а для решти комбінацій входів значення виходу Q_{n+1} не залежить від значення Q_n. Це дає можливість мінімізувати табл. 6.2 (див. табл. 6.3) та одержати характеристичне рівняння тригера:

$$\begin{aligned} Q_{n+1} &= S_n + Q_n \cdot \overline{R_n}\,; \\ \overline{Q}_{n+1} &= R_n + \overline{Q}_n \cdot \overline{S_n}\,. \end{aligned} \qquad (6.2)$$

Q_n \ R_nS_n	00	01	11	10
0	0	1	x	0
1	1	1	x	0

Рис. 6.4. Карта Карно

Табл. 6.3. Мінімізована таблиця станів *RS*-тригера

R_n	S_n	Q_{n+1}
0	0	Q_n
0	1	1
1	0	0
1	1	x

Перетворюючи рівняння (6.2) в базиси логічних функцій **І-НІ**, одержуємо:

$$\begin{aligned} Q_{n+1} &= \overline{\overline{S}_n \cdot \overline{(Q_n \cdot \overline{R}_n)}}; \\ \overline{Q}_{n+1} &= \overline{\overline{R}_n \cdot \overline{(\overline{Q}_n \cdot \overline{S}_n)}}. \end{aligned} \qquad (6.3)$$

Цим рівнянням відповідає схема рис. 6.9. Перше з них виконане на DD1, а друге на DD2. Цей тригер працює в інверсних кодах.

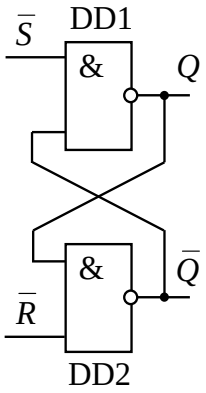
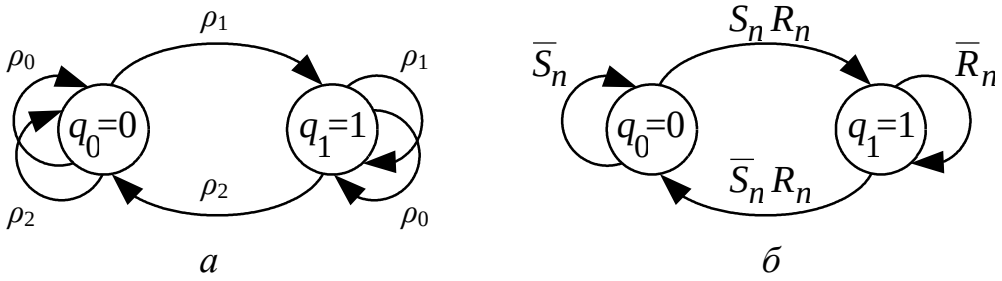


Рис. 6.5. Мінімізована схема *RS*-тригера

Зміна станів тригерних схем при подачі різних послідовностей вхідних сигналів зображується за допомогою графа переходів (рис. 6.6).



а *б*

Рис. 6.6. Графи переходів

Виходячи з таблиці станів, можемо розглядати тригер як пристрій з одним виходом Q (Q_{n+1}), оскільки другий вихід $\overline{Q}$ ($\overline{Q}_{n+1}$) є лише інверсією першого. Вихід Q може приймати два значення. Позначимо їх $q_0 = 0$ і $q_1 = 1$ як дві вершини графа. Вектор впливаючих сигналів позначимо буквою ρ. Він може приймати, у відповідності до табл. 6.3, значення $\rho_0 = \overline{S}_n\ \overline{R}_n$, $\rho_1 = S_n\ \overline{R}_n$, $\rho_2 = \overline{S}_n\ R_n$. Вектор ρ_0 не може змінювати стани тригера. Таку особливість на

граф-схемі (рис. 6.6, ***а***) зобразимо у вигляді дуги, що виходить з вершин і замикається на них. Вектор ρ_1 переводить тригер у стан $q_1 = 1$ і на граф-схемі зображується у вигляді дуги, що виходить з вершини q_0 і закінчується в q_1. Якщо тригер знаходиться в стані q_1, то впливаючий сигнал ρ_1 не змінить стану тригера. На граф-схемі це дуга, що замикається на стані q_1. Аналогічно, впливаючий сигнал ρ_2 переводить тригер зі стану q_1 в q_0, а в стані q_0 не впливає на нього. Оскільки перехід з q_1 в q_1 забезпечується двома сигналами з однаковим результатом, то поєднаємо його як:

$$\rho_{01} = \rho_1 + \rho_0 = S_n \overline{R_n} + \overline{S_n}\ \overline{R_n} = \overline{R_n}\,. \qquad (6.4)$$

Аналогічно маємо:

$$\rho_{02} = \rho_0 + \rho_2 = \overline{S_n}\ \overline{R_n} + \overline{S_n}\ R_n = \overline{S_n}\,. \qquad (6.5)$$

В результаті граф-схема переходів *RS*-тригера прийме вигляд рис. 6.10, ***б***.

Графи переходів відповідають часовим діаграмам, що ілюструють роботу тригера у часі. Крім того, часові діаграми дають більш детальну характеристику перехідним процесам у схемі.

На рис. 6.7 наведені часові діаграми роботи тригера. Елементи тригера перемикаються послідовно. Запуск тригера відбувається за фронтом вхідного сигналу *S* у момент часу, коли його значення досягне порогового рівня спрацьовування логічного елемента DD2. Вихідний стан $\overline{Q} = 1$ логічного елемента DD2 змінюється, і в момент часу, коли потенційний рівень спаду сигналу $\overline{Q}$ зменшиться до рівня порогової напруги елемента DD1, останній починає перемикатись. На рис. 6.7 послідовність перемикання показана стрілками.

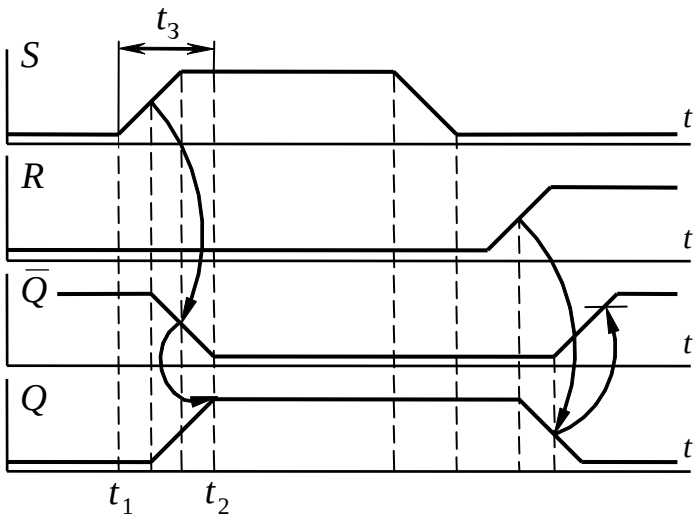


Рис. 6.7. Часові діаграми роботи тригера

Інтервал часу перемикання визначається з моменту t_1 початку дії сигналу S до моменту t_2 – завершення перемикання логічного елемента DD1. Цей інтервал часу t_3, що зветься *часом затримки на перемикання*, дає можливість оцінити мінімальну тривалість вхідних сигналів, при якій гарантовано будуть змінюватися стани тригера.

З опису процесу перемикання витікає, що t_3 тригера визначається сумою часових затримок двох базових логічних елементів, на яких він виготовлений.

RS-тригери у багатьох випадках використовуються як самостійні пристрої в тих ситуаціях, коли одним сигналом необхідно встановити якусь умову, а іншим – її зняти. Таке їх використання передбачається у контролерах і мікроконтролерах у складі регістрах ознак. Здебільшого *RS*-тригери використовуються у складі більш складних схем тригерів, модулів пам’яті. Вони знаходять широке використання в пристроях електронної автоматики.

D-тригери

Функціональна особливість тригерів цього типу полягає в тому, що сигнал на виході Q в (n +1)-ому такті повторює значення сигналу на вході D в n-му такті [1].

Роботу *D*-тригера пояснює таблиця його станів (табл. 6.4), з якої витікає, що при $C_n = 0$ значення $Q_{n+1} = Q_n$, а при $C_n = 1$ $Q_{n+1} = D_n$. Зобразивши логічну функцію $Q_{n+1} = f(Q_n, C_n, D_n)$ у вигляді карти Карно (рис. 6.8), після мінімізації знаходимо:

$$Q_{n+1} = C_n\, D_n + \overline{C_n}\, Q_n. \qquad (6.6)$$

Таблиця 6.4. Таблиця станів *D*-тригера

C_n	D_n	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

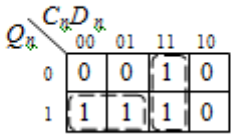


Рис. 6. 8. Карта Карно

Враховуючи той факт, що друга складова рівняння (6.6) характеризує лише режим зберігання інформації, закон функціонування *D*-тригера виразиться формулою:

$$Q_{n+1} = C_n\, D_n. \qquad (6.7)$$

З (6.7) витікає, що при $C_n = 1$ сигнал $Q_{n+1} = D_n$ і будь-яка зміна вхідного сигналу одразу з'явиться на виході. Тому ці тригери часто називають *"прозорими"*.

Виходячи з умови ліквідації гонок, з карти Карно отримуємо:

$$Q_{n+1} = C_n\, D_n + \overline{C_n}\, Q_n + D_n\, Q_n.$$

Якщо цю формулу перетворити під базис **І-НІ**, матимемо:

$$Q_{n+1} = \overline{\overline{C_n D_n} \cdot \overline{\overline{C_n} Q_n} \cdot \overline{D_n Q_n}},$$

яка є законом функціонування цифрового пристрою, відомого як *D-тригер Ерла*. Схема, побудована на основі отриманої формули, відрізняється від інших найбільшою реальною швидкодією і використовується в різноманітних пристроях обчислювальної техніки.

Граф переходів у відповідності до табл. 6.4 зображений на рис. 6.9.

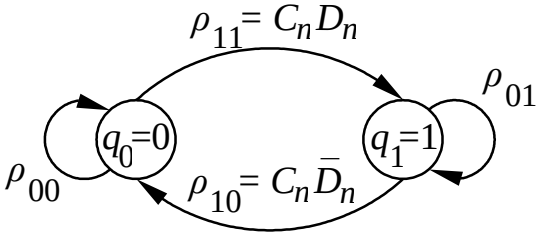


Рис. 6.9. Граф переходів

Перехід зі стану $q_0 = 0$ в стан $q_1 = 1$ відбувається лише при $\rho_{11} = C_n D_n$, а зворотній – тільки при $\rho_{10} = C_n \overline{D_n}$. При $q_0 = 0$ стан тригера не змінюється при:

$$\rho_{00} = \overline{C_n}\,\overline{D_n} + C_n\,\overline{D_n} + \overline{C_n}\,D_n = \overline{D_n} + \overline{C_n}. \qquad (6.8)$$

При $q_1 = 1$ стан тригера залишається незмінним за умови:

$$\rho_{01} = \overline{C_n}\,\overline{D_n} + C_n\,D_n + \overline{C_n}\,D_n = \overline{C_n} + D_n. \qquad (6.9)$$

D-тригер може бути створений на базі синхронного *RS*-тригера, якщо на його інформаційні входи одночасно подати взаємно інверсні сигнали *D* і $\overline{D}$.

На рис. 6.10 приведені два варіанти *D*-тригера (***а***, ***б***) та його умовне позначення (***в***).

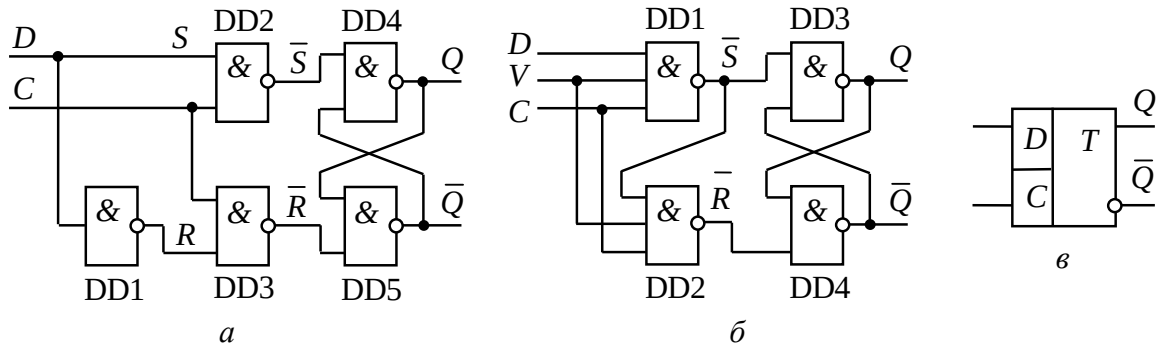


Рис. 6.10. Два варіанти *D*-тригера та його умовне позначення

При $C = 1$ і $D = 0$ на входах логічного елемента DD3 (рис. 6.10, ***а***) будуть сигнали високого рівня, що приведе до появи на вході $\overline{R}$ внутрішнього *RS*-тригера (ЛЕ DD4, DD5) сигналу низького рівня $\overline{R} = 0$, а на виході значення $Q = 0$ і $\overline{Q} = 1$. При появі сигналу високого рівня лише на вході *D* буде забезпечуватись значення $\overline{S} = 1$ і $\overline{R} = 1$, при яких попередня інформація внутрішнього *RS*-тригера залишається незмінною. Такий режим відображається на часових діаграмах, приведених на рис. 6.11, ***а***, інтервалом $t_1 - t_2$.

При одночасній дії синхросигналу і сигналу високого рівня на вході *D* (момент часу t_2) на виході DD2 встановиться рівень $\overline{S} = 0$, що приведе до зміни стану внутрішнього тригера, при якому $Q = 1$ і $\overline{Q} = 0$. Такий стан утримуватиметься до того моменту, поки не відбудеться зміна рівня сигналу $\overline{R}$ з **“1”** до **“0”**, що призведе до повернення тригера в початковий стан. Оскільки значення $\overline{R} = 0$ має місце лише при $C = 1$, то перехід тригера в початковий стан матиме місце при появі наступного тактового імпульсу.

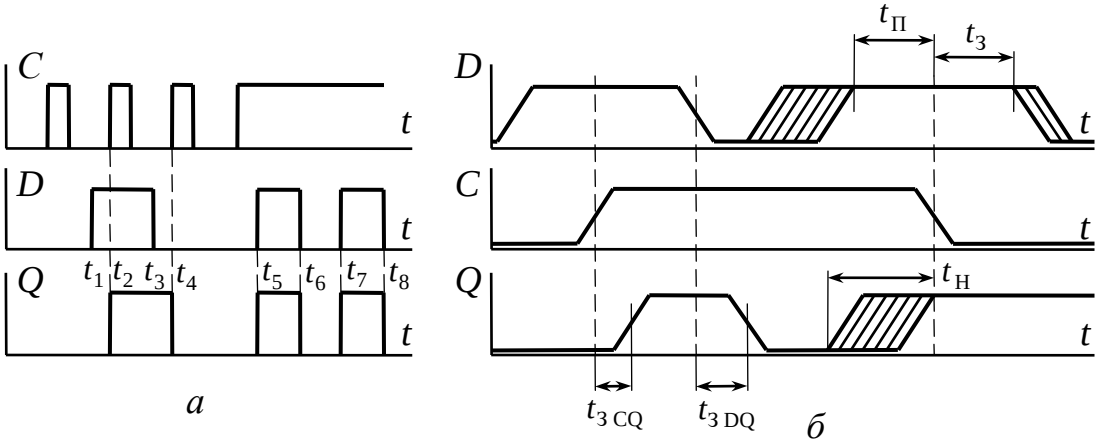


Рис. 6.11. Часові діаграми

У тригері, схема якого зображена на рис. 6.10, ***б***, така ж функція забезпечується тим, що один з входів DD2 приєднаний до виходу DD1. При значенні дозволяючого входу $V = 1$ тригер, приведений на рис. 6.10, ***б***, працює аналогічно вище описаному.

При практичному використанні *D*-тригерів у них фіксується значення сигналу *D* у момент відповідної зміни сигналу *C*. Тому в літературі такі тригери часто називають *фіксаторами* (*Latch*).

У загальному випадку логічна функція, що описує роботу *D*-тригера, має вигляд:

$$Q_{n+1} = V\,(\overline{C_n}\;Q_n + C_n\;D_n) + \overline{V}\;Q_n. \qquad (6.10)$$

З точки зору логіки роботи *D*-тригера, він затримує проходження сигналу, що поступає на *D*-вхід, на один такт періоду синхросигналу (*Delay* – затримка). Це добре відображають часові діаграми, приведені на рис. 6.11, ***а***.

Друге призначення *D*-тригера – зберігати дані (*Datas*), що надходять по *D*-входу. Синхросигнали в цьому випадку відіграють роль команди “Запис у тригер”. Особливість схеми проявляється в режимі $C = 1$. У цьому випадку будь-які зміни (t_5, t_6, t_7, t_8) на вході *D* приведуть до відповідних змін на виході *Q*. Тобто схема виконуватиме просто функцію повторювача сигналів, що присутні на вході *D*.

Іноді в *D*-тригер вводять вхід *R* (рис. 6.10, ***б***). Але використовувати його можна лише при $C = 0$. При $C = 1$ можлива поява забороненої ситуації $\overline{S} = 0$, $\overline{R} = 0$.

У деяких схемах вводять також *V*-вхід (від англ. *Valve* – клапан), що дає можливість вимикати тригер. Такі схеми називаються *DV-тригерами*.

Якщо в схемі синхронізованого імпульсом *D*-тригера замінити вхід *D* на інверсний $\overline{D}$, то виходи *Q* та $\overline{Q}$ поміняються місцями. Така властивість може бути корисною при використанні мікросхем, в яких виведені лише прямі виходи (див. мікросхеми 555ИР22 та 1533ИР33).

При проектуванні схем з використанням *D*-тригерів необхідно, особливо для швидкодіючих пристроїв обробки інформації, строго дотримуватись часових співвідношень між сигналами на *C*- та *D*- входах. Ці співвідношення для схеми рис. 6.10, ***б*** наведені на часових діаграмах рис. 6.11, ***б***, побудованих з урахуванням станів невизначеності. Перехідний процес характеризується

значеннями затримок розповсюдження сигналу по двох каналах: $t_{З\ CQ}$ – від входу C до виходу Q при $D = const$, та $t_{З\ DQ}$ – від входу D до виходу Q при $C = 1$. Водночас кожен з каналів характеризується величинами затримок при перемиканні з **1** на **0** та навпаки.

Для надійного запису інформаційного D-сигналу перехідний процес у схемі, обумовлений зрізом C-сигналу, не повинен накладатись на перехідний процес, обумовлений зміною стану D-входу. Це означає, що будь-які зміни стану D-входу повинні закінчитися за деякий час до зрізу C-сигналу (час підготовки $t_П$) та можуть знову початись не раніше ніж через час підтримки $t_Н$. Вони оцінюються як $t_П = (1 – 2)\, t_Н$; $t_З = (0 – 1)\, t_Н$, де $t_Н$ – час невизначеності стану логічного елемента при перемиканні.

Необхідність урахування вказаних інтервалів характерна для всіх функціональних вузлів, що мають вхід синхронізації. Крім цього, існує мінімально допустима тривалість синхроімпульсу, що забезпечує відсутність некоректного розвитку перехідних процесів від фронту та зрізу імпульсів.

У D-тригері розв’язуються проблеми $S = R = 1$, але якщо інтервал часу між спадами D і C сигналів досить короткий, то внутрішній RS-тригер може опинитись у метастабільному стані, з якого, як вказувалось вище, він може перейти у непередбачений стан.

JK-тригери

Цей тип тригерів за логікою роботи подібний до RS-тригерів, але, на відміну від них, не має невизначених переходів [1].

Скорочена таблиця станів (табл. 6.5) пояснює наведену різницю, якщо прийняти, що S-вхід відповідає J-входу, а R-вхід – відповідно, K-.

З таблиці витікає, що поява комбінації $J = K = 1$ у кожному такті призводить до зміни стану тригера на протилежний.

Таблиця 6.5. Скорочена таблиця станів *JK*-тригера

J_n	K_n	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q}_n$

На відміну від *RS*-тригерів, схеми *JK*-тригерів виготовляються тільки синхронними.

Однотактні тригери відрізняються наявністю зворотних зв'язків з виходів на входи (рис. 6.12), а також елементами часової затримки (елементи DD3, DD4). Стан виходів *JK*-тригера залежить не тільки від сигналів на входах *J* та *K*, але й від логічно пов'язаних з ними сигналів з виходів *Q* та $\overline{Q}$.

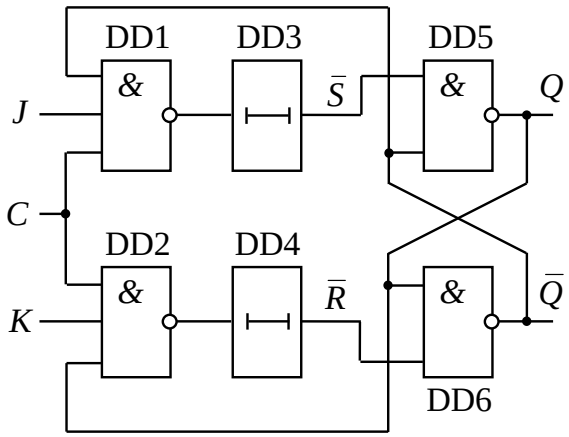


Рис. 6.12. Схема однотактного *JK*-тригера

Робота асинхронного *JK*-тригера описується наступним характеристичним рівнянням:

$$Q_{n+1} = J_n \overline{Q}_n + \overline{K}_n Q_n. \qquad (6.11)$$

Для синхронного *JK*-тригера є справедливим рівняння:

$$Q_{n+1} = C(J_n \overline{Q}_n + \overline{K}_n Q_n) + \overline{C} Q_n. \qquad (6.12)$$

При відсутності вхідних сигналів, тобто при $J_n = K_n = 0$ на виходах DD1 і DD2 зберігаються високі рівні сигналів, які забезпечують режим зберігання для

внутрішнього *RS*-тригера (мікросхеми DD5 і DD6). Низький рівень сигналу на виході DD1 або DD2, який може змінити стан *RS*-тригера, забезпечується лише при високому рівні на всіх трьох входах відповідного логічного елемента. Оскільки на один з входів ЛЕ DD1 і DD2 сигнал подається з діагонального виходу тригера, то низький рівень сигналу можливий лише на одному з виходів керованої логіки. Цим і досягається усунення проблеми невизначеності виходу при $J_n = K_n = 1$. Тому при появі такої комбінації вхідних сигналів тригер змінить свій стан на протилежний, незалежно від того, в якому стані він перебував раніше. Дійсно, якщо $Q_n = 0$, $\overline{Q}_n = 1$, то на виході DD1 з'явиться логічний нуль, який по входу $\overline{S}$ змінить стан тригера, і Q зміниться на **"1"**. Аналогічна ситуація матиме місце при $Q_n = 1$, $\overline{Q}_n = 0$. У цьому випадку логічний нуль з'явиться на виході DD2, що по входу $\overline{R}$ перекине *RS*-тригер у нульовий стан.

Елементи DD3, DD4 часової затримки у цій схемі відіграють роль стабілізаторів станів тригера і безпосереднього впливу на його функціональні властивості не здійснюють. Вони створюють часову затримку між моментом подачі вхідної інформації $J_n \overline{Q}_n$ або $K_n Q_n$ та початком формування вихідного стану Q_{n+1} та $\overline{Q}_{n+1}$. Без цих елементів виникає можливість генерації коливань в зв'язку з тим, що з кожною зміною вхідних сигналів на входах створювалася б комбінація, яка викликала б нову зміну станів тригера. Для уникнення подібної ситуації необхідно, щоб затримка перевищувала інтервал дії тактових сигналів. Такі принципи використані в мікросхемах К531ТВ9, К555ТВ9 (зарубіжні аналоги яких – SN54112, SN74112).

Уникання можливості появи автогенерації коливань просто забезпечується в тригерах з двоступінчатим керуванням через те, що обидва ступені тактуються послідовно. На такому принципі побудовані тригери КР1533ТВ1 (аналоги західних фірм-виробників – SN5472, SN7472).

На рис. 6.13 приведені часові діаграми, що пояснюють особливості зміни станів при умові $J = K = 1$, при якому тригер змінює стан виходів на протилежний. Цей режим широко використовується в цифровій техніці і називається *Т-режимом* (від англ. *Toggle* – перевертатись).

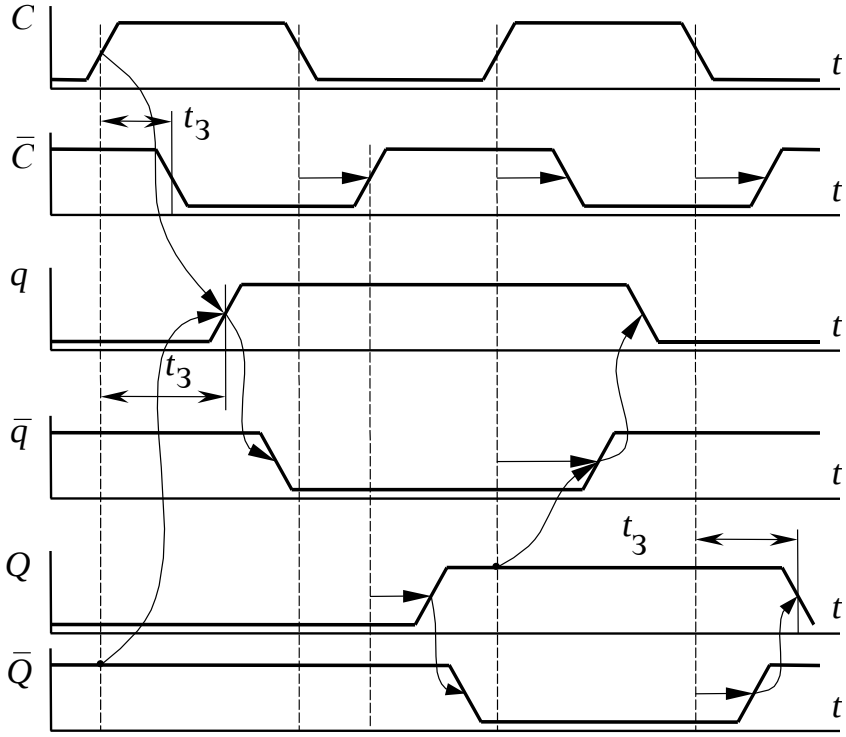


Рис. 6.13. Часові діаграми, що пояснюють особливості зміни станів

Таблиця 6.6. Скорочена таблиця функціонування *JK*-тригера

Режими	C_n	J_n	K_n	Q_{n+1}
Збереження	x	0	0	Q_n
Обнуління	⌊	0	1	0
Установка	⌊	1	0	1
Т-режим	⌊	1	1	$\overline{Q}_n$

Приведені на рис. 6.13 часові діаграми показують рекомендовані часові співвідношення між затримками сигналу синхронізації в інверторі DD9 і в *М*-тригері. В мікросхемі КР1533ТВ1 цей недолік усунений.

У схемах динамічних тригерів можливість появи гонок по входу завжди зв'язана з крутизною фронту або спаду *C*-імпульсу. Технічно в інтегральних мікросхемах ця проблема вирішується різними шляхами, але при проектуванні тригерних схем не слід їх піддавати необов'язковим випробуванням.

T- та *TV*-тригери

До тригерів *T*-типу відносяться такі схеми, які за сигналом на *T*-вхід переключаються у протилежний стан [1]. Це зазвичай тригери з динамічним *T*-входом або з динамічним *C*-входом і статичним *T*-входом. У зв'язку з тим, що вони легко можуть бути організовані на базі динамічних *D*- або *JK*-тригерів, як самостійні мікросхеми не виготовляються. У зв'язку з їх широким використанням в лічильниках імпульсів, *T*-тригери з динамічним *T*-входом часто називаються *тригерами з лічильним входом*, або *лічильними тригерами*. В залежності від характеру дії *T*-динамічного входу, вони часто поділяються на *T*-тригери, які спрацьовують за фронтом *T*-імпульсу, та $\overline{T}$-тригери, що спрацьовують за зрізом *T*-імпульсу.

У табл. 6.7 приведений перелік можливих станів обох типів тригерів. З таблиці витікає, що у *T*-тригерах, на відміну від раніше розглянутих схем, стан виходу Q_{n+1} залежить не від значень інформаційних сигналів, а визначається тільки їх станом у попередньому такті.

Таблиця 6.7. перелік можливих станів *T*-тригерів, які спрацьовують за фронтом *T*-імпульсу, та $\overline{T}$-тригерів, що спрацьовують за зрізом *T*-імпульсу

T-тригер		$\overline{T}$-тригер	
T_n	Q_n	T_n	Q_n
0	Q_n	0	Q_n
⌠	$\overline{Q}_n$	⌠	Q_n
⌡	Q_n	⌡	$\overline{Q}_n$
1	Q_n	1	Q_n

Рівняння роботи асинхронного тактованого *T*-тригера має вигляд:

$$Q_{n+1} = Q_n \overline{T_n} + \overline{Q_n}\, T_n . \qquad (6.13)$$

Для синхронного *T*-тригера маємо:

$$Q_{n+1} = (Q_n \overline{T} + \overline{Q_n}\, T)\, C + \overline{C}\, Q_n . \qquad (6.14)$$

Рівняння (6.13) за виглядом співпадає з виразом для логічної функції „**ВИКЛ. АБО**”, звідки витікає, що *T*-тригер виконує операцію складання за модулем **2** вхідної змінної та функції, що визначає вихідний стан тригера. Рис. 6.14 пояснює особливості роботи тригера з прямим керуванням. З рисунку витікає, що, спрацьовуючи за фронтом вхідних імпульсів, він ділить їх частоту на **2**.

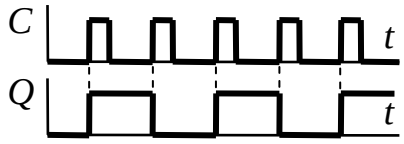


Рис. 6.14. Особливості роботи тригера з прямим керуванням

T-тригер легко можна одержати з *JK*- або *D*-тригерів. На рис. 6.15 приводяться приклади створення схем *T*-тригерів. Схеми рис. 6.15, ***а***, ***б*** зрозумілі без пояснень. *D*-тригер також можливо перетворити в *T*-тригер. Це пояснюється тим, що вихід Q_n приймає в такти (n + 1) значення, що було на вході *D* в *n*-ому такті. Тому з’єднання *D*-входу з виходом $\overline{Q}_n$ забезпечує зміну стану *Q* кожного разу при появі фронту *T*-входу.

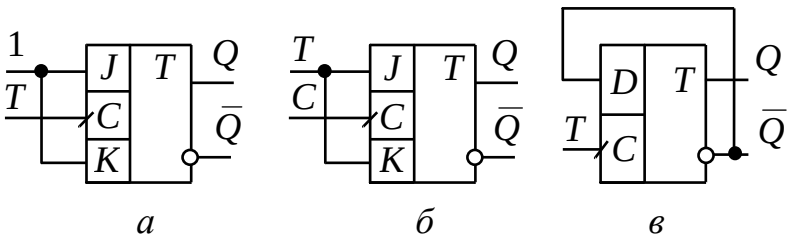


Рис. 6.14. Приклади створення схем *T*-тригерів

Тригери *TV*, окрім лічильного *T*-входу, мають ще й другий – керуючий – вхід для забезпечення дозволу прийому інформації. Рівняння *TV*-тригера має вигляд:

$$Q_{n+1} = (T_n \overline{Q_n} + \overline{T_n}\, Q_n)\, V_n + Q_n \overline{V_n}\,. \qquad (6.15)$$

Цей тригер також легко може бути одержаний з *JK*-тригерів (рис. 6.16).

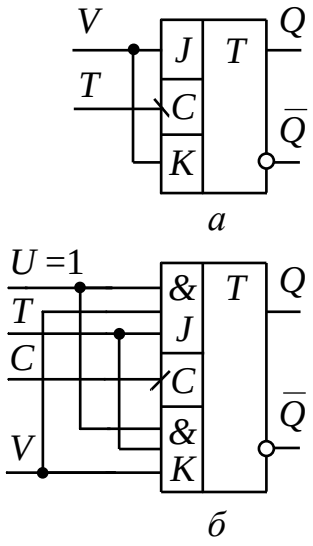


Рис. 6.16. *T*-тригер на основі *JK*-тригерів

У першому випадку (рис. 6.16, ***а***) маємо асинхронний *T*-тригер, у другому (рис. 6.16, ***б***) – синхронний.

Прикладом реалізації *T*-тригера на основі синхронного *RS*-тригера є схема, що представлена на рис. 6.17, ***а***.

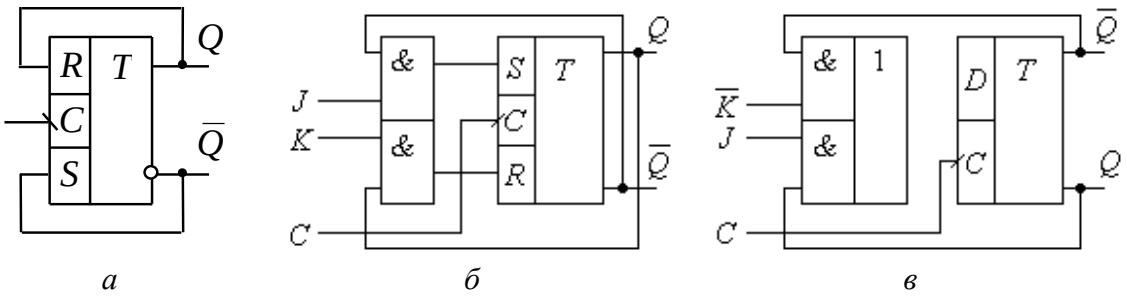


Рис. 6.17. Приклади реалізації *T*-тригера

З приведених прикладів видно, що достатньо мати будь-який один тип тригерів, щоб на його основі можна було створювати тригери інших типів.

Здебільшого такі перетворення витікають з порівняння алгоритмів роботи тригерів. Як приклад, використовуючи алгоритм роботи *RS*-тригера:

$$Q_{n+1} = S_n + Q_n \overline{R_n} \quad \text{і} \quad S_n \cdot R_n = 0$$

і підставивши $S_n = J \overline{Q_n}$ і $R_n = K Q_n$, отримаємо відому формулу:

$$Q_{n+1} = J \overline{Q_n} + \overline{K} Q_n,$$

тобто *JK*-тригер може бути побудований на основі *RS*-тригера.

Схема такого тригера приведена на рис. 6.28, ***б***.

Таким же шляхом може бути побудований *JK*-тригер на основі *D*-тригера. Якщо прийняти, що:

$$D_n = Q_{n+1} = J \overline{Q_n} + \overline{K} Q_n,$$

то отримаємо схему з *D*-тригером, що приведена на рис. 6.17, ***в***.

Перелік посилань

1. Рябенький В.М., Жуйков В.Я., Гулий В.Д. Цифрова схемотехніка: Навч. посібник. – Львів: Новий світ – 2000, 2009, – 736 с.

Лекція №7

Синхронізація в цифрових схемах

З підвищенням складності цифрових схем поява місцевих та загальних зв'язків у них призводить до того, що аналізувати і враховувати гонки в таких схемах стає практично неможливо. Радикальним вирішенням проблеми гонок є *синхронізація*. У практиці побудови систем синхронізації використовуються *однофазна* і *багатофазна синхронізація*, *одночастотна* і *багаточастотна* [1].

Розгляд систем синхронізації почнемо з двофазної системи, коли всі схеми синхронізуються двома послідовностями імпульсів C_1 та C_2 однієї частоти $f_{\text{Т}} = T_{\text{Т}}^{-1}$ та одного фазового зсуву T_{Φ}. Тривалість імпульсів двох послідовностей однакова і дорівнює T_i. Для симетричної двофазної синхронізації $T_{\text{Т}} = 2\,T_{\Phi}$. Для несиметричної $T_{\Phi 1} \neq T_{\Phi 2}$.

При побудові синхронних цифрових схем їх розподіляють на дві групи. В одну групу входять комбінаційні схеми з визначеною кількістю входів та виходів. До іншої групи входять схеми *D*-тригерів, які мають особливість зберігати записану інформацію протягом одного такту.

Узагальнена цифрова схема може бути приведена до структури, що зображена на рис. 7.1 (аналог конвеєрної обробки інформації в мікропроцесорах і мікропроцесорних системах).

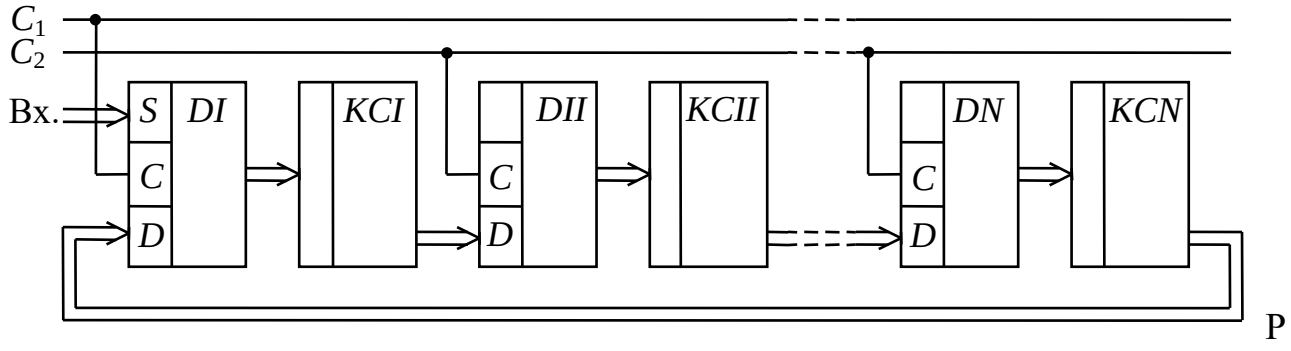


Рис. 7.1. Узагальнена цифрова схема системи *D*-тригерів

Схема включає в себе послідовно об'єднані групи *D*-тригерів, позначених на рис. 7.1 як *DI*, *DII*, …, *DN*, і комбінаційних схем, позначених *KCI*, *KCII*, …,

KC(*N*–1), *KCN*. Кожна з груп тригерів об’єднується за принципом синхронізації від одного синхроімпульсу і в загальному плані представляє собою паралельний регістр, виконаний на *D*-тригерах.

Кожна група *КС* включає в себе чисто комбінаційну схемотехніку, яка виконує одночасно ряд логічних функцій, приймаючи інформацію з виходів попереднього регістра пам’яті і передаючи на наступний, який синхронізується другим синхроімпульсом. Внутрішні зворотні зв’язки в групі комбінаційних схем відсутні.

Фізичну суть процесів у схемі та ідеологію проектування цифрових пристроїв з двофазною синхронізацією пояснює рис. 7.2.

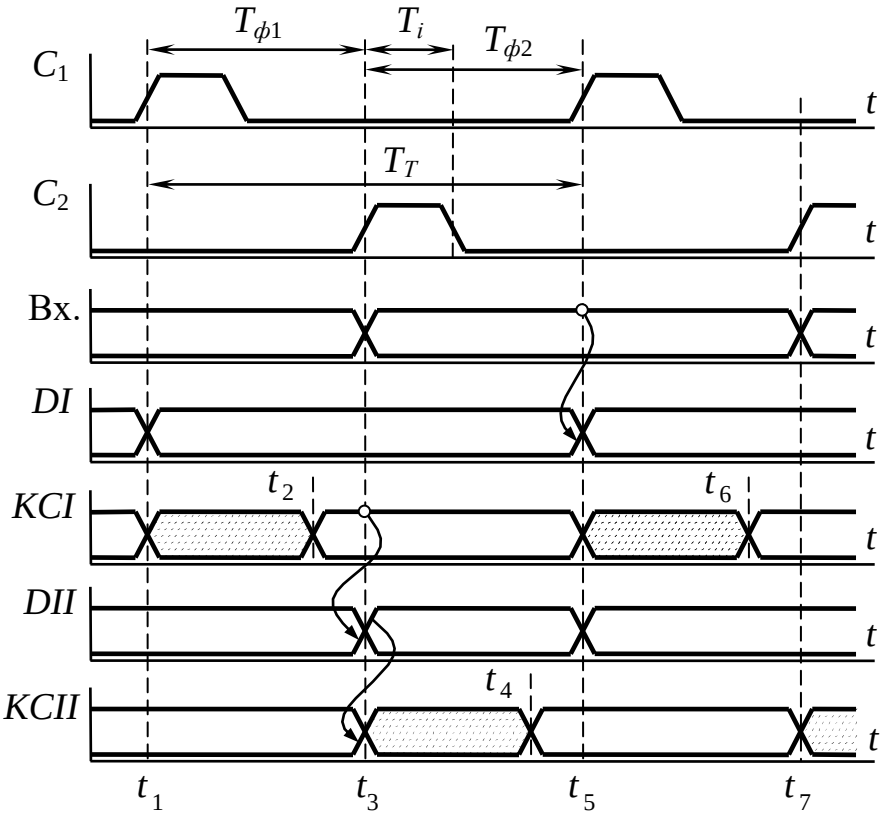


Рис. 7.2. Часові діаграми, що пояснюють фізичну суть процесів у схемі

За синхросигналом C_2 чергова група інформаційних сигналів записується в регістрову групу *D*-тригерів *N* і передається для обробки на вхід комбінаційної схеми *KCN*. Через деякий інтервал часу ця інформація з’явиться на вхідних шинах *D*-тригерів *DI*. У момент часу t_1 з’являється фронт синхросигналу C_1, за

яким як вхідна інформація, що подається на входи *S*, так і інформація з виходів *KCN* записуються в тригери *DI*. По закінченні перехідних процесів в тригерах *DI* на їх виходах з'являються сигнали, які починають опрацьовуватись комбінаційною схемою *KCI*. Як правило, в схемі мають місце паралельні шляхи розповсюдження сигналів, тому вихідні сигнали *KCI* спочатку є невизначеними, адже вони спотворюються перехідними процесами. На рис. 7.2 картина гонок в *KCI* відображена на інтервалі часу $t_1 - t_2$. Інтервал часу $t_1 - t_2$ перехідних процесів для *DII* не описаний через те, що всі тригери в цьому інтервалі часу закриті нульовим рівнем сигналу C_2. До моменту t_2 всі перехідні процеси закінчуються, сигнали на виході *KCI* фіксуються, і в інтервалі часу $t_2 - t_3$ ніякі стани в схемі не змінюються.

При подачі сигналу C_2 у момент t_3 установлені значення виходів *KCI* записуються в *DI* і по завершенні в них перехідних процесів подаються на входи послідуючої комбінаційної схеми *KCII*. Процеси гонок в *KCII* проходять в інтервалі часу $t_3 \ldots t_4$ і до моменту t_5 появи фронту синхроімпульсу С1 встановлюються незмінними. При появі С1 результати обробки сигналів в *KCII* перезаписуються в послідуючі регістрові схеми. Як результат, у синхронному пристрої йде циклічна багатоступенева обробка інформації в комбінаційних схемах, при якій комбінаційні схеми працюють по черзі. Завдяки цьому ніякі гоночні процеси в комбінаційних схемах не можуть внести похибку в обробку вхідних сигналів. Для цього необхідно лише, щоб інтервал часу $T_Ф$ перевершував максимальну тривалість перехідних процесів. Проектант завжди в змозі забезпечити таке співвідношення на основі паспортних значень максимальних затримок мікросхем.

Величина $T_Ф$ залежить від величини затримки $t_З$ комбінаційних схем, яка може змінюватись у широких межах. Якщо $t_З$ менша вибраної величини $T_Ф$, то таке співвідношення не має негативних наслідків, виключаючи лише зниження швидкості обробки інформації. Але тоді іноді затримка деяких комбінаційних схем перевищує величину робочого інтервалу $T_Ф$, в результаті чого, відповідно

до рис. 7.2, схема стає непрацездатною. У подібних ситуаціях можуть використовуватись різні шляхи вирішення проблеми.

Найпростішим з них є збільшення тривалості $T_Ф$ і, відповідно, періоду синхроімпульсів. Як результат, це може суттєво знизити швидкодію розробленої схеми. Для того, щоб залишити частоту синхронізації незмінною, використовують несиметричну двофазну синхронізацію, при якій $T_{Ф1} \neq T_{Ф2}$. У цьому випадку, якщо можливо, комбінаційні схеми з більшим часом затримки розміщуються в більшому робочому інтервалі. Якщо подвійна організація схемотехніки неможлива, то комбінаційну схему з великою тривалістю $t_З$ розбивають на дві схеми і між ними встановлюють проміжний запам'ятовуючий вузол. Такий спосіб приводить до необхідної послідуючої перефазовки схеми. Широко використовується спосіб, при якому комбінаційні вузли з низькою швидкодією виділяють окремо і для них знижують частоту синхронізації до необхідної.

Найбільш гнучкий спосіб забезпечення високої швидкодії при наявності комбінаційних схем з великою затримкою – це використання багатофазних схем синхронізації, які використовуються у швидкодіючих пристроях. Переваги таких схем ілюструє рис. 7.3.

У залежності від величини конкретної затримки кожної комбінаційної схеми, на *C*-входи пристроїв пам'яті можливо заводити різні фази синхронізації і, відповідно, відкривати тригери-приймачі з затримкою на інтервали часу, кратні $T_Ф$ ($T_Ф$, $2T_Ф$, $3T_Ф$, …) відносно тієї фази, яка синхронізує передавач інформації. Недопустимо тільки синхронізувати тригери-приймачі синхросигналом тієї фази, якою синхронізувалися тригери-передавачі даної комбінаційної схеми. Розглянутий спосіб широко використовується на практиці, адже він дає також можливість зменшити неробочі інтервали комбінаційних схем, що мають місце при очікуванні синхросигналу.

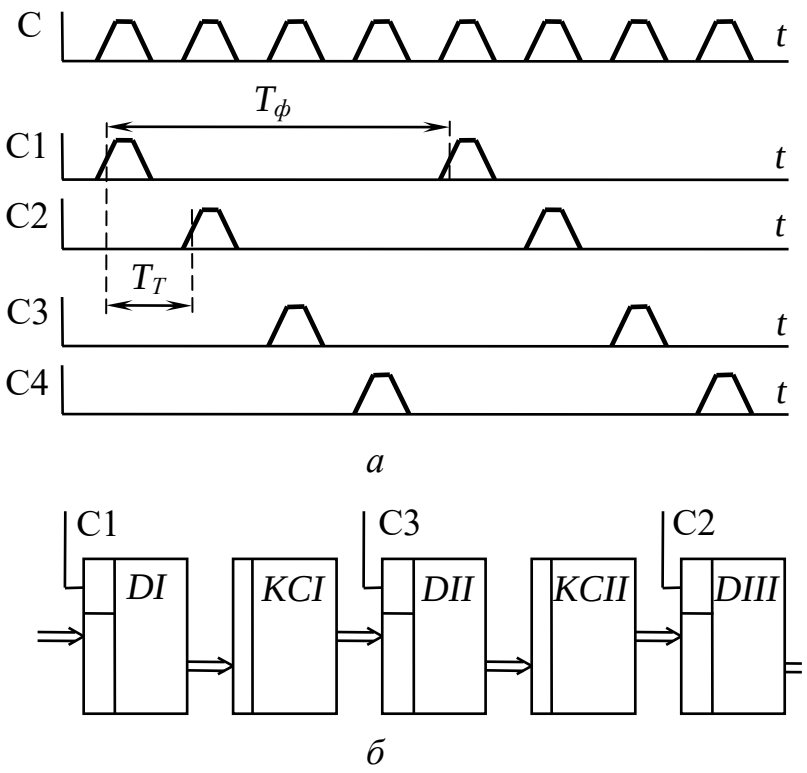


Рис. 7.3. Багатофазна схема синхронізації

На вибір тактової частоти генератора синхросигналів впливають також типи тригерів, розгалуженість схеми розподілення синхросигналів. Вказані особливості використовуються тільки досвідченими конструкторами при проектуванні складних цифрових схем автоматики. З розвитком потужних мікропроцесорів та машинних методів проектування ці особливості стають неактуальними.

У процесі проектування часто виникає необхідність у створенні зворотних зв’язків у синхронних схемах. У таких випадках необхідно дотримуватись наступних правил:

- в усіх схемах з двофазною синхронізацією петля зворотного зв’язку як з логічними елементами, так і без них повинна починатись з виходів тригерів, що синхронізуються однією фазою і закінчуватись на вході тригерів, що синхронізуються іншою фазою;
- недопустимі зв’язки, які передають сигнали з виходу однієї групи тригерів на вхід другої, що синхронізується однією і тією ж фазою.

З вказаних правил витікає, що відсутність у схемах з двофазною синхронізацією замкнутих кіл забезпечується тим, що у будь-який момент часу хоча б один з запам'ятовуючих пристроїв є відключеним і не передає інформацію з входу на вихід. Ці умови можуть бути забезпечені і при використанні однофазної синхронізації, якщо використовувати тригери, які не є "прозорими" для інформаційного сигналу, – наприклад, динамічні тригери. Особливості використання динамічних тригерів у схемах однофазної синхронізації пояснюються на рис. 7.4.

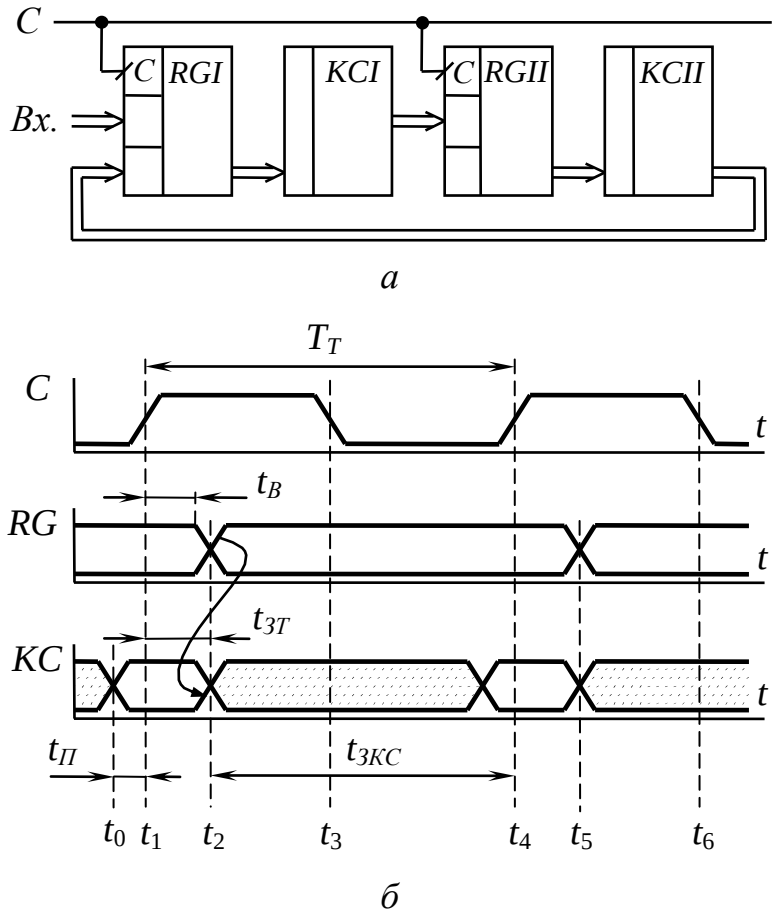


Рис. 7.4. Схема однофазної синхронізації

Для розподілення комбінаційних схем *KCI, KCII* використовуються динамічні тригери, запис інформації в які проходить за фронтом синхроімпульсу. До моменту часу t_0 всі перехідні процеси в комбінаційній схемі *KC* повинні завершитись, залишивши невеликий резерв часу підготовки $t_П$ до появи фронту синхроімпульсу. Поява його в момент t_1, як відомо з роботи

динамічних тригерів, не призводить до миттєвої зміни його станів, а початок цього процесу затримується на інтервал часу витримки t_B. Через інтервал затримки $t_{ЗТ}$ у зміні станів виходів тригерів нові дані з'являються на вході комбінаційної схеми (наприклад, схеми *KCI*), і в ній починається описаний процес гонок, який проходить протягом часу затримки комбінаційної схеми $t_{ЗКС}$ до моменту часу t_4. З аналізу розглянутих динамічних процесів витікає, що фактично тактова частота $f_T = T_T^{-1}$ визначається часом $t_{ЗКС}$.

При однофазній синхронізації допускається використання тригерів, які у різних модулях мають різні моменти зміни станів. Наприклад, тригери пристрою *RGI* спрацьовують за фронтом, а *RGII* – за спадом. Завдяки такій комбінації виникає можливість використовувати *КС* з різними інтервалами затримок.

Особливістю однофазної синхронізації є складність її використання при розгалуженій системі синхронізації. Пояснюється це тим, що на окремих ділянках схеми синхронізації можуть виникати суттєві затримки. Інформаційні сигнали для ланки схеми, що розглядається, можуть не мати затримок. Як результат цього явища, неспівпадіння інформаційних тактів з фронтами синхросигналу – наприклад, *N*-го такту синхросигналу з (*N*+1) тактом інформаційного сигналу. Проконтролювати подібну ситуацію не завжди можливо, тому однофазна синхронізація находить обмежене використання.

Двофазна і багатофазна синхронізація не має вказаного недоліку через те, що вона має можливість попередньо врахувати будь-які затримки як в передачі синхронізуючих, так і інформаційних сигналів.

Важливо звернути увагу і на інші переваги багатофазної синхронізації. Перш за все, модулі пам'яті в багатофазних схемах синхронізації можуть бути побудовані на найпростіших синхронних тригерах, а принципових обмежень на типи тригерів практично немає. Немає обмежень також на часові співвідношення в імпульсних послідовностях синхросигналу або крутизну фронтів, що є обов'язковими для динамічних тригерів. Вказані переваги

багатофазних схем синхронізації, незважаючи на складність побудови розгалуженого дерева синхронізуючих сигналів, приводять до того, що в складних цифрових схемах використовуються переважно вони. Однофазні схеми знаходять використання лише в окремих вузлах або нескладних схемах – регістрах, лічильниках і т. п. Часто однофазна синхронізація використовується в мікроконтролерах, в яких немає необхідності багатоступінчатого розмноження сигналів.

Узгодження вхідних сигналів

Сигнали, що подаються на цифрові схеми з зовнішніх пристроїв, мають характеристики, які можуть суттєво відрізнятись від стандартів цифрових сигналів. Невідповідність між ними може суттєво вплинути на роботу цифрових схем [1].

Найпростіше вирішуються задачі узгодження полярності та амплітуд сигналів, адже в різних серіях цифрових мікросхем випускається достатня кількість перетворювачів рівнів сигналів.

Більш суттєвими задачами є такі, як тривалість фронтів та прив’язування вхідних сигналів до синхросигналів цифрових пристроїв.

Скорочення фронтів. Ця задача пов’язана з тим, що ряд джерел інформаційних сигналів має велику тривалість фронтів, яка значно перевищує тривалість фронтів логічних елементів. Як результат, навіть незначна неідентичність порогових рівнів логічних елементів призводитиме до значного часового розкиду моментів їх спрацьовування. В інтервалі напруг, близьких до порогових рівнів транзистори логічних елементів можуть знаходитись в активному режимі, що приводить до значних витрат потужності, перегріву, появи автогенерації, наскрізних струмів великих величин.

Для попередження подібних ситуацій необхідно перетворити пологі фронти в круті, які співпадали б з тривалістю фронтів використовуваних логічних елементів. Такі задачі вирішуються за допомогою тригерів Шмідта, які мають підвищені пороги спрацьовування. Ці елементи описані раніше.

Тремтіння контактів. Цей недолік будь-яких систем пов'язаний з вібрацією, що має місце при ударі. Це явище спостерігається в будь-яких контактних системах – реле, кнопках, клавіатурі комп'ютерів.

Частота вібраційних коливань контактів при їх замиканні знаходиться в межах від сотень герц до одиниць кілогерц, тому при зміні стану реле або при замиканні кнопкового контакту на вхід цифрової схеми поступає не сигнал, як зображено на рис. 7.32, *а*, а імпульсна послідовність (рис. 4.32, *б*), яка має місце в інтервалі часу $t_1 - t_2$. Для ліквідації вказаного недоліку електромеханічних схем використовуються шунтуючі електронні пристрої.

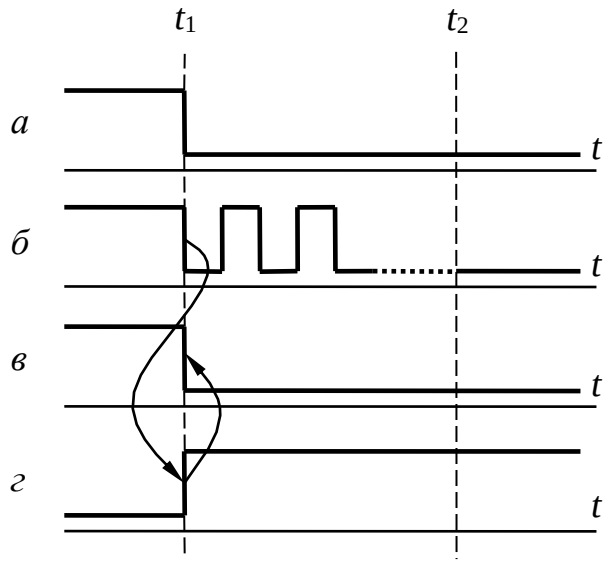


Рис. 7.5. Часові діаграми у контактній системі

На рис. 7.6 приведена схема пристрою для ліквідації вібраційних коливань реле. Його робота полягає в наступному. При знаходженні контакту реле в позиції **1** нульовий потенціал подається на вхід елемента DDI-I, і на виході Q (вихід елемента DDI-2) буде низький потенціал (рис. 7.5, *г*). При зміні положення контакту (з **1** в **2**) на вхід DDI-2 поступить сигнал нульового рівня, і тригер змінить свій стан на одиничний, тобто прямий вихід $Q = 1$ (рис. 7.5, *г*) і інверсний вихід $\overline{Q} = 0$ (рис. 7.5, *в*). Наявність вібраційних коливань, які полягають у тому, що заземлюючий контакт відривається від контакту **2**, але не дотикається до контакту **1**, не зможе привести до зміни стану тригера.

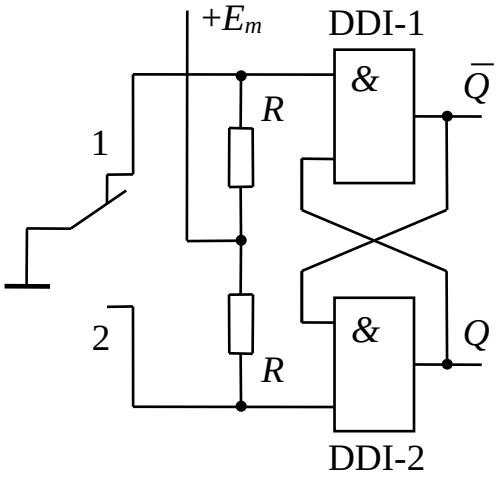


Рис. 7.6. Схема пристрою для ліквідації вібраційних коливань реле

На практиці використовуються різні засоби вирішення проблеми вібраційних коливань. Використовуються, наприклад, схеми одновібраторів з витримкою часу, що перевершує інтервал вібраційних коливань. Знаходить використання спосіб інтегрування вібраційних коливань за допомогою *RC*-інтеграторів з наступним використанням порогових елементів. У контролерах клавіатури усунення проблеми вібраційних коливань досягається за рахунок програмного багаторазового опиту стану натиснутої клавіші в заданому інтервалі часу.

Синхронізатори

Прив'язка зовнішніх сигналів до синхроімпульсів необхідна тому, що синхронні цифрові схеми приймають вхідні сигнали без похибок лише в визначені інтервали часу. Якщо вхідний сигнал подається на комбінаційну схему безпосередньо перед синхроімпульсом, то перехідні процеси в ній можуть не завершитись до появи синхроімпульсу, і в тригери буде записана хибна інформація [1]. Інша причина пов'язана з реакцією на одиночні сигнали. Наприклад, сигнал від натискання кнопки може тривати багато періодів синхросигналу. У той же час, для цифрових синхронних схем його тривалість не повинна перевищувати один період синхрочастоти. Вказані задачі

вирішуються за допомогою тригерних схем, які називаються *синхронізаторами*.

На рис. 7.7 приведена схема синхронізатора на базі статичних *D*-тригерів. Вхідний сигнал з однаковою ймовірністю може з'явитись у будь-який момент часу на інтервалі періоду синхросигналу. Якщо його фронт на часовій діаграмі відображається лініями *а*, *в*, *с*, то тригер DD1 записує на свій вихід Q_1 значення вхідного сигналу, яке за синхросигналом C_2 переписується на вихід Q_2 тригера DD2 і до появи C_1 передається на вихід синхронізатора через логічний елемент DD4 **3І**. На вході матимемо інформаційний сигнал *A*, співпадаючий з синхросигналом C_2. Якщо вхідний сигнал з'являється пізніше синхроімпульсу C_1 (по лінії d_1, *e*), то тригер DD1 прийме його за третім синхроімпульсом (лінії d_1, e_1), який також через DD2 і DD4 за синхросигналом C_2 пройде на вхід у вигляді одиночного імпульсу. Повторення імпульсу неможливе, адже при $Q_2 = 1$ та $C_1 = 1$ тригер DD3 змінить свій стан і $Q_3 = 0$. Тобто, з'явиться сигнал заборони повтору вхідного сигналу. Зрозуміло, що, приймаючи асинхронний сигнал за можливу наявність фазового скиду, на один тактовий інтервал не слід звертати увагу.

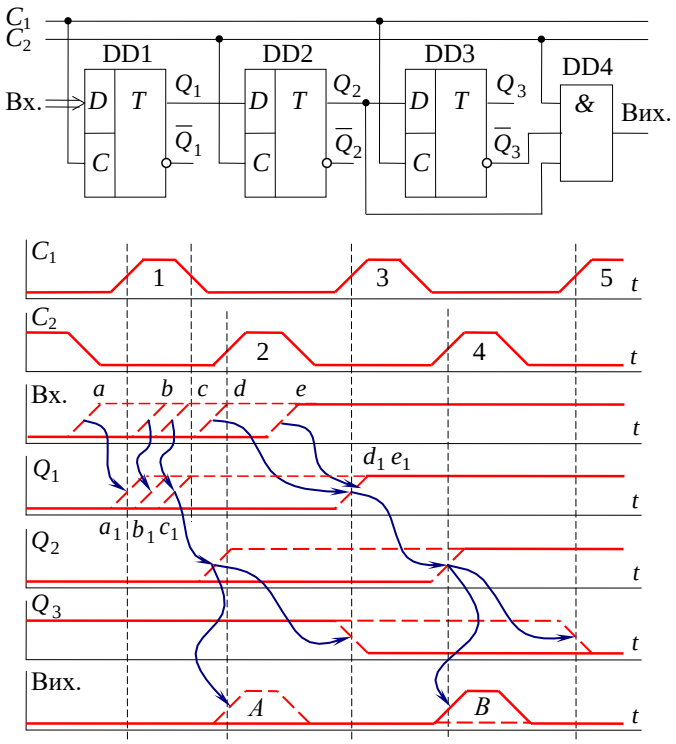


Рис. 7.7. Схема синхронізатора на базі статичних *D*-тригерів

У розглянутій схемі тригер DD1 визначає, з яким із двох сусідніх синхроімпульсів слід зв'язати вхідний сигнал, але, коли б він не з'явився, приведена схема завжди відреагує на нього одним імпульсом. До того ж, вхідний імпульс буде стандартизованим по відношенню до схеми-приймача.

Якщо з'являється необхідність перетворити вхідний сигнал у постійний високопотенціальний рівень, то це можливо зробити, виключивши DD3.

Розглянута схема може служити основою для перетворення неоднорідної вхідної послідовності сигналів у періодичну.

Подібна схема може бути реалізована і з використанням динамічних *D*-тригерів та однофазною синхронізацією. Принципової різниці у побудові така схема не має.

У розглянутій схемі в рамках описаних особливостей роботи виникає сумнів щодо необхідності тригера DD2. Ці сумніви справедливі до того моменту, поки не з'явиться ситуація, при який вхідний сигнал співпадає зі спадом імпульсу **1**. Тоді, незважаючи на наявність позитивного зворотного зв'язку в тригерах, тригер DD1 може знаходитись у невизначеному стані відносно тривалий інтервал часу, затягуючи фронт на імпульсі по виходу Q1. Тригер DD2 у такому випадку виконує функцію швидкодіючого порогового елемента.

Асинхронний обмін інформацією має місце між цифровими пристроями, кожен з яких має свою власну схему синхронізації. В такому випадку сигнали, що поступають з іншого пристрою, сприймаються приймачем як асинхронні. При інтенсивному обміні інформацією постає питання максимально можливої частоти передачі при асинхронному зв'язку.

Розглянемо, як приклад, однофазну синхронізацію, схема якої приводиться на рис. 7.8, *а*. Вхідний сигнал з'являється в часовому інтервалі закінчення синхроімпульсу. В зв'язку з тим, що динамічні тригери DDI і DD2 спрацьовують за спадом імпульсу, на вході DD3 синхроімпульсу з'явиться в один з інтервалів: або *A*, або *B*. Як витікає з часових діаграм (рис. 7.8, *б*), для

появи на вході послідовності синхронних імпульсів необхідно, щоб виконувались умови:

$$T_{imin} > T_T \text{ і } T_{\Pi min} > T_T,$$

де T_{imin} і $T_{\Pi min}$ – мінімальні тривалості імпульсу та паузи вхідної послідовності.

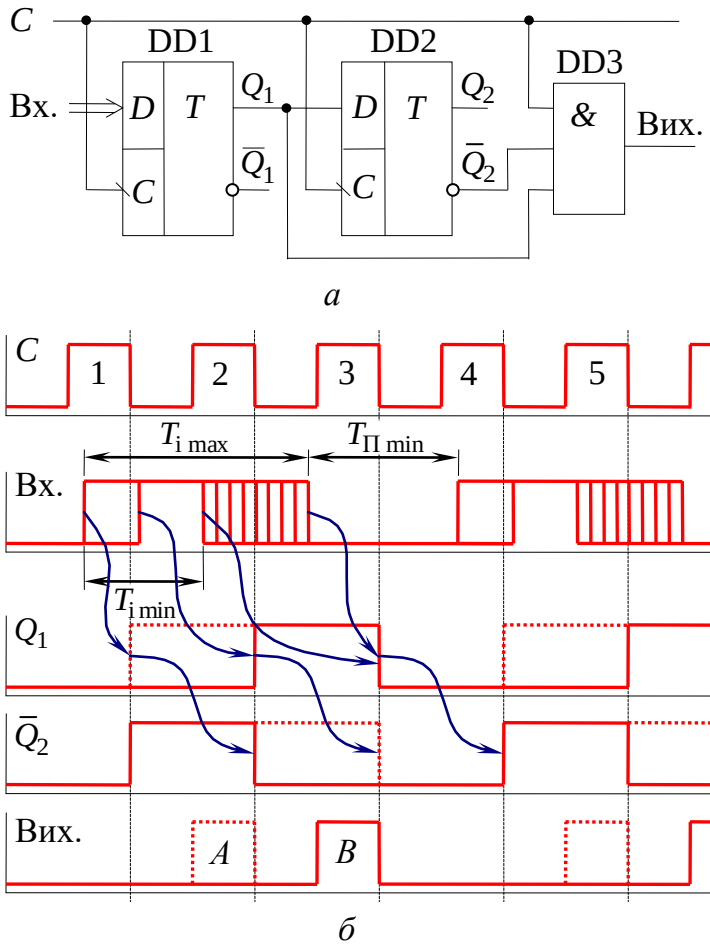


Рис. 7.8. Схема однофазної синхронізації

Враховуючи властивість *D*-тригерів відновлювати свій стан при відсутності вхідного сигналу за спадом синхроімпульсу, з аналізу діаграм витікає, що при синхроімпульсному обміні частота передачі інформації зменшується практично в 2-3 рази, порівняно з синхронним. Це пояснюється тим, що при відносній швидкості обміну біт за такт, частота вихідних тактових сигналів падає в 2,5-3 рази. До того ж, у вихідній послідовності буде порушена періодичність імпульсів, оскільки вони можуть з'являтися в інтервалі *A* або *B*, а це ускладнює задачу обробки інформації.

Практичні проблеми проектування синхронних цифрових схем

Вище були описані засоби синхронізації як надійний метод побудови цифрових систем. У той же час, з підвищенням робочих частот та, відповідно, зменшенням допустимих інтервалів часу на передачу сигналу від одного тригера до іншого виникає ряд реальних проблем, які можна помітити лише в процесі налагодження. Одна з них – зміщення синхросигналу. Вона обумовлена тим, що синхронні динамічні тригери коректно запам'ятовують і передають інформацію лише тоді, коли всі вони одночасно сприймають вхідний сигнал за фронтом або за зрізом синхросигналу [1].

На рис. 7.9 приводиться приклад, коли внаслідок затримки синхросигналу по провіднику друкованої плати між тригерами DD1 та DD2 на виході елемента DD2 отримується хибний сигнал.

Дійсно, за першим синхросигналом *C* низький рівень вхідного сигналу записується в тригер DD1, а в DD2 у цей час також записується низький рівень сигналу з Q_1. За другим синхросигналом у DD1 записується високий рівень вхідного сигналу, а в DD2 повинен записатися низький рівень з Q_1. Але внаслідок того, що по лінії передачі синхросигналу мала місце його затримка на величину $\tau_{зс}$, що перевищує реальну затримку завантаження тригера, в DD2 записується сигнал високого рівня. Величина $\tau_{зс}$ називається *часом зміщення синхросигналу*. Фактично наявність $\tau_{зс}$ еквівалентна появі завади в синхронних схемах. З рис. 7.9 бачимо, що для реально проектованих схем необхідно, щоб виконувалась умова $\tau_{зс} < t_з$.

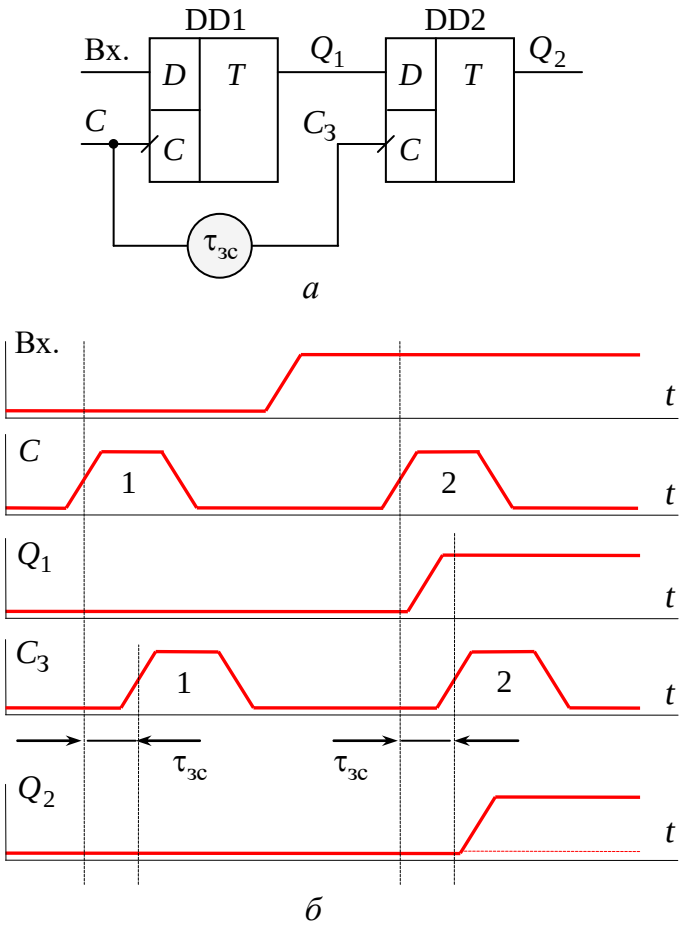


Рис. 7.9. Приклад, коли внаслідок затримки синхросигналу отримується хибний сигнал

Звичайно, розглядаючи приведений приклад ізольовано від цифрової схеми, може виникнути питання: “Чому схема спроектована так, що інформаційний сигнал проходить короткий шлях, а синхросигнал – довгий, в той час як вони повинні проходити майже разом?”. Особливість синхросигналів полягає у тому, що, на відміну від інформаційних, вони розповсюджуються від одного джерела по всій схемі електронної системи. Як результат, характер навантаження джерела синхросигналу суттєво відрізняється від характеру навантаження джерела інформаційних сигналів. Для того, щоб зменшити навантаження джерела синхросигналу, створюються копії синхросигналу за схемою, що приведена на рис. 7.10.

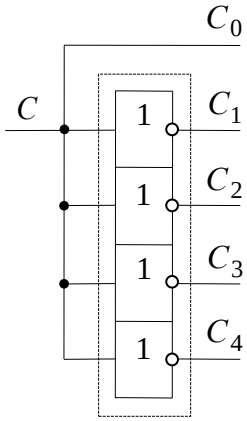


Рис. 7.10. Схема створення копії синхросигналу

Всі копії синхросигналу $C_1 \div C_4$ повинні створюватись ідентичними буферами, які мають однакові часові затримки і, здебільшого, розміщені в одній мікросхемі. Недопустимо використовувати поряд з копіями синхросигналів $C_1 \div C_4$ базовий C_0, оскільки між ними вже існуватиме різниця в часі. Але копії $C_1 \div C_4$ необхідно використовувати ідентично, оскільки різне їх навантаження може привести до різних за величиною часових затримок буферних мікросхем.

При проектуванні друкованих плат слід звернути увагу на характер розводки провідників синхросигналу відповідними комп'ютерними програмами. Здебільшого програми орієнтовані на мінімізацію перехідних отворів, внаслідок чого провідники синхросигналу розміщуються по складній конфігурації між корпусами мікросхем. Оскільки для програм розводки друкованих плат всі провідники однакові, то провідники синхросигналу можуть розміщуватись паралельно заземленню, створюючи цим значні ємності і, відповідно, величини затримок. Як наслідок, різні мікросхеми матимуть різні величини затримок, що в складних і швидкодіючих цифрових системах може призвести до появи помилок. Тому в таких ситуаціях розводку провідників синхросигналу не слід довіряти програмам, а виконувати вручну або закладати, якщо це можливо, в програми розробки друкованих плат.

Окрім синхросигналів, у цифрових системах існує ще ряд сигналів, на які слід звертати особливу увагу при проектуванні. Це можуть бути сигнали дозволу, які необхідно подавати до подачі синхросигналу, а також інші керуючі сигнали. Слід також звертати особливу увагу на введення у цифрову систему асинхронних сигналів. Асинхронними є сигнали вводу інформації (наприклад, з клавіатури), сигнали переривань, а також ряд внутрішніх сигналів, що з'являються в результаті виконання обчислень (ознаки). Зрозуміло, що для вводу цих сигналів у синхронну цифрову систему використовуються синхронізатори, які забезпечують вибір асинхронного сигналу в тактовий момент часу. Проблема синхронізації обумовлена розглянутими вище затримками синхросигналу. Якщо, наприклад, асинхронний сигнал одночасно подається на декілька тригерів, то можлива ситуація, коли в один з тригерів інформація буде записана, а в інший, внаслідок затримки синхросигналу, – ні. В результаті в роботі системи виникне помилка.

Перелік посилань

1. Рябенький В.М., Жуйков В.Я., Гулий В.Д. Цифрова схемотехніка: Навч. посібник. – Львів: Новий світ – 2000, 2009, – 736 с.

Лекція №8

Загальна характеристика скінченних автоматів

Існує велика кількість різноманітних схем, у яких використовується декілька тригерів. У такому випадку кількість внутрішніх станів схеми зростає в пропорції 2^m, де m – кількість тригерів. При цьому, зрозуміло, зростає складність “реакції” такого пристрою на вхідні сигнали, оскільки кожного разу внутрішні стани тригерів можуть бути іншими.

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

У загальному плані *цифровим автоматом* називається пристрій, який формує ряд вихідних дискретних сигналів у відповідності із вхідною бінарною комбінацією сигналів [1, 2]. Найпростішим з таких автоматів є перетворювач кодів, і він називається *комбінаційним*.

У подальшому розглядатимуться автомати з пам’яттю, завдяки якій значення їх виходів залежать не лише від вхідних сигналів, що надходять у визначений момент часу, а також від внутрішнього стану автомата. Оскільки кількість внутрішніх станів залежить від кількості використовуваних тригерів, то цифрові автомати поділяються на *скінченні*, що мають обмежену кількість станів, і автомати *з нескінченною кількістю станів*, або *послідовнісні машини*. До останніх відносяться ЕОМ.

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

автоматів є кодовий замок, реакція на черговий сигнал якого залежить від попередніх сигналів.

Скінченні автомати можуть бути *синхронними*, зміна станів яких відбувається в тактові моменти часу, що задаються зовнішнім генератором. В *асинхронних* автоматах зміна станів відбувається внаслідок дії вхідних сигналів без затримок. Тому асинхронні автомати вважаються більш швидкодіючими і знаходять використання у швидкодіючих інформаційних пристроях вимірювання і керування різноманітними процесами, де необхідна миттєва реакція на зміну вхідних сигналів.

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

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

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

Абстрактний автомат задається *множиною внутрішніх станів* $Q = \{ Q_1, Q_2, \ldots, Q_m \}$ (*алфавітом станів*), *множиною вхідних сигналів* $X = \{ X_1, X_2, \ldots, X_p \}$ (*вхідним алфавітом*), *множиною вихідних сигналів* $Y = \{ Y_1, Y_2, \ldots, Y_k \}$ (*вихідним алфавітом*) і *початковим станом* Q_0. Перехід з одного стану в інший визначається функцією переходів f_p, що визначає стан автомата Q_s, в який він переходить з попереднього стану Q_m при дії сигналу X_p:

$$Q_s = f_p (Q_m, X_p).$$

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

$$Y_k = \lambda (Q_m, X_p).$$

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

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

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

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

Аналогічно, у відповідності з функцією виходів λ, на виході отримається сигнал Y_k, де $Y_k \in Y$.

Приклад 8.1. Розглянути *RS*-тригер як скінчений автомат.

Розв'язання. *RS*-тригер має два стани із множини станів Q: $Q_1 = 0$ і $Q_2 = 1$, де $Q_1, Q_2 \in Q$. Автомат має чотири значення входів X_p з алфавіту входів X_1, X_2, X_3, X_4, де $X_1 = \overline{S} \cdot \overline{R} = 00$; $X_2 = S \cdot \overline{R} = 10$; $X_3 = \overline{S} \cdot R = 01$; $X_4 = S \cdot R = 11$. Вихідної комбінаційної логіки автомат не має, тобто залежність $Y_k = \lambda(Q_m, X_p)$ визначається формулою: $Y = Q$.

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

Поняття *станів* в описі автоматів пов'язане з необхідністю враховувати типи і характер попередніх сигналів, тобто сигналів, які діяли на один або декілька тактів раніше. Стани автомата і є тими відповідними елементами пам'яті, що характеризують попередні сигнали. Введення поняття станів дозволяє усунути час як явну змінну і визначати вихідний сигнал як функцію внутрішнього стану і входу в тактовий момент часу. Такий спосіб опису автоматів має великі переваги перед іншими для асинхронних пристроїв, в яких інтервали часу між сусідніми тактами можуть мати довільні значення, що значно відрізняються між собою. З такої точки зору комбінаційні пристрої відносяться до автоматів, в яких вихід не залежить від попередніх сигналів і повністю визначається комбінацією вхідних сигналів.

У практиці роботи автоматів часто мають місце випадки, коли деякі комбінації значень вхідних символів подавати неприпустимо (згадаємо таблицю станів *RS*-тригера). Такі комбінації є *забороненими* для автомата. Автомати, що мають заборонені вхідні слова, називаються *частковими*. Наприклад, деякий абстрактний автомат має вхідний алфавіт X, що складається з двох символів X_0 і X_1, кожен з яких може приймати значення **0** або **1**. Слова $\overline{X_1}\,\overline{X_0}$, $\overline{X_1}\,X_0$, $X_1\,\overline{X_0}$ є *дозволеними*, а слово $X_1\,X_0$ – *забороненим*.

Наявність заборонених станів зменшує кількість слів вхідного алфавіту, зменшує кількість станів і дозволяє будувати часткові автомати з меншими затратами, ніж повні, для яких вказаних обмежень не існує.

Функція $Y_k = \lambda(Q_m, X_p)$ задає значення виходів автомата з алфавіту Y. Кількість символів алфавіту Y не співпадає з кількістю символів алфавіту X. За аналогією з перетворювачами кодів, автомат забезпечує відображення слів вхідного алфавіту в слова вихідного алфавіту. Якщо прийняти можливу кількість слів вхідного алфавіту за P_x, вихідного алфавіту за K_y, то в залежності від співвідношення між P_x та K_y автомати можуть за функціональним призначенням розділятися на три групи.

При $K_y < P_x$ маємо автомати, які призначені для розв'язання задач керування, стиснення інформації, розпізнавання повідомлень. Наприклад, цифровий кодовий замок може мати 3…5 вхідних символів і лише один вихідний. Інший приклад – розпізнавання введеного пароля при вході в комп'ютер.

Якщо $K_y > P_x$, то маємо ситуацію, коли довжина вихідних слів буде більшою, ніж вхідних, оскільки кількість слів повинна бути однаковою. Тобто вихідні слова будуть нести збиткову інформацію, що використовується, як відмічалося в попередніх розділах, для синтезу перешкодостійких кодів.

При $K_y = P_x$ кількість і довжина вхідних і вихідних слів однакові. Такі автомати використовуються для деяких кодових перетворень (наприклад, перетворення двійкового коду в код Грея, і т. д.), для реалізації деяких методів захисту даних від несанкціонованого доступу (метод підстановок) і т. п.

Автомати Мілі і Мура. Закони функціонування автоматів

У практиці використання цифрових автоматів можна виділити невелику кількість типових алгоритмів їх функціонування. Найбільшого розповсюдження набули два типи автоматів – *автомати Мілі* і *Мура* [1, 2].

Закон функціонування автомата Мілі задається рівняннями:

$$\begin{cases} Q(t+1) = f_p\left[Q(t),\, X(t)\right]; \\ Y_k(t) = \lambda\left[Q(t),\, X(t)\right]; \qquad t = 0,\ 1,\ 2,\ \ldots \end{cases} \tag{8.1}$$

Закон функціонування автомата Мура описується рівняннями:

$$\begin{cases} Q(t+1) = f_p\left[Q(t),\ X(t)\right]; \\ Y_k(t) = \lambda\left[Q(t)\right]; \qquad t = 0,\ 1,\ 2,\ \ldots \end{cases} \tag{8.2}$$

Різниця між автоматами Мілі і Мура полягає лише в тому, що вихідний сигнал останнього залежить тільки від внутрішнього стану, у той час як в автоматі Мілі значення вихідного сигналу залежить також і від вхідного сигналу.

Структура автомата Мілі зображена на рис. 8.1.

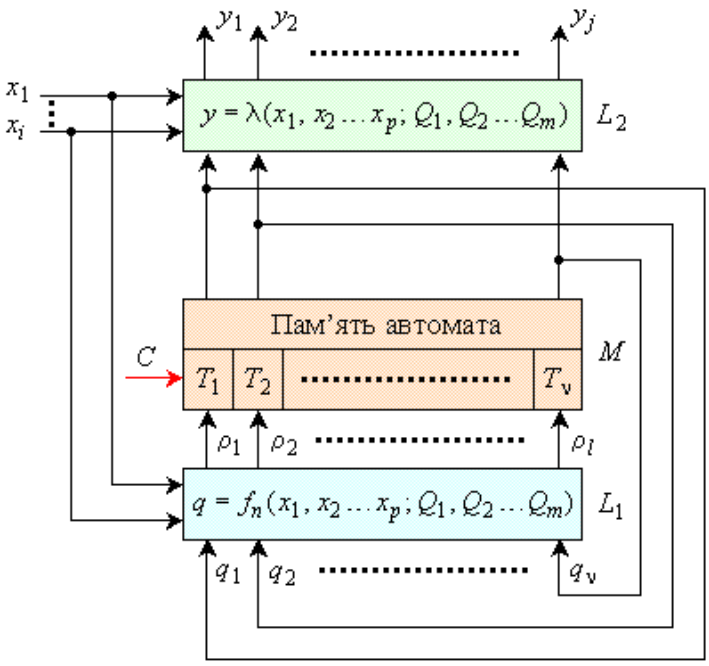


Рис. 8.1. Структура автомата Мілі

Автомат складається з набору елементарних автоматів (тригерів T_1, T_2, ..., T_n), стани яких q_1, q_2, …, q_n в кожний момент часу визначають внутрішній стан автомата в цілому. Під дією вхідних сигналів $x_1 \dots x_i$, які подаються в дискретні моменти часу, відбувається формування сигналів $\rho_1 \dots \rho_l$, що забезпечують послідовне перемикання тригерів $T_1 \dots T_n$. Таким шляхом відбувається послідовна зміна станів автомата. Оскільки комбінаційна схема L_1 формує керуючі сигнали тригерів в залежності від значень виходів тригерів у момент подачі тактового сигналу (для синхронних схем) або в моменти подачі вхідних сигналів (для асинхронних), то значення виходів тригерів у послідуючий момент часу і, відповідно, стан автомата однозначно визначається вхідними сигналами і станом автомата в попередній момент часу.

Сигнали керування тригерами $\rho_1 \dots \rho_l$ формуються комбінаційним пристроєм L_1, структура якого визначає функцію переходів автомата. Функція виходів реалізується комбінаційним пристроєм L_2, який формує сигнали $y_1 \dots y_j$

як функції виходів пам’яті (для автоматів Мура) або як функції виходів пам’яті і вхідних сигналів (для автоматів Мілі).

Переходячи до відносного часу роботи, формули (8.1), (8.2) можна зобразити у вигляді:

- для автомата Мілі:

$$\begin{cases} Y_{n+1} = L_2(Q_{n+1},\ X_n)\ ; \\ Q_{n+1} = L_1(Q_n, X_n)\ ; \end{cases} \qquad (8.3)$$

- для автомата Мура:

$$\begin{cases} Y_{n+1} = L_2(Q_{n+1})\ ; \\ Q_{n+1} = L_1(Q_n, X_n)\ . \end{cases} \qquad (5.4)$$

Рівняння (8.3), (8.4) можуть бути заданими аналітично або у вигляді таблиць станів (таблиць відповідностей).

Взаємозв’язок між поточними Q_n і послідуючими значеннями Q_{n+1} виходів визначається характеристичними рівняннями використовуваних тригерів.

У табл. 8.1 приведені характеристичні рівняння тригерів.

Таблиця 8.1. Характеристичні рівняння тригерів

Тип використовуваного тригера	Характеристичне рівняння
Асинхронний *RS*-тригер	$Q_{n+1} = S_n + \overline{R_n}\ Q_n$
Синхронний статичний *D*-тригер	$Q_{n+1} = C_n\ D_n$
Динамічний *D*-тригер	$Q_{n+1} = D_n$
Динамічний *JK*-тригер (*JK*- *MS*-тригер)	$Q_{n+1} = J_n\ \overline{Q_n} + \overline{K_n}\ Q_n$
T-тригер	$Q_{n+1} = C_n\ \overline{Q_n}$
T-тригер з дозволяючим входом	$Q_{n+1} = C_n\ (V\overline{Q_n} + \overline{V}\,Q_n)$

Поєднуючи характеристичні рівняння тригерів і рівняння комбінаційних схем, можна проводити аналіз роботи існуючого автомата або виконувати його синтез.

Способи опису роботи автоматів

В практиці аналізу і синтезу цифрових автоматів використовують різні способи опису їх роботи [3, 4]. Найбільш поширеними є *табличний* і *графічний способи*.

Розглянемо спочатку опис роботи автомата з використанням таблиць переходів та виходів. Стовбці (рядки) цих таблиць позначають символами з множини *Q*, а рядки (стовбці) – символами з множини *X*.

Кількість рядків таблиці переходів визначається кількістю комбінацій вхідних сигналів ρ, а кількість стовбців – відповідно, кількість станів M автомата.

У табл. 8.2 зображена таблиця автомата з $\rho = 3$, $M = 2^m = 4$.

В кожній клітці таблиці переходів записується стан, в який переходить автомат з попереднього стану (стану, що стоїть у заголовку стовпця) при дії відповідного вхідного сигналу. Так, наприклад, якщо автомат знаходиться у стані Q_0, то при дії сигналу X_1 він перейде в стан Q_3; при дії сигналу X_2 залишиться в стані Q_0, а при дії сигналу X_3 перейде в стан Q_1.

Таблиця 8.2. Таблиця автомата з $\rho = 3$, $M = 2^m = 4$

X \ Q	Q_0	Q_1	Q_2	Q_3
X_1	Q_3	Q_2	Q_1	Q_0
X_2	Q_0	Q_3	Q_2	Q_1
X_3	Q_1	Q_0	Q_3	Q_2

Таблиця виходів (табл. 8.3) відрізняється від таблиці переходів лише тим, що у кожній клітці записується відповідне значення вихідного сигналу автомата.

Таблиця 8.3. Таблиця виходів

X \ Q	Q_0	Q_1	Q_2	Q_3
X_1	Y_0	Y_1	Y_2	Y_3
X_2	Y_3	Y_0	Y_1	Y_2
X_3	Y_2	Y_3	Y_0	Y_0

Таблиці переходів і виходів автомата Мілі можуть бути представлені у вигляді однієї суміщеної таблиці, у клітках якої вказані значення як станів, так і виходів.

Функції переходів і виходів автомата Мура задаються однією таблицею переходів, яка будується так само, як і таблиця переходів автомата Мілі. Різниця полягає лише в тому, що над заголовками кожного стовпця встановлюється окремим рядком значення виходів автомата (табл. 8.4).

Таблиця 8.4. Таблиця переходів

	Y_2	Y_1	Y_0
X \ Q	Q_0	Q_1	Q_2
X_1	Q_2	Q_1	Q_2
X_2	Q_0	Q_2	Q_1
X_3	Q_1	Q_0	Q_0

Для частково заданих автоматів, у яких функції виходів і функції переходів визначені не для всіх комбінацій Q_m і X_p, відповідні клітки залишаються незаповненими.

Як приклад, розглянемо більш детально автомати, що описуються табл. 8.2 – табл. 8.4.

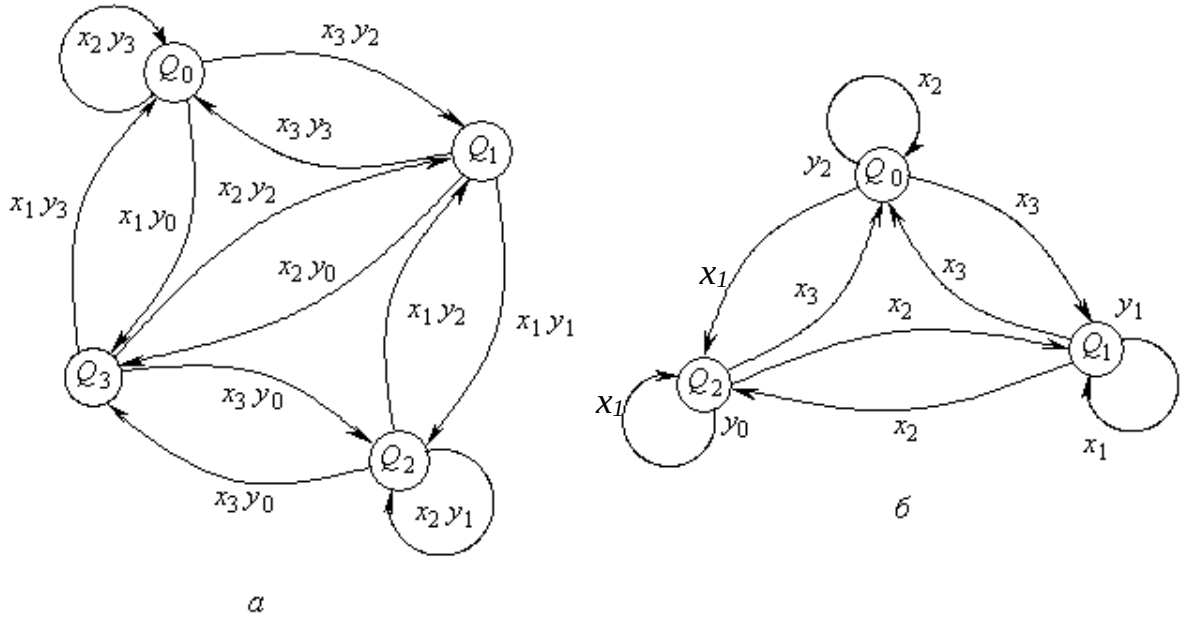
При двійкових вхідних сигналах входи автоматів X_1, X_2, X_3 можуть бути задані вхідними логічними змінними x_1 та x_2, тобто з множини $J = 2$. В алфавіті X матимемо $2^J = 4$ різних слова, а саме: $X_1 = \overline{x_1}\ \overline{x_2}$, $X_2 = x_1 \overline{x_2}$, $X_3 = \overline{x_1} x_2$, $X_4 = x_1 x_2$. Звідси витікає, що $X_4 = x_1 x_2$ – це заборонене слово для автоматів, що розглядаються. Але така заборона може існувати не для всіх станів автомата, а тільки для деяких. Приклад такого автомата заданий таблицею переходів табл. 8.5. З таблиці витікає, що при стані Q_0 на вхід автомата не повинен надходити сигнал X_2, оскільки перехід у такому випадку не визначений.

Таблиця 8.5. Таблиця переходів неповністю заданого автомата

X \ Q	Q_0	Q_1	Q_2
X_1	Q_1	Q_2	Q_0
X_2	—	Q_2	Q_1

Можна прийняти, що $X_1 = x = 1$, а $X_2 = \bar{x} = 0$ і, відповідно, лише значення $X_1 = 1$ може змінювати стан автомата. З табл. 8.5 однозначно визначається і послідовність зміни станів автомата при дії X_1: $Q_0 \Rightarrow Q_1 \Rightarrow Q_2 \Rightarrow Q_0$. Для цього необхідно задати послідовність X_1, X_1, X_1. З цього витікає, що автомат закінчує свою роботу переходом до початкового стану. Якщо функція λ = 1, то виходи автомата Мілі одночасно визначатимуться значеннями його внутрішніх станів. В тому випадку, коли X_1 є тактовим сигналом, що діє лише на тригери, а стани автомата змінюються упорядковано в зростаючому або спадаючому напрямку, автомат називається *лічильним автоматом*, або *лічильником*.

Більш наочним є спосіб опису автоматів з допомогою графів, подібно до того, як описувалася робота тригерів. Різниця полягає в тому, що автомат може мати суттєво більшу кількість станів. На рис. 8.2 показані граф-схеми автоматів Мілі і Мура, які задані табл. 8.2 – 8.3 (рис.8.2.*а*), табл.8.4 (рис.8.2.*б*).



а *б*

Рис. 8.2. Граф-схеми автоматів Мілі і Мура

Граф-схеми широко використовуються як при аналізі, так і при синтезі автоматів, а також при переході від словесного до формалізованого їх опису.

За допомогою таблиць переходів і виходів, як і за допомогою граф-схеми завжди можна знайти вихідну реакцію автомата на будь-яке вхідне слово, що належить множині X.

Робота автомата може описуватися у часі за допомогою спеціальної таблиці (табл. 8.6), яка називається *стрічкою цифрового автомата*.

Таблиця 8.6. Стрічка цифрового автомата

Такт	0	1	2	3	4	5	6	7	8	9	10
Вхідний сигнал	X_2	X_1	X_1	X_2	X_3	X_1	X_1	X_3	X_2	X_1	X_1
Стан	Q_0	Q_2	Q_3	Q_0	Q_2	Q_0	Q_3	Q_1	Q_2	Q_3	Q_3
Вихідний сигнал	Y_0	Y_1	Y_2	Y_0	Y_0	Y_1	Y_2	Y_2	Y_1	Y_1	Y_2

Особливість такої стрічки полягає в тому, що для будь-якої пари сусідніх тактів i та (i+1) можна виділити четвірку символів (виділена у табл. 8.6 жирною лінією), яка показує, в який стан перейде цифровий автомат в (i+1)-ому такті і який вихідний сигнал буде сформований під дією вхідного сигналу.

Однією з форм зображення автомата є його *дерево переходів і виходів*. Дерево може мати декілька ярусів, у кожному з яких за допомогою гілок показуються можливі переходи, починаючи з нульового стану.

На рис. 8.3 приводиться приклад дерева переходів, що відповідає табл. 8.2.

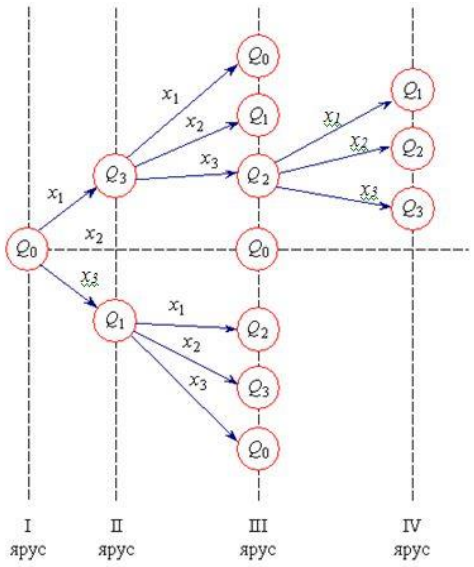


Рис. 8.3. Дерево переходів

Еквівалентні перетворення автоматів

Для будь-якого автомата Мілі можна побудувати відповідний автомат Мура, і навпаки [1, 2].

Розглянемо алгоритми таких переходів на прикладі автомата Мура, що заданий табл. 8.7 і табл. 8.8.

Таблиця 8.7. Таблиця переходів автомата Мура

X \ Q	Q_0	Q_1	Q_2	Q_3
x_1	Q_1	Q_2	Q_0	Q_2
x_2	Q_3	Q_0	Q_0	Q_0
x_3	Q_2	Q_3	Q_2	Q_3

Таблиця 8.8. Таблиця виходів автомата Мура

X \ Q	Q_0	Q_1	Q_2	Q_3
x_1	Y_3	Y_2	Y_1	Y_1
x_2	Y_2	Y_1	Y_3	Y_3
x_3	Y_1	Y_2	Y_2	Y_3

Поставимо у відповідність кожній парі Q_m та X_p автомата Мілі стан Q_{mp} автомата Мура. До множини станів автомата Мура включимо початковий стан автомата Мілі Q_0, але позначимо його Q_{00}. Для прикладу, що розглядається, така відповідність приведена в табл. 8.9.

Таблиця 8.9. Таблиця відповідності автомата Мілі автомату Мура

X \ Q	Q_0 / Q_{00}	Q_1	Q_2	Q_3
x_1	Q_1 / Q_{01}	Q_2 / Q_{11}	Q_0 / Q_{21}	Q_2 / Q_{31}
x_2	Q_3 / Q_{02}	Q_0 / Q_{12}	Q_0 / Q_{22}	Q_0 / Q_{32}
x_3	Q_2 / Q_{03}	Q_3 / Q_{13}	Q_2 / Q_{23}	Q_3 / Q_{33}

Якщо автомат Мілі має m станів і p вхідних сигналів, то відповідний йому автомат Мура матиме $mp+1$ станів.

З табл. 8.9 витікає той факт, що стан Q_0 автомата Мілі співпадає зі станами Q_{00}, Q_{21}, Q_{12}, Q_{22}, Q_{32} автомата Мура. Тобто має місце наступна тотожність:

$$Q_0 = Q_{00} = Q_{21} = Q_{12} = Q_{22} = Q_{32}.$$

Аналогічно,

$Q_1 = Q_{01}$; $Q_2 = Q_{03} = Q_{11} = Q_{23} = Q_{31}$; $Q_3 = Q_{02} = Q_{13} = Q_{33}$.

Така відповідність означає, що одному переходу автомата Мілі з Q_0 в Q_1 повинні бути відповідними всі переходи автомата Мура зі станів Q_{00}, Q_{21}, Q_{12}, Q_{22}, Q_{32} в стан Q_{01}; переходу з Q_1 в Q_2 повинні бути відповідні всі переходи автомата Мура зі стану Q_{01} в стани Q_{03}, Q_{11}, Q_{23}, Q_{31}, і т.д. Зі сказаного витікає наступний висновок: якщо стан Q_{mp} входить до множини станів, що співпадає зі станом, наприклад, Q_1, то стовбець таблиці переходів для стану Q_{mp} співпадатиме з стовбцем таблиці переходів для стану Q_1. Значення m функції виходів для еквівалентного автомата Мура:

$$\lambda\left(Q_{mp}\right) = \lambda\left(Q_m,\ X_p\right) \text{ при } Q_{mp} \neq Q_{00}.$$

Наприклад, для стану Q_{01} функція виходу буде Y_3, оскільки вона відповідає набору Q_0 і X_1 з Табл. 5.8; для стану Q_{02} функція виходу буде Y_2, що відповідає набору Q_0 і X_2, і т. д.

Для початкового стану Q_{00} значення вихідного сигналу вибирається довільно.

Внаслідок описаних відповідностей і перетворень можна побудувати таблицю переходів і виходів еквівалентного автомата Мура (див. табл. 8.10).

Таблиця 8.10. Таблиця переходів і виходів еквівалентного автомата Мура

	Y_3	Y_3	Y_2	Y_1	Y_2	Y_1	Y_2	Y_1	Y_3	Y_2	Y_1	Y_3	Y_3
	Q_{00}	Q_{01}	Q_{02}	Q_{03}	Q_{11}	Q_{12}	Q_{13}	Q_{21}	Q_{22}	Q_{23}	Q_{31}	Q_{32}	Q_{33}
X_1	Q_{01}	Q_{11}	Q_{31}	Q_{21}	Q_{21}	Q_{01}	Q_{31}	Q_{01}	Q_{01}	Q_{21}	Q_{21}	Q_{01}	Q_{31}
X_2	Q_{02}	Q_{12}	Q_{02}	Q_{22}	Q_{22}	Q_{02}	Q_{32}	Q_{02}	Q_{02}	Q_{22}	Q_{22}	Q_{12}	Q_{32}
X_3	Q_{03}	Q_{13}	Q_{33}	Q_{23}	Q_{23}	Q_{03}	Q_{33}	Q_{33}	Q_{33}	Q_{23}	Q_{23}	Q_{33}	Q_{33}

Розглянемо особливість переходу від автомата Мура до еквівалентного автомата Мілі.

Якщо f_{p1} і λ_k функції переходів і виходів автомата Мура, то функції переходів $f_{p2}(Q,\ X)$ і виходів $\lambda_2(Q,\ X)$ еквівалентного автомата Мілі:

$$f_{p2}(Q,X)=f_{p1}(Q,X);\qquad \lambda_2(Q,\ X)=\lambda_1\left(f_{p1}(Q,\ X)\right).$$

Звідси витікає, що таблиця переходів еквівалентного автомата Мілі співпадає з таблицею переходів автомата Мура, а в кожну клітку таблиці виходів записується символ, яким відмічений стан автомата Мура в заданій клітці.

При такому перетворенні граф автомата Мілі відрізняється від графу автомата Мура лише тим, що вихідні сигнали з вузлів графа перенесені на всі гілки, що входять у даний вузол.

Перелік посилань

1. Рябенький В.М., Жуйков В.Я., Гулий В.Д. Цифрова схемотехніка: Навч. посібник. – Львів: Новий світ – 2000, 2009, – 736 с.

2. Жабін В.І., Жуков І.А., Клименко І.А., Ткаченко В.К. Прикладна теорія цифрових автоматів. – К. НАУ, 2007. – 364 с.

Лекція №9

Основи аналізу цифрових автоматів

Аналіз роботи автомата виконується з метою визначення його стану в послідуючий тактовий момент часу і передбачення послідуючих станів [1, 2]. Такі задачі з'являються при вивченні роботи невідомих схем, при налагодженні пристроїв цифрової схемотехніки. Задачі аналізу є досить складними і розв'язуються поетапно.

Рекомендуються наступні кроки:

- визначаються стани на послідуючому тактовому моменті часу і значення виходів вхідної та вихідної комбінаційних функцій L_1 і L_2;
- використовуються функції L_1 і L_2 для побудови таблиці станів, яка повністю визначає послідуючий стан і значення виходу схеми для кожної комбінації поточного стану і виходів;
- будується діаграма станів, яка містить інформацію з попереднього кроку (граф переходів).

Аналіз автоматів з D-тригерами

Розглянемо особливості аналізу на конкретних прикладах скінченних автоматів [1].

Виконаємо аналіз схеми, що приведена на рис. 9.1 і містить вхідну комбінаційну функцію L_1, що реалізована на елементах DD1, DD2, вихідну комбінаційну функцію L_2, реалізовану на DD5, і модуль пам'яті з використанням динамічних *D*-тригерів DD3 і DD4. Тобто схема, що розглядається, відноситься до групи автоматів Мілі.

Розглянемо поведінку схеми в роботі.

За фронтом синхросигналу C_n кожен з тригерів приймає сигнал, що попередньо був поданий на *D*-вхід, і передає його на свій вихід в відповідності до алгоритму (див. табл. 8.1). Сигнали q_1 і q_2 є *встановлюючими* (*збуджуючими*) для *D*-тригерів у кожний тактовий момент часу.

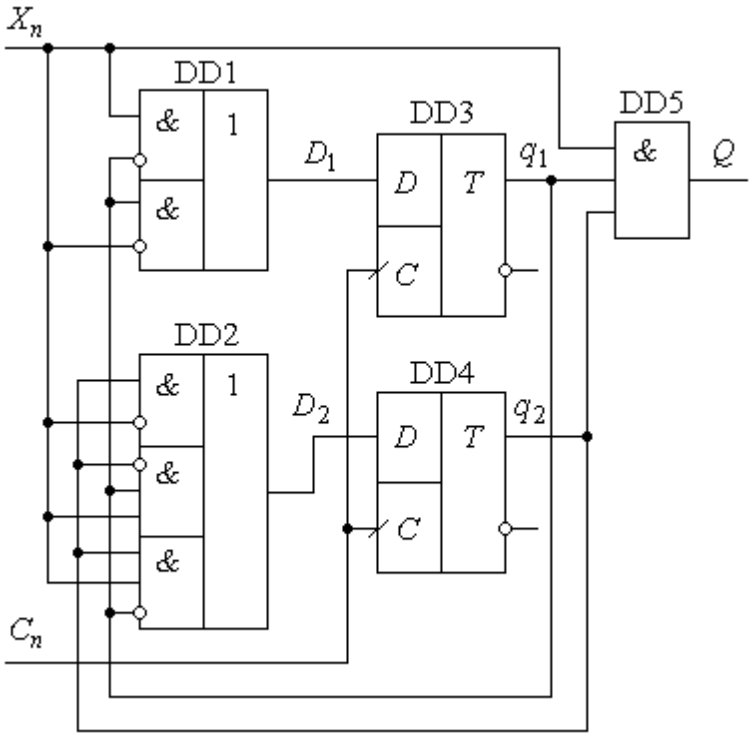


Рис. 9.1. Схема автомата на *D*-тригерах

Логічні рівняння, що визначають встановлюючі сигнали як функції поточного стану і вхідних сигналів, називаються *збуджуючими рівняннями* (*Excitation equations*) і можуть бути отримані з принципової схеми:

$$D_{1n} = X_n \overline{q_{1n}} + \overline{X_n}\, q_{1n} = X_n \oplus q_{1n}; \tag{9.1}$$

$$D_{2n} = \overline{X_n}\, q_{2n} + X_n\, q_{1n}\, \overline{q_{2n}} + X_n\, \overline{q_{1n}}\, q_{2n} = \overline{X_n}\, q_{2n} + X_n \left(q_{1n} \oplus q_{2n}\right). \tag{9.2}$$

Функція L_1, що описується збуджуючими рівняннями, називається *функцією збудження*.

Використовуючи характеристичні рівняння *D*-тригерів, можемо записати:

$$q_{1(n+1)} = D_{1n} \cdot C_n; \tag{9.3}$$

$$q_{2(n+1)} = D_{2n} \cdot C_n. \tag{9.4}$$

Підставляючи значення D_{1n}, D_{2n} з (9.1) – (9.2), отримуємо:

$$q_{1(n+1)} = C_n \cdot \left(X_n \oplus q_{1n}\right); \tag{9.5}$$

$$q_{2(n+1)} = C_n \cdot \left(\overline{X_n}\ \ q_{2n} + X_n \left(q_{1n} \oplus q_{2n}\right)\right). \tag{9.6}$$

Отримані рівняння (9.5) – (9.6), що виражають значення стану автомата в (n +1)-й момент часу в залежності від значень його входів і стану в n-й момент часу, називаються *перехідними рівняннями*. Їх суть полягає в тому, що вони надають можливість передбачити стан автомата в (n +1)-й момент часу на основі інформації про стан і значення входів в n-й момент часу.

Оскільки внутрішній стан автомата визначається значеннями виходів двох тригерів, то можна стверджувати, що існують лише чотири можливі стани автомата: $(q_1, q_2) = 00, 01, 10, 11$. Для кожного стану в приладі, що розглядається, можливі лише два значення вхідного сигналу X: 0, 1.

В результаті маємо лише вісім комбінацій стан-вхід (у загальному плані автомат з m станами і p вхідними сигналами матиме 2^{m+p} комбінацій стан-вхід).

Перехідні рівняння (9.5) – (9.6) можна задати в табличній формі (див. табл. 9.1).

Таблиця 9.1. Таблиця представлення перехідних рівнянь

C_n	X_n	q_{2n}	q_{1n}	$q_{2(n+1)}$	$q_{1(n+1)}$
1	0	0	0	0	0
2	0	0	1	0	1
3	0	1	0	1	0
4	0	1	1	1	1
1	1	0	0	0	1
2	1	0	1	1	0
3	1	1	0	1	1
4	1	1	1	0	0

З табл. 9.1 наглядно витікає особливість роботи автомата. Вона полягає в тому, що при $X_n = 0$ автомат не працює, тобто значення вихідних станів в (n +1)-й момент часу повторюють відповідні значення в n-й момент часу. При $X_n = 1$ він виконує функцію двійкового лічильника з максимальним фіксованим значенням $q_2\, q_1 = 11_2 = 3_{10}$.

Побудуємо тепер на основі табл. 9.1 таблицю переходів автомата. Для цього кожен з можливих станів автомата позначимо відповідним значенням з множини Q, починаючи з нульового:

$$Q_0 = \overline{q_{2n}}\ \ \overline{q_{1n}} = 00;\quad Q_1 = \overline{q_{2n}}\ \ q_{1n} = 01;\quad Q_2 = q_{2n}\ \ \overline{q_{1n}} = 10;\quad Q_3 = q_{2n}\ \ q_{1n} = 11.$$

В результаті отримуємо таблицю переходів у вигляді табл. 9.2 або її скорочену форму – табл. 9.3.

Таблиця 9.2. Таблиця переходів

C_n	X_n	Q_n	Q_{n+1}
1	0	Q_0	Q_0
2	0	Q_1	Q_1
3	0	Q_2	Q_2
4	0	Q_3	Q_3
1	1	Q_0	Q_1
2	1	Q_1	Q_2
3	1	Q_2	Q_3
4	1	Q_3	Q_0

Таблиця 9.3. Скорочена таблиця переходів

$X \cdot C_n$ \ Q	Q_0	Q_1	Q_2	Q_3
0	Q_0	Q_1	Q_2	Q_3
1	Q_1	Q_2	Q_2	Q_0

Граф-схема автомата приведена на рис. 9.2.

Послідовність переходів з одного стану до іншого забезпечується при наявності динамічного сигналу C_n від зовнішнього тактового генератора і потенційного (дозволяючого) сигналу x.

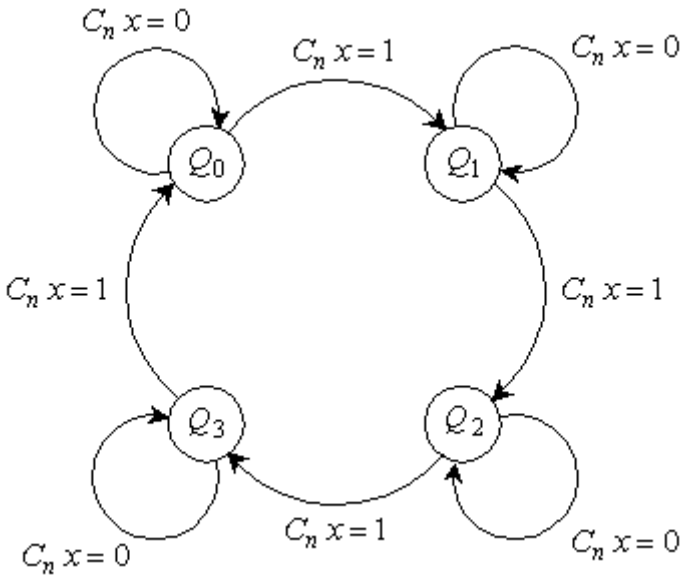


Рис. 9.2. Граф-схема автомата

Робота автомата може бути описана і за допомогою часових діаграм (рис. 9.3), які відображають зміну станів тригерів автомата (q_{1n}, q_{2n}) у відповідні тактові моменти часу, і відповідні стани автомата ($Q_0 \div Q_3$) на кожному тактовому інтервалі.

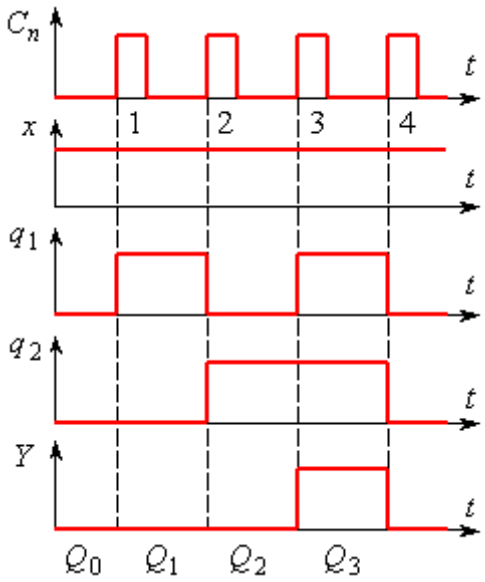


Рис. 9.3. Часові діаграми, що описують роботу схеми

Розглянуту схему автомата можливо спростити за рахунок видалення інверторів, оскільки інверсні значення виходів тригерів можна отримати безпосередньо з тригерів.

Вихідний сигнал автомата визначається внутрішнім його станом і значенням вхідного сигналу, тому функція L_2 описується рівнянням:

$$Y = x \cdot q_{1n} \cdot q_{2n}.$$

Приклад 9.1. Проаналізувати роботу скінченного автомата, принципова схема якого приведена на рис. 9.3. Побудувати його таблицю станів та часові діаграми роботи.

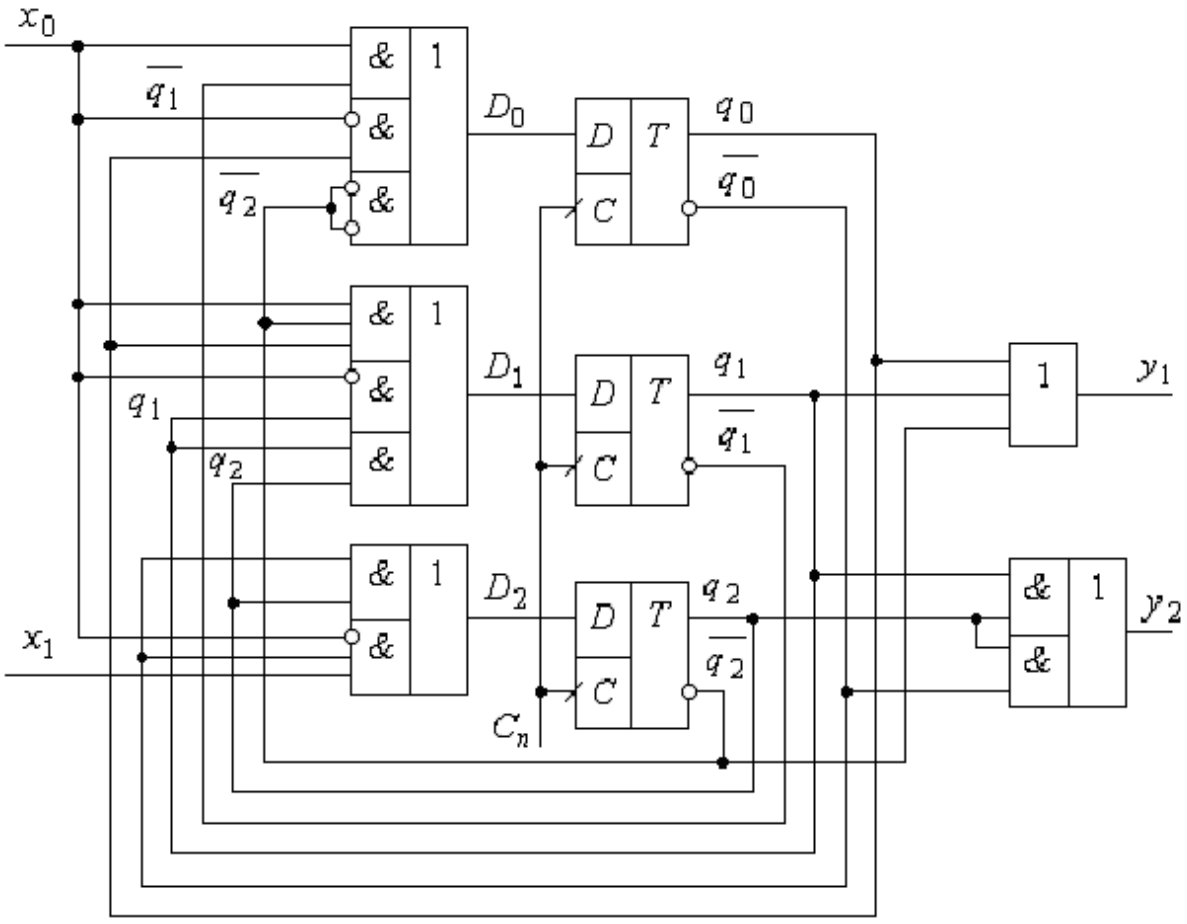


Рис. 9.3. Принципова схема скінченного автомата

Розв’язання. Із зовнішнього вигляду принципової схеми бачимо, що аналізований автомат відноситься до класу машин Мура, а рівняння для вихідних сигналів y_1 та y_2 мають вигляд:

$$y_1 = \overline{q_2} + q_1 + q_0; \quad (9.7)$$

$$y_2 = q_2 \cdot (q_1 + \overline{q_0}). \quad (9.8)$$

Аналізуючи роботу логічної схеми, можемо записати рівняння збудження тригерів:

$$D_{0n} = x_{0n}\overline{q_{1n}} + \overline{x_{0n}}q_{0n} + q_{2n};$$

$$D_{1n} = x_{0n}\overline{q_{2n}}q_{0n} + \overline{x_{0n}}q_{1n} + q_{2n}q_{1n};$$

$$D_{2n} = q_{2n}\overline{q_{0n}} + x_{1n}\overline{x_{0n}}\,\overline{q_{0n}}.$$

Підставляючи рівняння збудження в характеристичне рівняння *D*-тригерів, отримаємо:

$$q_{0(n+1)} = C_n D_{0n} = C_n(x_{0n}\overline{q_{1n}} + \overline{x_{0n}}q_{0n} + q_{2n});$$

$$q_{1(n+1)} = C_n D_{1n} = C_n(x_{0n}\overline{q_{2n}}q_{0n} + \overline{x_{0n}}q_{1n} + q_{2n}q_{1n});$$

$$q_{2(n+1)} = C_n D_{2n} = C_n(q_{2n}\overline{q_{0n}} + x_{1n}\overline{x_{0n}}\,\overline{q_{0n}})$$

Функції $q_{0(n+1)}$, $q_{1(n+1)}$, $q_{2(n+1)}$ зобразимо в табличній формі (див. табл. 9.4).

Таблиця 9.4. Зображення функцій у табличній формі

C_n	x_0	x_1	q_{2n}	q_{1n}	q_{0n}	$q_{2(n+1)}$	$q_{1(n+1)}$	$q_{0(n+1)}$
1	0	0	0	0	0	0	0	0
2	0	0	0	0	1	0	0	1
3	0	0	0	1	0	0	1	0
4	0	0	0	1	1	0	1	1
5	0	0	1	0	0	1	0	1
6	0	0	1	0	1	0	0	1
7	0	0	1	1	0	1	1	1
8	0	0	1	1	1	0	1	1
9	0	1	0	0	0	1	0	0
10	0	1	0	0	1	0	0	1
11	0	1	0	1	0	1	1	0
12	0	1	0	1	1	0	1	1
13	0	1	1	0	0	1	0	1
14	0	1	1	0	1	0	0	1
15	0	1	1	1	0	1	1	1
16	0	1	1	1	1	0	1	1
17	1	0	0	0	0	0	0	1
18	1	0	0	0	1	0	1	1
19	1	0	0	1	0	0	0	0
20	1	0	0	1	1	0	1	0
21	1	0	1	0	0	1	0	1
22	1	0	1	0	1	0	0	1
23	1	0	1	1	0	1	1	1
24	1	0	1	1	1	0	1	1
25	1	1	0	0	0	0	0	1
26	1	1	0	0	1	0	1	1
27	1	1	0	1	0	0	0	0
28	1	1	0	1	1	0	1	0
29	1	1	1	0	0	1	0	1
30	1	1	1	0	1	0	0	1
31	1	1	1	1	0	1	1	1
32	1	1	1	1	1	0	1	1

Введемо умовні позначення станів у відповідності до табл. 9.5.

Таблиця 9.5. Умовне позначення станів

№ п/п	q_{2n}	q_{1n}	q_{0n}	Q_m
0	0	0	0	Q_0
1	0	0	1	Q_1
2	0	1	0	Q_2
3	0	1	1	Q_3
4	1	0	0	Q_4
5	1	0	1	Q_5
6	1	1	0	Q_6
7	1	1	1	Q_7

Кодування станів дає можливість побудувати таблицю переходів автомата на основі табл. 9.4 (див. табл. 9.6).

Таблиця 9.6. Таблиця переходів

Q_n	Q_{n+1}				$y_1\ y_2$
	$C_n\ x_0\ x_1$				
	00	01	10	11	
Q_0	Q_0	Q_4	Q_1	Q_1	10
Q_1	Q_1	Q_1	Q_3	Q_3	10
Q_2	Q_2	Q_6	Q_0	Q_0	10
Q_3	Q_3	Q_3	Q_2	Q_2	00
Q_4	Q_5	Q_5	Q_5	Q_5	11
Q_5	Q_1	Q_1	Q_1	Q_1	10
Q_6	Q_7	Q_7	Q_7	Q_7	11
Q_7	Q_3	Q_3	Q_3	Q_3	11

На основі таблиці переходів будується граф-схема автомата (рис. 9.4) і аналізуються умови переходів.

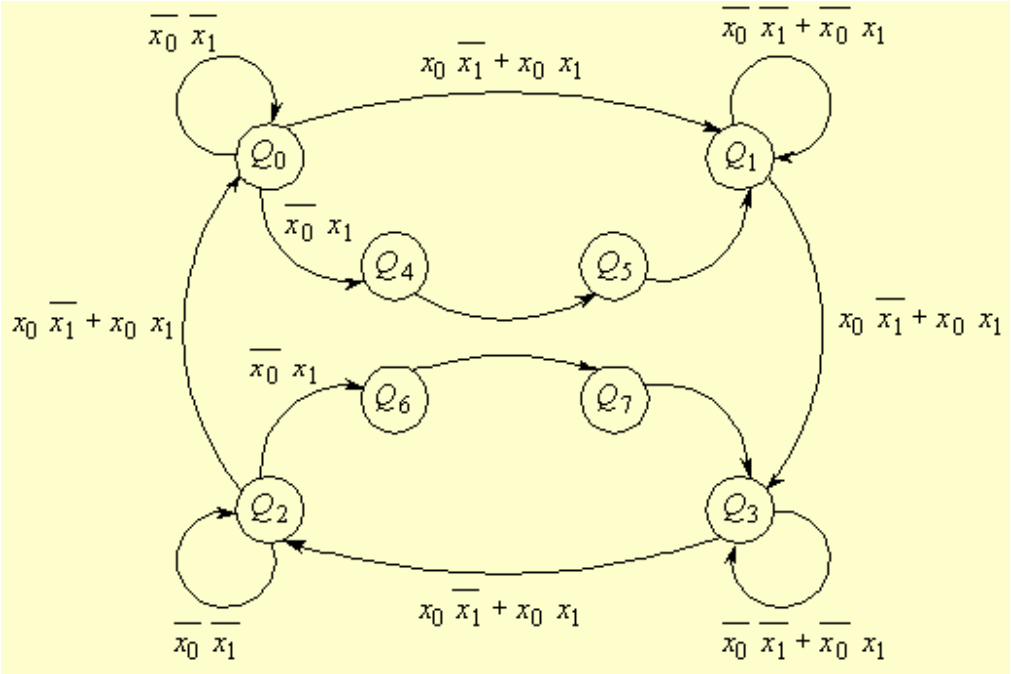


Рис. 9.4. Граф-схема автомата

Аналіз роботи автомата на основі граф-схеми з урахуванням керуючих сигналів дає можливість виявити деякі аспекти його роботи. Наприклад, перехід зі стану Q_4 в стан Q_5 забезпечується при всіх значеннях $x_0\,x_1$ і є безумовним. Аналогічно, маємо переходи з Q_5 в Q_1; з Q_6 в Q_7; з Q_7 в Q_3.

Звідси витікає, що перехід з Q_0 в Q_4 можна замінити на перехід з Q_0 в Q_1, а перехід з Q_2 в Q_6 – на перехід з Q_2 в Q_3. Таблиця переходів при цьому може бути спрощена (див. табл. 9.7), а вигляд граф-схеми відповідатиме рис. 9.5.

Таблиця 9.7. Таблиця переходів

Q_n	Q_{n+1}				$y_1\ y_2$
	$C_n\ x_0\ x_1$				
	00	01	10	11	
Q_0	Q_0	Q_1	Q_1	Q_1	10
Q_1	Q_1	Q_1	Q_3	Q_3	10
Q_2	Q_2	Q_3	Q_0	Q_0	10
Q_3	Q_3	Q_3	Q_2	Q_2	00

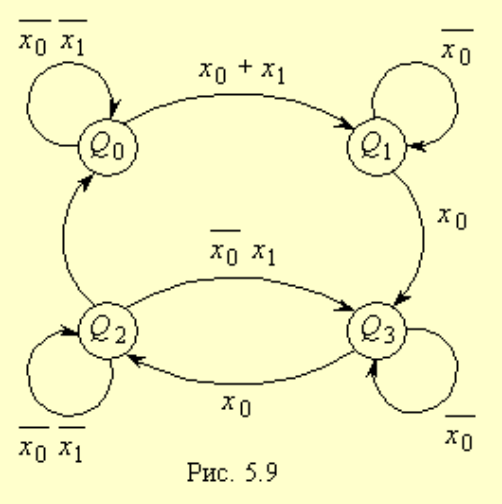


Рис. 9.5. Граф-схема автомата

Приведений аналіз схеми автомата показує, що при $x_1 = 0$ матимемо лічильник, який працює у відповідності до коду Грея. При $x_1 = 1$ виникає можливість забезпечувати своєрідний тригерний режим між станами Q_2 і Q_3.

Отримані результати дають можливість побудувати часові діаграми роботи автомата. При цьому слід враховувати, що при станах $Q_4 \div Q_7$ має місце $y_1 = 1$, а y_2 переходить в нуль у стані Q_5, який є нестійким. Тому можна вважати, що $y_2 = 1$.

Часові діаграми роботи автомата приведені на рис. 9.6. Часові діаграми виходів y_1 та y_2 не приведені, оскільки вони зрозумілі з формул (9.7), (9.8).

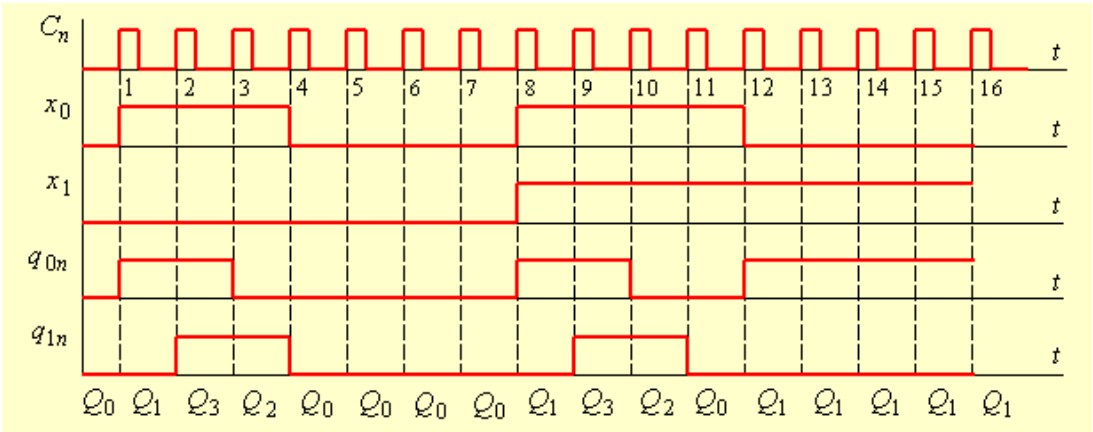


Рис. 9.6. Часові діаграми роботи автомата

Особливості аналізу скінченних автоматів з JK-тригерами

Синхронізовані скінченні автомати, що будуються на базі *JK*-тригерів, можуть також синхронізуватись у відповідності до процедури, яка була описана вище [2]. Різниця полягає лише в тому, що необхідно мати два збуджуючих рівняння (*Excitation equations*) для кожного тригера – одне для *J*-входу, друге – для входу *K*-.

Для отримання рівнянь переходів (*Transition equations*) кожне з них повинно використовувати характеристичне рівняння *JK*-тригера:

$$Q_{n+1} = \left(J_n \ \overline{Q_n} + \overline{K_n} \ Q_n \right) C_n \ .$$

На рис. 9.7 приводиться приклад скінченного автомата з використанням *JK*-тригерів.

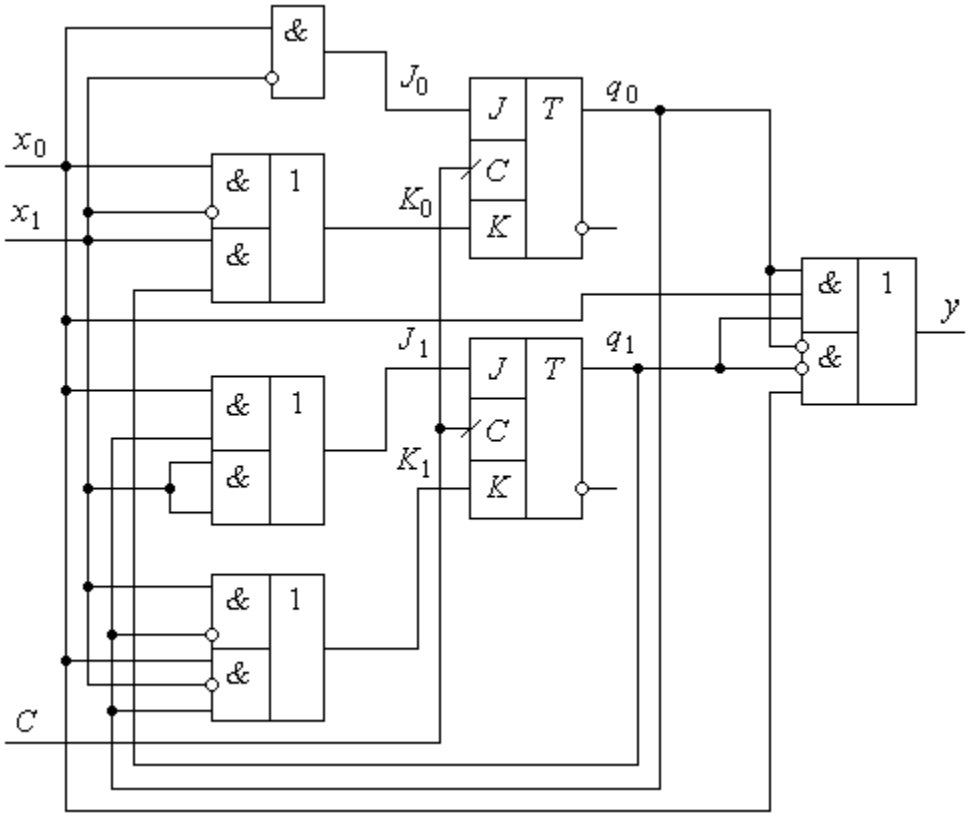


Рис. 9.7. Приклад скінченного автомата з використанням *JK*-тригерів

З рис. 9.7 можемо записати наступні рівняння збудження для входів *JK*-тригерів:

$$J_{0n} = x_0 \ \overline{x_1} \quad ; \qquad K_{0n} = x_0 \ \overline{x_1} + x_1 \ q_{1n} \quad ;$$

$$J_{1n} = x_0\, q_{0n} + x_1 \quad ; \qquad K_{1n} = x_1\, \overline{q_{0n}} + x_0\, \overline{x_1}\, q_{0n} \quad .$$

Підставляючи в характеристичне рівняння тригера, отримуємо рівняння переходів:

$$q_{0\,(n+1)} = C_n\ (x_0\ \ \overline{x_1}\ \ \overline{q_{0n}} + \overline{(x_0\ \ \overline{x_1} + x_1\ \ q_{1n})}\ \ q_{0n}); \qquad (9.9)$$

$$q_{1(n+1)} = C_n((x_0 q_{0n} + x_1)\overline{q_{1n}} + \overline{(x_1 \overline{q_{0n}} + x_0 \overline{x_1} q_{0n})} q_{1n}. \qquad (9.10)$$

Таблиця переходів, що базується на цих рівняннях, легко може бути побудована.

Рівняння (9.9) і (9.10) приведемо до досконалої форми:

$$q_{0(n+1)} = C_n(\overline{x_{0n}}\,\overline{x_{1n}} q_{0n}\,\overline{q_{1n}} + \overline{x_{0n}}\,\overline{x_{1n}} q_{0n}\,\overline{q_{1n}} + \overline{x_{0n}} x_{1n} q_{0n} q_{1n} + x_{0n}\,\overline{x_{1n}} q_{0n} q_{1n} + x_{0n} x_{1n}\,\overline{q_{0n}} q_{1n} + x_{0n} x_{1n} q_{0n}\,\overline{q_{1n}})$$

$$q_{1\,(n+1)} = C_n\ (x_0\ x_1\ q_{0n}\,\overline{q_{1n}} + x_0\ \overline{x_1}\ q_{0n}\,\overline{q_{1n}} + x_0\ x_1\,\overline{q_{0n}}\ q_{1n} + \overline{x_0}\ x_1\,\overline{q_{0n}}\ q_{1n} +$$
$$+ x_0\ \overline{x_1}\ q_{0n}\ q_{1n} + x_0\ x_1\,\overline{q_{0n}}\ \overline{q_{1n}} + \overline{x_0}\ x_1\ q_{0n}\ \overline{q_{1n}} + \overline{x_0}\ x_1\,\overline{q_{0n}}\ \overline{q_{1n}}) \quad .$$

Таблиця станів для обох функцій $q_{0(n+1)}$ і $q_{1(n+1)}$ має вигляд табл. 9.8, а карти Карно приведені на рис. 9.8.

Таблиця 9.8. Таблиця станів для функцій $q_{0(n+1)}$ і $q_{1(n+1)}$

N	x_0	x_1	q_{0n}	q_{1n}	$q_{0(n+1)}$	$q_{1(n+1)}$
0	0	0	0	0	0	0
1	0	0	0	1	0	0
2	0	0	1	0	1	0
3	0	0	1	1	0	0
4	0	1	0	0	0	1
5	0	1	0	1	0	1
6	0	1	1	0	0	1
7	0	1	1	1	1	0
8	1	0	0	0	0	0
9	1	0	0	1	1	0
10	1	0	1	0	0	1
11	1	0	1	1	1	1
12	1	1	0	0	0	1
13	1	1	0	1	1	1
14	1	1	1	0	1	1
15	1	1	1	1	0	0

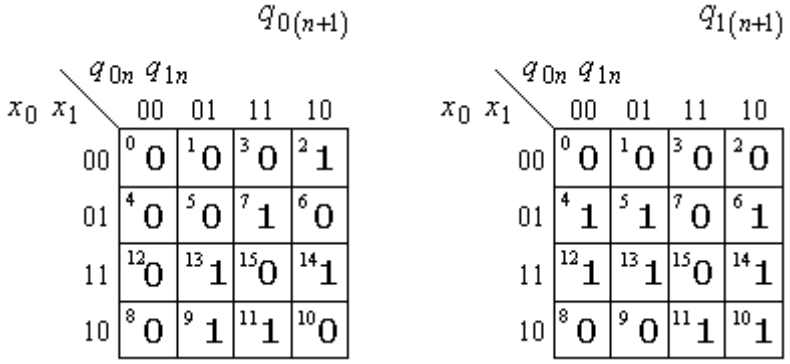


Рис. 9.8. Карти Карно

Складаємо таблицю кодування станів (табл. 9.9). Складаємо таблицю переходів (табл. 9.10).

Таблиця 9.9. Таблиця кодування станів

N	q_{0n}	q_{1n}	Q
0	0	0	Q_0
1	0	1	Q_1
2	1	0	Q_2
3	1	1	Q_3

Таблиця 9.10. Таблиця переходів

Q_n	Q_{n+1}			
	$x_0\ x_1$			
	00	01	10	11
Q_0	Q_0	Q_1	Q_0	Q_1
Q_1	Q_0	Q_1	Q_2	Q_3
Q_2	Q_2	Q_1	Q_1	Q_3
Q_3	Q_0	Q_2	Q_3	Q_0

Будуємо граф-схему переходів (рис. 9.9).

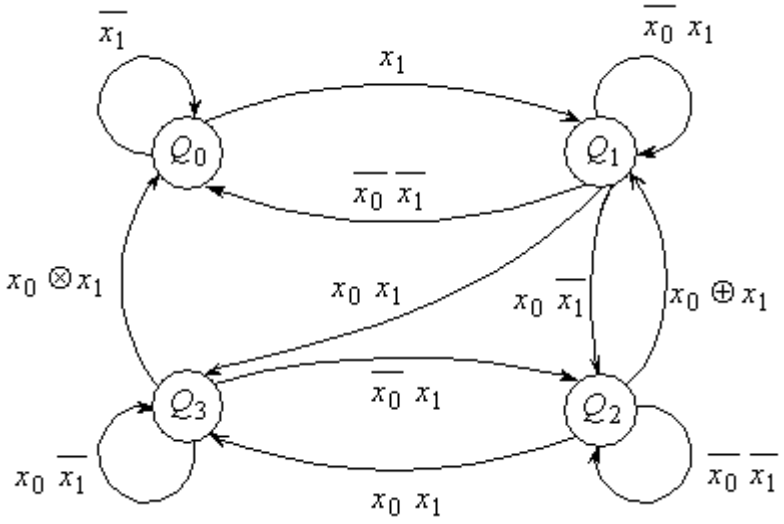


Рис. 9.9. Граф-схема переходів

Рівняння, що описує залежність вихідного сигналу, має вигляд:

$$y = x_0\ q_{0n}\ q_{1n} + x_1\ \overline{q_{0n}}\ \overline{q_{1n}} \quad .$$

Слід відмітити, що при використанні *JK*-тригерів можливо використати і інший шлях побудови таблиці переходів. Він полягає в тому, що створюються рівняння збудження і таблиця збудження відносно входів тригерів J_0, K_0, J_1, K_1, які потім використовуються в характеристичних рівняннях кожного тригера при побудові таблиці переходів.

Звернемося знову до прикладу схеми двійкового лічильника, що приведена на рис. 9.1, і розглянемо інший варіант його побудови з використання *JK*-тригерів. Для цього побудуємо таблицю станів з урахуванням сигналів на входах *J*- і *K*- кожного тригера і визначимося з логічною функцією L_1, що задаватиме сигнали *J* і *K*.

Оскільки нам необхідно мати два тригери, то позначимо їх, як і в схемі-прототипі, – через DD3 і DD4. Відповідні входи тригерів позначимо як J_3, K_3 і , K_4, а виходи – q_3 і q_4.

Таблиця будується в такій послідовності. В стовбці C_n задаємо послідовність тактових сигналів. У стовбцях q_3 і q_4 задаємо початкові

значення виходів тригерів до початку дії синхросигналу і відповідних інформаційних сигналів. Формально в цих стовпцях записується таблиця двійкового коду для двох змінних. У стовбцях $q_{3(n+1)}$ і $q_{4(n+1)}$ записуються значення виходів q_3 і q_4 після дії синхросигналу в відповідності до табл. 8.4 (с. 1). В стовбцях J_3, K_3, J_4, K_4 записуються ті значення входів, які приведуть до відповідної зміни кодових значень в стовбцях q_3 і q_4. Наприклад: у стовбці q_{3n} до початку дії синхросигналу маємо **“0”**, а після його дії – **“1”**. До такої зміни станів тригерів можуть привести сигнали $J_4 = 0$, $J_3 = 1$, $K_3 = 0$. Значення входу K_4 – байдуже.

Внаслідок побудови таблиці отримуємо залежності між значеннями виходів q_3 і q_4 від J_i, K_i (див. табл. 9.11).

Таблиця 9.11. Таблиця збудження тригерів

C_n	q_{4n}	q_{3n}	$q_{4(n+1)}$	$q_{3(n+1)}$	J_4	K_4	J_3	J_4
1	0	0	0	1	0	x	1	x
2	0	1	1	0	1	x	x	1
3	1	0	1	1	x	0	1	x
4	1	1	0	0	x	1	x	1

Для першого тригера бачимо, що значення J_3 і K_3 мають значення входів, або байдуже, або рівня **“1”**. Звідси знаходимо, що на його входах постійно може бути присутньою **“1”**, а, відповідно до алгоритму його роботи, це режим *T*-тригера.

Для визначення значень входів J_4 і K_4 також бачимо, що вони можуть бути однаковими, тобто $J_4 = K_4$, а подальший аналіз зв’язку J_4 з q_3 приводить до того, що знаходимо $J_4 = K_4 = q_{3n}$.

Звідси витікає схема лічильника на *JK*-тригерах (рис. 9.10).

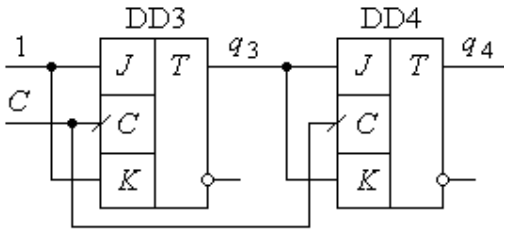


Рис. 9.10. Схема лічильника на *JK*-тригерах

Проаналізуємо його роботу. При $q_3 = q_4 = 0$ поява першого синхроімпульсу приведе до $q_3 = 1$; $q_4 = 0$. Другий синхроімпульс приведе до зміни стану DD4, тобто $q_4 = 1$, а також до зміни стану DD3: q_3 зміниться в нуль. Третій синхроімпульс знову приведе до зміни стану q_3 з **“0”** в **“1”** і, нарешті, четвертий імпульс приведе до обнуління обох тригерів.

Отриманий результат говорить про те, що складність комбінаційної схемотехніки скінченного автомата суттєво залежить від типу використовуваних тригерів. Якщо ж говорити більш точно, то складність комбінаційних схем залежить від уміння розробника використати логічні властивості використовуваних тригерів.

Перелік посилань

1. Рябенький В.М., Жуйков В.Я., Гулий В.Д. Цифрова схемотехніка: Навч. посібник. – Львів: Новий світ – 2000, 2009, – 736 с.

2. Жабін В.І., Жуков І.А., Клименко І.А., Ткаченко В.К. Прикладна теорія цифрових автоматів. – К. НАУ, 2007. – 364 с.

Лекція №10

Синтез скінченних автоматів

Задача синтезу (проектування) скінченних автоматів майже діаметрально протилежна задачі аналізу [1, 2].

Приблизно такі кроки можна описати як етапи проектування:

1. Задається закон функціонування автомата;
2. Мінімізується кількість внутрішніх станів автомата;
3. Кодуються стани автомата;
4. Визначаються функції збудження елементів пам’яті і функції виходів, а також забезпечується їх мінімізація;
5. Складається функціональна схема автомата в вибраному елементному базисі.

Перші три етапи відносяться до *абстрактного синтезу* автомата, при якому по значеннях вхідних і вихідних сигналів визначаються і кодуються його стани.

При абстрактному синтезі на основі словесного, табличного, за допомогою часових діаграм і інших засобів опису роботи автомату виявляється закон його функціонування і задаються функції переходів і виходів. На цьому етапі забезпечується мінімізація кількості станів і визначається кількість елементів пам’яті, необхідних для побудови автомата.

Решта станів забезпечують процедуру одержання структурних схем автоматів, що задаються кодованими таблицями входів і виходів. Така процедура називається *структурним синтезом*.

Задача структурного синтезу полягає у виборі типів елементарних автоматів (тригерів) і знаходження такого способу їх з’єднання між собою, який забезпечить функціонування автомата у відповідності до заданих кодованими таблицями переходів і виходів.

Основною задачею цього етапу є синтез комбінаційних схем з урахуванням вибраних тригерів, знаходження мінімальних форм функцій

збудження і виходів. Етап закінчується побудовою функціональної схеми автомата.

Розглянемо змістовну частину кожного з етапів на конкретних прикладах.

Етап 1. *Задання закону функціонування асинхронного автомата.*

Початковою інформацією для першого етапу може служити словесний опис алгоритму функціонування автомата або аналіз часових діаграм його роботи. Результатом цього етапу повинно бути отримання формалізованого опису закону функціонування в вигляді таблиць переходів, графів переходів і т. п. Практичний досвід показує, що зручно задавати автомат за допомогою *початкових таблиць переходів* (ПТП), оскільки їхня структура є найбільш простою.

ПТП характеризується тим, що в кожному її рядку є тільки один стійкий стан, а всі переходи між станами прості.

На основі аналізу зміни вхідних і вихідних сигналів можна вводити стани автомата, які характеризуються станом входу (сигналами x_0 і x_1 з вхідного алфавіту *X*) і станом виходу *Y* (сигналом *y*).

Різноманітним комбінаціям вхідних та вихідних сигналів припишемо відповідні номери, які характеризують всі можливі стани автомата. За початковий приймемо стан $Q_{00} = 0$, який характеризується значеннями вхідних сигналів $x_1 x_0 = 00$ і значенням вихідного сигналу $y = 0$. Стан $Q_{00} = 0$ є *стійким*, тобто таким, який не зміниться без дії вхідних сигналів. В таблиці ПТП, що будується, стійкі стани будемо виділяти дужками. *Нестійкі* стани автомата – це такі, які можуть змінюватися на стійкі без зміни вхідних сигналів. Такі стани в ПТП будемо позначати їх номерами без дужок.

Якщо на автомат, що знаходиться в стійкому стані Q_{00}, діятиме вхідний сигнал $x_1 x_0 = 01$, то він перейде в новий, але нестійкий, стан, який потім перейде в стійкий $Q_1 = (1)$. Нестійкий стан відповідає ситуації, коли в результаті зміни вхідного сигналу *X* змінились сигнали збудження тригерів, але ще не

змінився вихідний сигнал, тобто y залишився рівним нулю. По закінченні перехідних процесів автомат перейде в стійкий стан $Q_1 = (1)$, який характеризується вхідними сигналами $x_1 x_0 = 01$ і вихідним $y = 1$.

Переходу зі стійкого стану (0) в нестійкий 1 в ПТП відповідає переміщення в межах першого рядка з клітинки з координатами $x_1 x_0 = 00$ в клітинку з координатами $x_1 x_0 = 01$. Вихідний сигнал при цьому залишається незмінним: $y = 0$.

Переходу з нестійкого стану 1 в стійкий стан (1) в ПТП відповідає переміщення в межах одного стовпця з координатами $x_1 x_0 = 01$ з першого рядка в другий. Вихідний сигнал при цьому змінюється і приймає значення $y = 1$ (табл. 10.1).

Таблиця 10.1. Початкова таблиця переходів

Q \ X	$x_1 x_0$				y
	00	01	11	10	
0	(0)	1			0
1		(1)			1

Звідси маємо, що перехід з одного стійкого стану (0) в інший стійкий стан (1) відповідає двом фрагментам переходу: спочатку у межах рядка, а потім у межах стовпця.

З часової діаграми, приведеної на рис. 10.1, видно, що стійкий стан (1) при $x_1 x_0 = 00$ змінюється на стан (0), а потім під дією сигналу $x_1 x_0 = 10$ переходить в стан (2), і т.д.

Номери станів на часовій діаграмі зображають так, щоб кожна змінювана ситуація в сигналах *X*, *Y* мала свій номер і щоб були враховані всі зовнішні однакові ситуації, які вимагають спеціального запам’ятовування. Наприклад, якщо автомат згенерує третій імпульс, то кожен з трьох імпульсів і кожну з пауз необхідно позначити своїм номером.

Аналіз всієї часової діаграми дозволяє заповнити ПТП (див. табл. 10.2), але не повністю – частина клітинок залишається вільними.

Таблиця 10.2. Неповністю заповнена початкова таблиця переходів

Q \ X	$x_1 x_0$				y
	00	01	11	10	
0	(0)	1	—	2	0
1	0	(1)	4	—	1
2	0	—	3	(2)	0
3	—	5	(3)	2	0
4	—	1	(4)	2	1
5	0	(5)	3	—	0

Наприклад, у першому рядку ПТП залишилась незаповненою клітка з $X = 11$. Оскільки стан $Q = 0$ є стійким при $X = 00$, то перехід в новий стан, що відповідає $X = 11$, вимагає одночасної зміни обох вхідних сигналів x_1 і x_0, що практично неможливо. Оскільки перехід $00 \to 11$ виконати неможливо, в ПТП в відповідній клітинці проставляється риска, яка інформує про наявність невизначеного стану. Аналогічно виключаються інші переходи типу $00 \leftrightarrow 11$, а також $01 \leftrightarrow 10$.

Етап 2. *Мінімізація кількості внутрішніх станів автомата***.**

Ідея формальної процедури мінімізації кількості станів полягає в визначенні еквівалентних станів, де два стани є *еквівалентними*, якщо неможливо визначитись з різницею в їх функціонуванні, розглядаючи лише поточні і майбутні стани виходів автомата. Пара еквівалентних станів може бути замінена одним.

Існують детальні процедури, які дозволяють формально розв’язувати задачу мінімізації кількості станів, але на практиці вони рідко використовуються. Реально це обумовлено тим, що досвідчені інженери з електроніки проектують цифрові автомати з мінімальною кількістю станів без будь-яких проблем, не використовуючи формальні процедури. Більш того, часто мають місце ситуації, коли збільшення кількості станів не ускладнює проект, а, навпаки, спрощує його, і мінімізуючи процедури не надають необхідної користі.

Як правило, ПТП містить ряд збиткових внутрішніх станів, усунення яких не порушить алгоритму роботи автомата, але, в той же час, дозволить спростити його функціональну схему. В ряді випадків наявність збиткових станів не є очевидним фактом, тому для їх виявлення необхідно проводити детальний аналіз ПТП.

Скорочення частини станів асинхронного автомата базується на виявленні еквівалентних і псевдоеквівалентних станів, а також на виявленні і об'єднанні сумісних внутрішніх станів автомата (рядків ПТП).

Еквівалентними станами автомата називаються стійкі стани, які задовольняють наступним умовам:

1. Їм відповідає один і той же стан входу (тобто вони знаходяться в одному і тому ж стовбці ПТП);

2. Їм відповідає один і той же стан виходу (однакові вихідні сигнали автомата);

3. Будь-якій послідовності станів входу автомата відповідає одна і та ж сама послідовність станів його виходу, незалежно від того, який зі стійких станів автомата прийнято за початковий.

Для неповністю визначеного автомата, окрім поняття еквівалентності, існує і використовується поняття псевдоеквівалентності станів. *Псевдоеквівалентними* називаються такі два стійкі стани автомата, яким:

- відповідає один і той самий стан входу автомата;
- відповідають стани виходів автомата, між якими немає протиріччя;
- будь-якій послідовності станів входу автомата відповідають непротирічні послідовності станів його виходу, незалежно від того, який з цих стійких станів взятий за початковий. Іншими словами, серед пар послідовностей станів входу і виходу, що починаються з цих стійких станів, не повинно бути таких, що мають протиріччя.

Після об'єднання еквівалентних і псевдоеквівалентних станів завжди отримується таблиця, в кожному рядку якої, як і в ПТП, знаходиться лише один

стійкий стан. Скорочення кількості рядків, порівняно з ПТП, відбувається за рахунок зменшення кількості стійких станів. Але після об'єднання стійких станів можливо виконати подальше спрощення таблиці переходів за рахунок об'єднання сумісних внутрішніх станів автомата.

Під *сумісним внутрішнім станом* розуміють два або більше станів, яким відповідають рядки з розміщенням цифр в них, яке не має протиріччя, тобто такі рядки, в одному і тому ж стовбці яких знаходяться однакові цифри (або в одному рядку цифра, а в іншому – риска).

Рядки таблиці переходів, що відповідають сумісним станам, можуть бути об'єднані в одну. В об'єднаному рядку повинні бути стійкими стани у будь-якій з об'єднаних рядків. Тобто, при об'єднанні сумісних станів отримуємо таблицю, в кожному рядку якої може бути декілька стійких станів. При об'єднанні сумісних станів буде отриманий автомат, еквівалентний вихідному.

Об'єднання рядків з поєднаними стійкими станами виконується на основі наступних правил:

- два або більше рядків можуть бути об'єднані, якщо значення вихідних змінних (станів виходу) для цих рядків не мають протиріччя для автомата Мура і можуть бути будь-якими для автомата Мілі, а номери станів, що записані в одних і тих же стовпцях, співпадають між собою, або з прочерком;
- при об'єднанні станів з однаковими номерами в дужках і без них результуючий стан повинен бути у дужках;
- якщо при об'єднанні станів в одному з рядків знаходиться прочерк, а в іншій – номер стану, то в скороченій таблиці пишеться номер стану.

Слід відмітити, що виявляти і об'єднувати еквівалентні і псевдоеквівалентні стани перед виявлення і об'єднанням сумісних станів немає потреби. Можна починати з виявлення і об'єднання сумісних внутрішніх станів. При цьому об'єднуються, якщо вони є, еквівалентні і псевдоеквівалентні стани. Більш того, у ряді випадків першочергове об'єднання псевдоеквівалентних станів недовизначеного автомата може ускладнити

остаточне рішення, тобто може бути отриманий автомат з більшою кількістю внутрішніх станів, порівняно з автоматом, для якого зразу об’єднувались сумісні внутрішні стани.

Суттєву допомогу при аналізі сумісності рядків ПТП можуть надати діаграми (графи) сумісності внутрішніх станів, на яких сумісні стани розміщуються у вузлах, з’єднаних ненаправленими лініями. Множина внутрішніх станів є *сумісною*, якщо всі її стани є попарно сумісними. Таку множину можна замінити одним станом. Вибираючи мінімальну кількість таких множин, які охоплюють всі стани без їх повторення, отримують мінімальну кількість станів, яка достатня для реалізації автомата.

Для забезпечення знаходження максимальних множин сумісних станів іноді використовується трикутна таблиця сумісності.

Етап 3. *Кодування станів автомата.*

Найважливішою задачею, що має місце при кодуванні станів автомата є виключення значень його елементів пам’яті при заданих переходах. Таке кодування називається протигоночним. Найпростішим способом протигоночного кодування є сусіднє кодування, при якому два стани, що пов’язані між собою простими переходами кодуються наборами двійкових чисел, що відрізняються станом лише одного елемента пам’яті (ЕП). Основними вимогами до автомата, в якому забезпечується сусіднє кодування станів є:

–в графі автомата не повинно бути замкнутих контурів, що містять непарну кількість вершин;

–два сусідні стани другого порядку не повинні мати більше двох станів, що лежать між ними. При цьому під станами другого порядку маються на увазі два стани, шлях між якими по графу автомата складається з двох ребер (незалежно від орієнтації). На рис. 10.1 зображені два графа, які не задовольняють вказаним вимогам і не можуть бути закодовані сусідніми кодами.

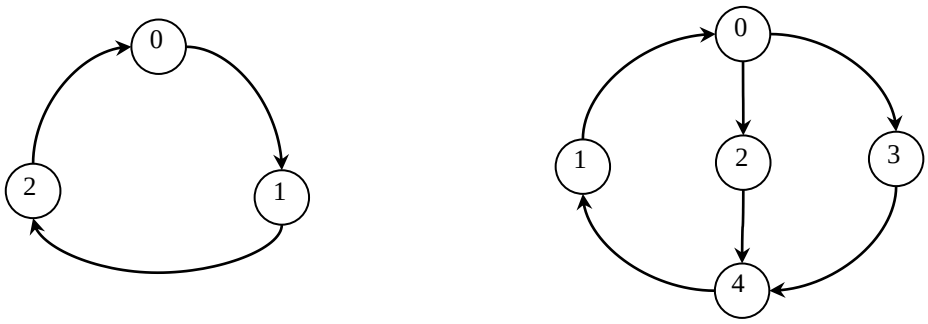


Рис. 10.1. Два графа, які не можуть бути закодовані сусідніми кодами

Граф переходів для генератора з забороною зображено на рис. 10.2. В розглянутому випадку граф має непарну кількість вершин, але контур незамкнений. Один з варіантів кодування станів приводить до табл. 10.3.

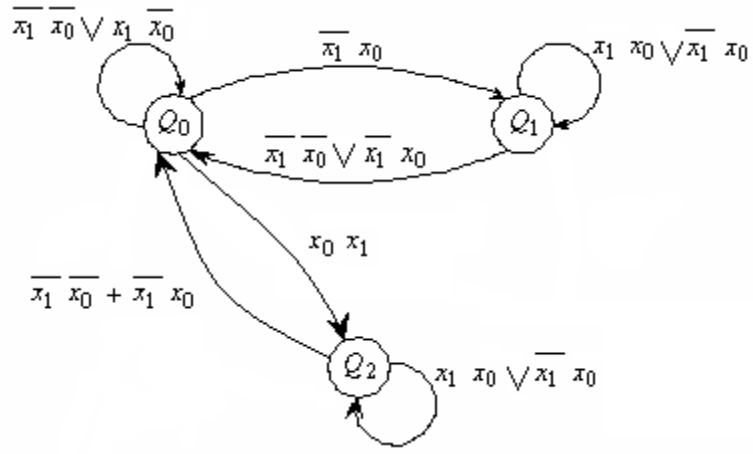


Рис. 10.2. Граф переходів для генератора з забороною

Таблиця 10.3. Один з варіантів кодування станів

Стани	Код	
	q_1	q_2
Q_0	0	0
Q_1	0	1
Q_2	1	0

Після закінчення кодування станів будується закодована таблиця переходів автомата, що проектується. Для цього в скороченій таблиці переходів замінюються назви переходів їх кодовими значеннями.

Етап 4. *Визначення функцій збудження елементів пам'яті і функцій виходів автомата.*

Функції збудження автомата залежать від використовуваних типів тригерів, що вибираються в якості ЕП.

Розглянемо послідовно на конкретних прикладах особливості визначення функцій збудження ЕП для кожного з типів тригерів.

Характеристична таблиця *RS*-тригера має вигляд, зображений в табл. 10.4, де зірочкою (*) зображені дозвільні значення відповідної змінної.

Таблиця 10.4. Характеристична таблиця *RS*-тригера

q_n	q_{n+1}	R_n	S_n
0	0	*	0
0	1	0	1
1	0	1	0
1	1	0	*

Визначимо функції збудження *RS*-тригерів. Для цього скористаємось закодованою таблицею переходів автомата, в якій замість переходів $Q_n \to Q_{n+1}$ підставимо значення сигналів збудження відповідних входів тригерів. Таблиця прийме вигляд, зображений у табл. 10.5.

Таблиця 10.5. Закодована таблиця переходів автомата з *RS*-тригерами

X / Q_n	$x_1\,x_0$															
	00				01				11				10			
$q_1\,q_0$	R_1	S_1	R_0	S_0	R_1	S_1	R_0	S_0	R_1	S_1	R_0	S_0	R_1	S_1	R_0	S_0
0 0	*	0	*	0	*	0	0	1	0	1	*	0	*	0	*	0
0 1	*	0	1	0	*	0	0	*	0	0	0	*	*	0	1	0
1 1	—				—				—				—			
1 0	1	0	*	0	0	*	*	0	0	*	*	0	1	0	*	0

Заповнення клітинок табл. 10.5 забезпечується безпосереднім використанням табл. 10.4. Для отримання функції збудження для кожного з входів тригерів переносимо їх значення в карти Карно (рис. 10.3, *а – г*).

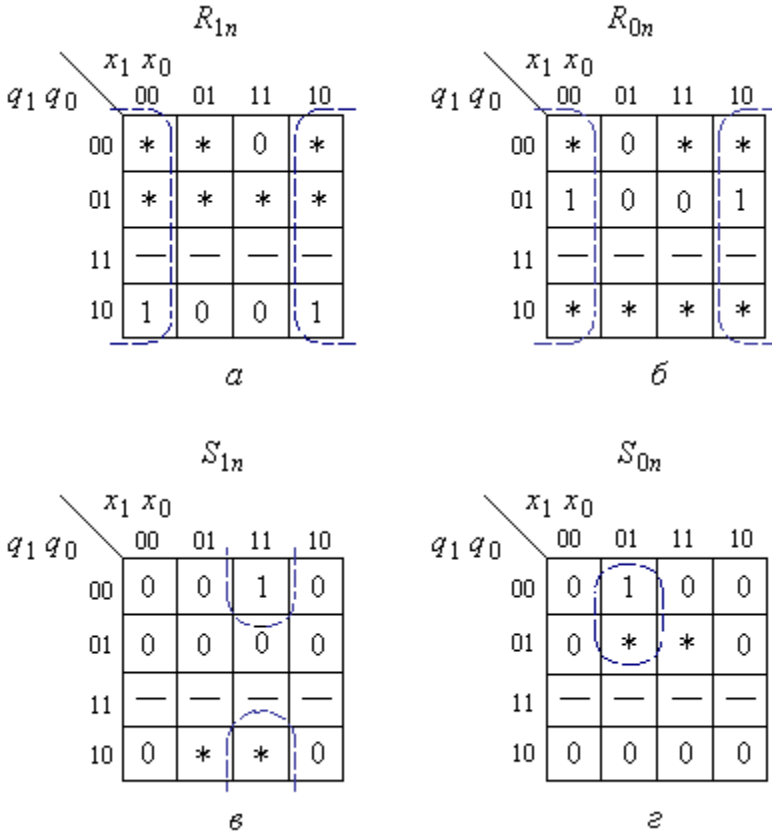


Рис. 10.3. Карти Карно

Мінімізуючи кожну з функцій збудження входів, отримуємо:

$$R_{1n} = \overline{x_0};$$

$$R_{0n} = \overline{x_0};$$

$$S_{1n} = \overline{q_0}\, x_1\, x_0;$$

$$S_{0n} = \overline{q_1}\,\overline{x_1}\, x_0.$$

Для визначення функції виходу карта Карно будується на основі скороченої таблиці переходів, в якій стійкі стани заміняються значеннями у відповідному рядку, а нестійкі – символом * – довільного значення функції. В результаті отримаємо карту, зображену на рис. 10.4.

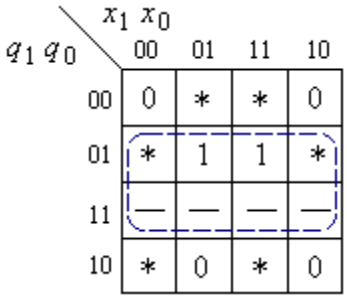


Рис. 10.4. Карта Карно функції виходу

З карти отримуємо мінімізоване значення функції виходу:

$$y = q_0 .$$

Етап 5. *Складання функціональної (принципової) схеми автомата в вибраному елементному базисі.*

Отримані в попередньому етапі вирази функцій збудження ЕП і функції виходу дають можливість легко розробити функціональну і принципову схеми автомата. Така схема приведена на рис. 10.5.

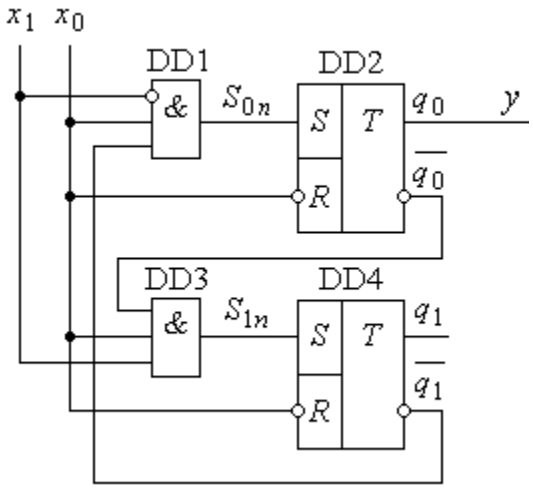


Рис. 10.5. Принципова схема автомата

Розглянемо її роботу. У нульовому стані при відсутності вхідних сигналів значення $q_0 = q_1 = 0$. В першому стані $x_1 = 0$, а $x_0 = 1$, тому, враховуючи, що $\overline{q_1} = 1$, сигнал S_{0n} з виходу DD1 буде поданий на тригер DD2 і встановить його вихід q в 1. При повторенні нульового стану тригер DD1 установиться в

початковий стан нульовим значенням вхідного сигналу x_0. В другому – третьому станах при $x_1 = 1$ на вході DD1 формується сигнал заборони для сигналу x_0. З цього короткого опису бачимо, що функціональна схема повністю відтворює часову діаграму, за якою спроектований автомат.

Приклад 10.1. Для автомата, заданого закодованою таблицею переходів (табл. 10.5), розробити функціональну схему з використанням *JK*-тригерів асинхронного типу.

Розв'язання. На основі таблиці переходів *JK*-тригера побудуємо характеристичну таблицю (табл. 10.6).

Таблиця 10.6. Характеристична таблиця

$q_n \to q_{n+1}$		J_n	K_n
0	0	0	*
0	1	1	*
1	0	*	1
1	1	*	0

Розробимо таблицю для визначення функції збудження тригерів автомата. Після підстановки в табл. 10.5 значень входів тригерів з характеристичної таблиці (табл. 10.6) отримаємо табл. 10.7.

Таблиця 10.7. Таблиця для визначення функції збудження тригерів автомата

X / Q_n	$x_1\,x_0$															
	00				01				11				10			
$q_1\,q_0$	J_1	K_1	J_0	K_0	J_1	K_1	J_0	K_0	J_1	K_1	J_0	K_0	J_1	K_1	J_0	K_0
0 0	0	*	0	*	0	*	0	*	0	*	1	*	0	*	0	*
0 1	1	*	*	0	0	*	*	0	0	*	*	0	0	*	*	0
1 1	*	0	*	0	*	0	*	1	*	0	*	1	*	0	*	0
1 0	—	—	—	—	—	—	—	—	*	0	0	*	0	1	0	*

Карта функції виходу будується на основі табл. 10.3 (рис. 10.6, *а*).

Будуємо карти Карно для функцій збудження *JK*-тригерів (рис. 10.6, *б* – *д*).

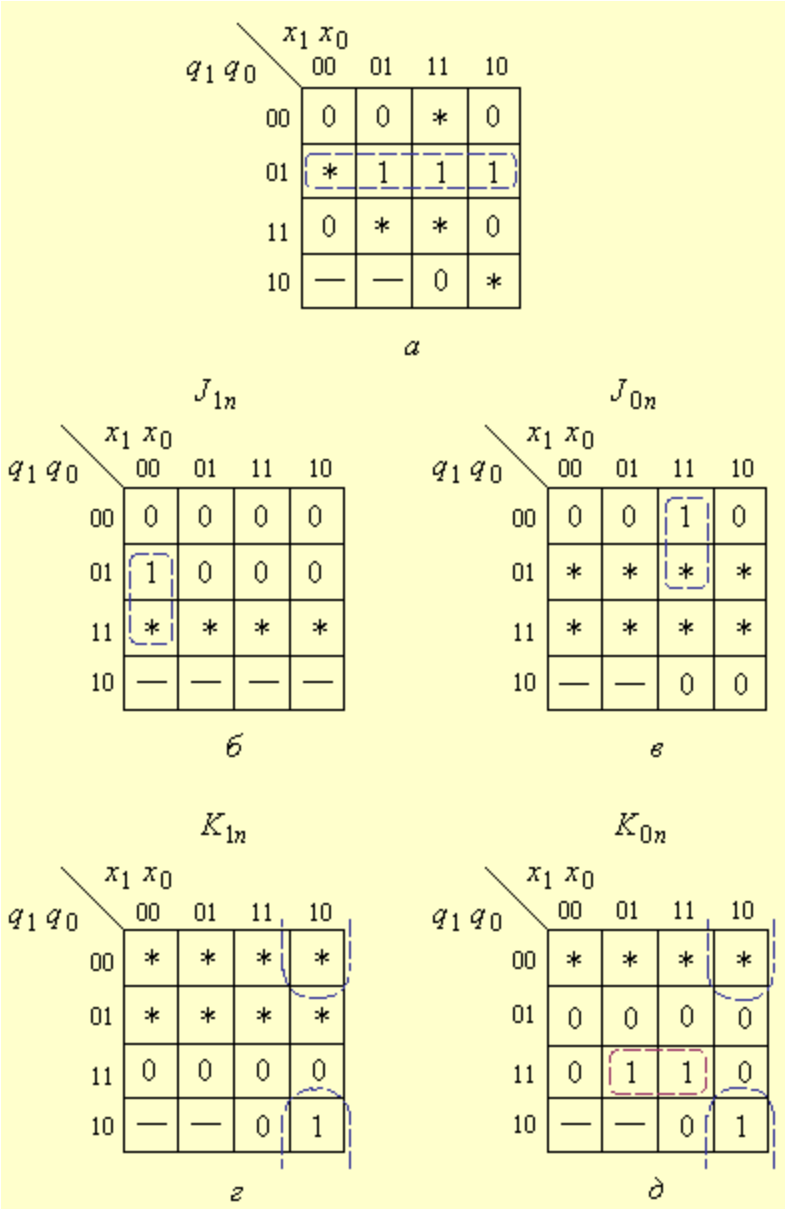


Рис. 10.6. Карти Карно

Мінімізовані функції збудження тригерів мають вигляд:

$J_{1n} = \overline{q_0}\ \overline{x_1}\ \overline{x_0}$;

$J_{0n} = \overline{q_1}\ x_1\ x_0$;

$K_{1n} = \overline{q_0}\ x_1\ x_0$;

$K_{0n} = q_1\ q_0\ x_0 + \overline{q_0}\ x_1\ x_0$.

З карти Карно функції виходу отримуємо:

$y = \overline{q_1}\ q_0$.

За отриманими функціями будується комбінаційна схема автомата і його функціональна схема (рис. 10.7).

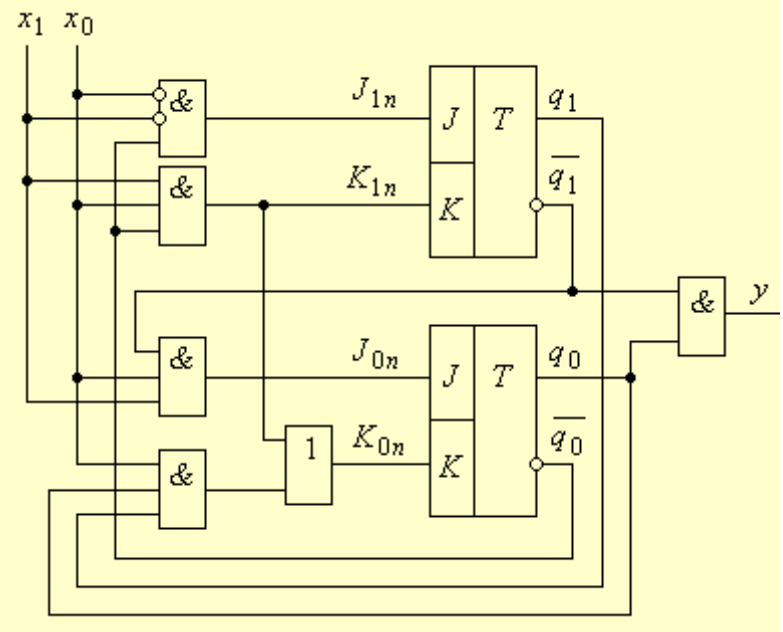


Рис. 10.7. Функціональна схема автомата

Перелік посилань

1. Рябенький В.М., Жуйков В.Я., Гулий В.Д. Цифрова схемотехніка: Навч. посібник. – Львів: Новий світ – 2000, 2009, – 736 с.

2. Жабін В.І., Жуков І.А., Клименко І.А., Ткаченко В.К. Прикладна теорія цифрових автоматів. – К. НАУ, 2007. – 364 с.

Лекція №11

Синтез асинхронних імпульсних автоматів

Асинхронними імпульсними автоматами називаються такі, на які вхідні сигнали впливають лише на короткі інтервали часу зміни їх рівня з **“0”** в **“1”** або навпаки [1, 2]. Строге описання законів функціонування даних автоматів можливо лише на основі їх логічних властивостей, що визначаються функціями переходів. Для аналітичного формування закону функціонування асинхронних імпульсних автоматів вводиться оператор переходу d. Використання оператора переходу до потенційних сигналів $x(t)$ дає імпульсний сигнал $dx(t)$ (рис. 11.1, *а*, *б*), який приймає одиничне значення в момент переходу $x(t)$ з **“0”** в **“1”**. Для опису асинхронних імпульсних автоматів, що спрацьовують по передньому фронту, слід використовувати вираз для імпульсного сигналу у вигляді $d\bar{x}(t)$ (рис. 11.1, *в*).

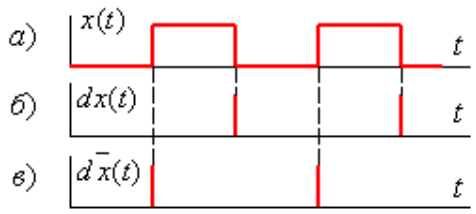


Рис. 11.1. Використання оператора переходу до потенційних сигналів

Задача синтезу асинхронних імпульсних автоматів може бути зведена до задачі синтезу асинхронних потенційних автоматів шляхом розробки методики перетворення функцій переходів, в які входять оператори d, в функції переходів виходів, що не містять операторів d. Таке перетворення називається *інтегруванням функції переходів імпульсного автомата* або *інтегруванням імпульсного автомата*.

Найбільш просто імпульсні автомати інтегруються табличним методом, оскільки є можливість побудувати таблицю переходів асинхронного потенційного автомата безпосередньо за функцією переходу імпульсного автомата.

Розглянемо це на прикладі синтезу імпульсного *JK*-тригера, повна функція переходів якого задається табл. 11.1.

Таблиця 11.1. Повна функція переходів

dJ	dK	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Вона практично не відрізняється від відповідної функції потенційного тригера, тому аналітичний вираз для закону функціонування (характеристичне рівняння) має вигляд:

$$Q_{n+1} = d\,J\overline{Q}_n + d\,\overline{K}Q_n\,.$$

Виконуємо інтегрування характеристичного імпульсного *JK*-тригера побудовою ПТП асинхронного потенційного автомата, еквівалентного синтезованому імпульсному тригеру.

Якщо сигнали *J*- і *K*- не змінюються, то $dJ = dK = 0$, тому в стійких станах, відповідно до характеристичного рівняння тригера, має місце рівність $Q_{n+1} = Q_n$. З цього витікає, що ПТП повинна мати в кожному стовбці, що відповідає стану входу, по два стійкі стани, які відрізняються значенням функції виходу $Q = 0$ або $Q = 1$. Оскільки можливих станів входу чотири, то число стійких станів асинхронного потенційного автомата дорівнює 8. Пронумерувавши їх у ряд, починаючи з нуля, заповнимо діагональ в ПТП (табл. 11.2). Потім, використовуючи характеристичне рівняння тригера, знайдемо переходи між стійкими станами.

Таблиця 11.2. Початкова таблиця переходів

Q \ $X(J,K)$	JK			
	00	01	11	10
0	(0), 0	2	4	6
1	(1), 1	3	5	7
2	0	(2), 0	5	6
3	0	(3), 1	5	7
4	1	3	(4), 0	6
5	1	3	(5), 1	7
6	1	3	4	(6), 0
7	0	3	4	(7), 1

Нехай, наприклад, початковим є стан входу $x=1,0$, якому відповідають стійкі стани Q_2 і Q_3. Допустимо, що стан входу змінюється на $x=1,0$. Оскільки при цьому (в відповідності до рис. 5.40, *б, в*), J змінюється в напрямку $0\to1$, а K $1\to0$, то $dJ=0$, а $d\overline{K}=1$, то автомат повинен перейти в той внутрішній стан, якому відповідає стан виходу, який визначається характеристичним рівнянням тригера $Q_{n+1}=0\cdot\overline{Q}_n+\overline{1}\cdot Q_n=0$, тобто, незалежно від вихідного стійкого стану (Q_2 або Q_3), він повинен перейти в стійкий стан Q_4. Аналогічно заповнюється решта клітинок ПТП, які відповідають нестійким станам у табл. 11.2.

Наступним етапом проектування є абстрактний і структурний синтез за методикою синтезу асинхронних потенційних автоматів.

Особливості синтезу синхронних автоматів

У синхронних автоматах, на відміну від асинхронних, використовується тактовий сигнал *C*, що задає моменти часу, в які можуть мати місце зміни внутрішніх станів автомата [3, 4]. У синхронних автоматах здебільшого використовуються *D*- або *JK*-тригери. З особливостей роботи синхронних тригерів витікає, що зміна інформаційних сигналів не може викликати зміни стану ЕП. Звідси витікає, що особливість синхронних тригерів, яка полягає в тому, що інформаційні сигнали лише керують змінами станів тригерів, переноситься і на синхронні автомати. Тому, оскільки на всі тригери автомата

синхросигнал подається одночасно, то вони і спрацьовують одночасно, незалежно від моменту подачі інформаційних сигналів.

У той же час, для синхронних автоматів, як відмічалось раніше, характерним є дискретний час, і стан автомата можна розглядати лише в дискретні моменти часу. Це дозволяє виключити з розгляду перехідні процеси, що протікають у комбінаційній схеми при зміні її вхідних сигналів.

Оскільки таблиці станів синхронних і асинхронних *D*- і *JK*-тригерів у цілому співпадають, то методи структурного синтезу синхронних автоматів ідентичні методам, використовуваним для синтезу асинхронних автоматів. У той же час, наявність синхронного режиму роботи всіх ЕП приводить до появи ряду особливостей проектування:

- комбінаційні схеми можуть проектуватись не вільними від гонок;
- допускаються несусідні зміни станів входу;
- допускаються критичні змагання ЕП, тобто допускається несусіднє кодування внутрішніх станів.

У великій кількості практичних задач із синтезу синхронних автоматів перехід від словесного опису до стандартних способів задання функцій переходів і виходів буває дуже простим. Тому для синхронних автоматів часто можливо, минаючи етап абстрактного синтезу, зразу ж перейти до етапу структурного синтезу. Таке спрощення процедури синтезу є наслідком використання синхронних ЕП, які виконують значно складніші функції, ніж асинхронні потенційні тригери.

Приклад 11.1. Синтезувати лічильник за модулем **5**, що задається таблицею переходів (табл. 11.2) з використанням *D*- і *JK*-тригерів. Характеристичні таблиці переходів для *D*- і *JK*-тригерів приведені в табл. 11.3.

Таблиця 11.3. Характеристична таблиця переходів для *D*-тригера

N	Q_n			Q_{n+1}		
	q_2	q_1	q_0	q_2	q_1	q_0
Q_0	0	0	0	0	0	1
Q_1	0	0	1	0	1	0
Q_2	0	1	0	0	1	1
Q_3	0	1	1	1	0	0
Q_4	1	0	0	0	0	0

Таблиця 11.4. Характеристична таблиця переходів для *JK*-тригера

Q_n	Q_{n-1}	D_n	J_n	K_n
0	0	0	0	*
0	1	1	1	*
1	0	0	*	1
1	1	1	*	0

Розв’язання. Будуємо таблицю збудження для *D*-входів в *D*-тригерах (табл. 11.5).

Таблиця 11.5. Таблиця збудження для *D*-входів в *D*-тригерах

$Q_n \to Q_{n+1}$	Q_n			D_n		
	q_2	q_1	q_0	D_2	D_1	D_0
$Q_0 \to Q_1$	0	0	0	0	0	1
$Q_1 \to Q_2$	0	0	1	0	1	0
$Q_2 \to Q_3$	0	1	0	0	1	1
$Q_3 \to Q_4$	0	1	1	1	0	0
$Q_4 \to Q_0$	1	0	0	0	0	0

З цієї таблиці отримуємо функціональні залежності входів *D*-тригерів $D_0 \dots D_2$ від попереднього стану виходів тих же тригерів (функція збудження):

$$D_0 = \overline{q_2}\,\overline{q_0}\ ;$$

$$D_1 = \overline{q_2}\left(q_1 \oplus q_0\right);$$

$$D_2 = \overline{q_2}\,q_1\,q_0\ .$$

За отриманими функціями збудження розробляємо функціональну схему лічильника (рис. 11.2).

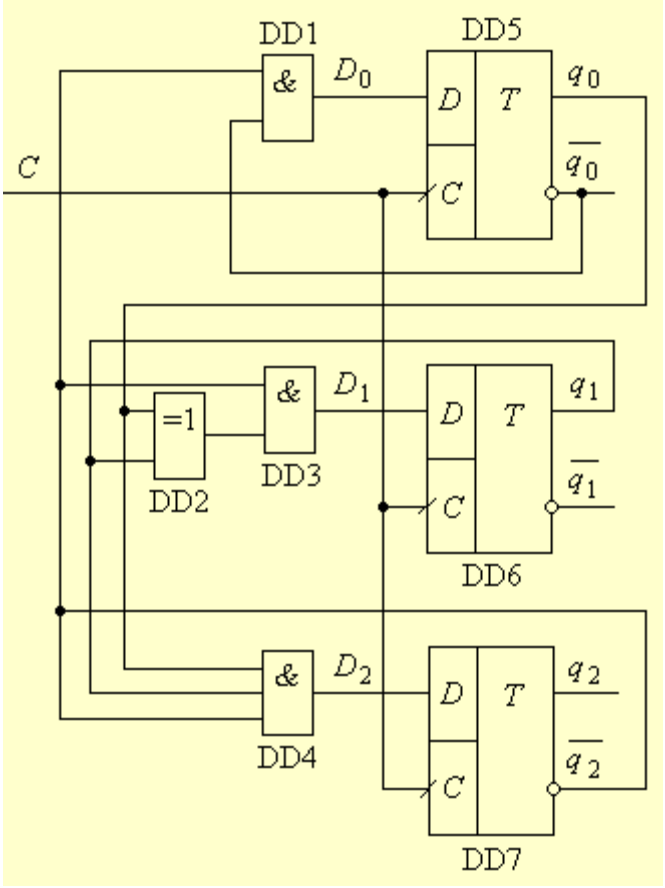


Рис. 11.2. Функціональна схема лічильника

У початковому стані Q_0 всі прямі виходи лічильника $q_2\,q_1\,q_0 = 000$, а їх інверсні еквіваленти $\overline{q_2}\,\overline{q_1}\,\overline{q_0} = 111$. Тому на вході тригера DD5 маємо $D_0 = 1$, а на входах тригерів DD6 і DD7, відповідно, $D_1 = 0$ і $D_2 = 0$. Тому при подачі синхросигналу DD5 записує на своєму прямому вході $q_0 = 1$. Це приводить до зміни значень входів перед другим тактовим імпульсом. Тепер ми маємо, що $D_0 = 0$, $D_1 = 1$, $D_2 = 0$ і по фронту C обнуляється q_0, а $q_1 = 1$. Схема циклічно змінюватиме свої стани у відповідності до алгоритму роботи.

Тепер розглянемо особливості синтезу з використанням *JK*-тригерів. Будуємо таблицю збудження для входів *JK*-тригерів (табл. 11.6).

Таблиця 11.6. Таблиця збудження для для входів *JK*-тригерів

$Q_n \to Q_{n+1}$	Q_n			$J_n\ K_n$					
	q_2	q_1	q_0	J_2	K_2	J_1	K_1	J_0	K_0
$Q_0 \to Q_1$	0	0	0	0	*	0	*	1	*
$Q_1 \to Q_2$	0	0	1	0	*	1	*	*	1
$Q_2 \to Q_3$	0	1	0	0	*	*	0	1	*
$Q_3 \to Q_4$	0	1	1	1	*	*	1	*	1
$Q_4 \to Q_0$	1	0	0	*	1	0	*	0	*

Після мінімізації знаходимо:

$$J_0 = \overline{q_2}\ ; \qquad K_0 = 1\ ;$$

$$J_1 = q_0\ ; \qquad K_1 = q_0\ ;$$

$$J_2 = q_1\, q_0\ ; \qquad K_2 = 1\ .$$

Схема лічильника з використанням синхронних тригерів приведена на рис. 11.3.

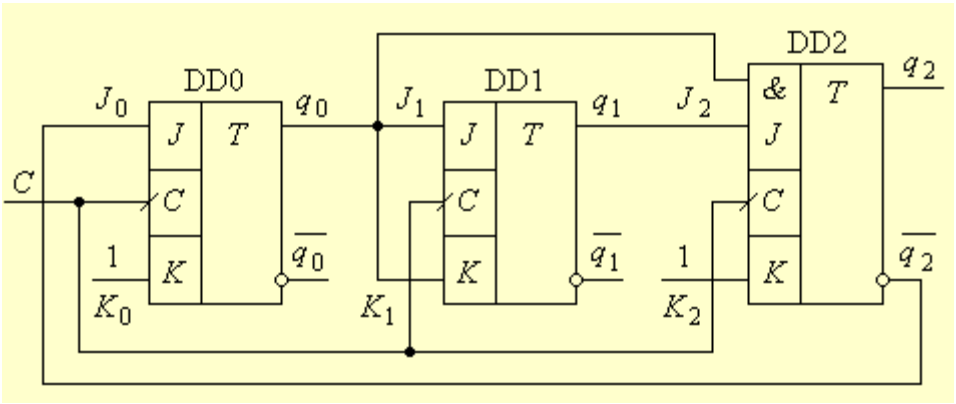


Рис. 11.3. Схема лічильника з використанням синхронних тригерів

Робота схеми повністю відповідає заданому алгоритму.

Використання теореми Шенона при синтезі скінченних автоматів на основі JK-тригерів

Як відомо, характеристичне рівняння *JK*-тригера має вигляд [1, 2]:

$$Q_{n+1} = J_n\, \overline{Q_n} + \overline{K_n}\, Q_n\ .$$

Звернемо тепер увагу на теорему Шенона:

$$f(x_{n-1},\ldots,x_i,\ldots,x_0)=\overline{x_i}\cdot f(x_{n-1},\ldots,0,\ldots,x_0)+x_i\cdot f(x_{n-1},\ldots,1,\ldots,x_0). \quad (11.1)$$

Зрозуміло, що функції $f(x_{n-1},\ldots,0,\ldots,x_0)$ і $f(x_{n-1},\ldots,1,\ldots,x_0)$ – це різні функції, тому позначимо їх відповідно як функції f_1 та f_2, а формулу (11.1) перепишемо у вигляді:

$$f(x_{n-1},\ldots,x_i,\ldots,x_0)=\overline{x_i}\cdot f_1+x_i\cdot f_2.$$

Тепер звернемо увагу на той факт, що функцію переходів будь-якого тригера скінченного автомата можемо розглядати як функцію вигляду:

$$Q_{(n+1)_i}=f(x_0\ldots x_n;Q_0\ldots Q_n).$$

тому, розглядаючи її в відповідності до теореми Шенона, відносно стану заданого *i*-го тригера, отримуємо:

$$Q_{(n+1)_i}=f_{1i}\cdot\overline{Q_i}+f_{2i}\cdot Q_i, \quad (11.2)$$

де, зрозуміло, функції f_{1i} та f_{2i} вже не містять відповідно змінних Q_i та $\overline{Q_i}$.

Порівнюючи тепер характеристичне рівняння *JK*-тригера з формулою (11.2), бачимо, що:

$$f_{1i}=J \text{ і } \overline{f_{2i}}=K .$$

Звідси витікає, що, прийнявши в функції переходів тригера $Q_i=0$, отримуємо функцію збудження для входу *J*-:

$$J_i=f_{1i}=Q_{(n+1)_i}\Big|_{Q_i=0},$$

а, прийнявши $Q_i=1$, отримуємо функцію збудження для входу *K*-:

$$\overline{K_i}=f_{2i}=Q_{(n+1)_i}\Big|_{Q_i=1} \text{ або } K_i=\overline{f_{2i}}.$$

Приклад 11.2. Розробити автомат, який би в залежності від керуючого сигналу *G* міг працювати як двійковий лічильник $(G=0)$ або як лічильник з відліком в відповідності до коду Грея з $M=8$ $(G=1)$. Автомат розробити з використанням *JK*-тригерів.

Розв’язання. Таблиця переходів автомата має вигляд табл. 11.7.

Таблиця 11.7. Таблиця переходів автомата

M	Q_{2n}	Q_{1n}	Q_{0n}	$Q_{2(n+1)}$	$Q_{1(n+1)}$	$Q_{0(n+1)}$
0	0	0	0	0	0	1
0	0	0	1	0	1	0
0	0	1	0	0	1	1
0	0	1	1	1	0	0
0	1	0	0	1	0	1
0	1	0	1	1	1	0
0	1	1	0	1	1	1
0	1	1	1	0	0	0
1	0	0	0	0	0	1
1	0	0	1	0	1	1
1	0	1	1	0	1	0
1	0	1	0	1	1	0
1	1	1	0	1	1	1
1	1	1	1	1	0	1
1	1	0	1	1	0	0
1	1	0	0	0	0	0

Рівняння переходів кожного з тригерів:

$$Q_{2(n+1)} = \overline{M} \cdot (\overline{Q_{2n}}\, Q_{1n}\, Q_{0n} + Q_{2n}\, \overline{Q_{1n}}\, \overline{Q_{0n}}) + M \cdot (\overline{Q_{2n}}\, Q_{1n}\, \overline{Q_{0n}} + Q_{2n}\, Q_{1n}\, Q_{0n});$$

$$Q_{1(n+1)} = \overline{M}\, Q_{2n}\, \overline{Q_{1n}}\, Q_{0n} + M\, \overline{Q_{2n}}\, Q_{1n}\, Q_{0n};$$

$$Q_{0(n+1)} = \overline{M} \cdot (\overline{Q_{2n}}\, Q_{1n}\, \overline{Q_{0n}} + Q_{2n}\, \overline{Q_{1n}}\, \overline{Q_{0n}}) + M \cdot (\overline{Q_{2n}}\, \overline{Q_{1n}}\, Q_{0n} + Q_{2n}\, Q_{1n}\, Q_{0n}).$$

Функції збудження для входів J- і K-:

$$J_2 = \overline{M}\, Q_{1n}\, Q_{0n} + M\, Q_{1n}\, \overline{Q_{0n}};\ \overline{K_2} = \overline{M}\, \overline{Q_{1n}}\, \overline{Q_{0n}} + M\, Q_{1n}\, Q_{0n};$$

$$J_1 = \overline{M}\, Q_{2n}\, Q_{0n};\ \overline{K_1} = M\, \overline{Q_{1n}}\, Q_{0n};$$

$$J_0 = \overline{M}\, \overline{Q_{2n}}\, Q_{1n} + M\, Q_{2n}\, \overline{Q_{1n}};\ \overline{K_0} = M\, (\overline{Q_{2n}}\, \overline{Q_{1n}} + Q_{2n}\, Q_{1n}).$$

Перелік посилань

1. Рябенький В.М., Жуйков В.Я., Гулий В.Д. Цифрова схемотехніка: Навч. посібник. – Львів: Новий світ – 2000, 2009, – 736 с.

2. Жабін В.І., Жуков І.А., Клименко І.А., Ткаченко В.К. Прикладна теорія цифрових автоматів. – К. НАУ, 2007. – 364 с.

Лекція №12

Колективна поведінка автоматів

У 1960-і роки з'явилися роботи А.А. Ляпунова, М.Л. Цетліна, В.І. Варшавского, Д.А. Поспелова, пов'язані з побудовою моделей колективної поведінки. Ідеї, що розвивалися вченими про еволюцію технічних систем і управління ними випередили свій час і стали по-справжньому затребуваними лише в самому кінці XX – початку XXI ст. Про це говорить публікація нового видання популярної монографії В.І. Варшавского і Д.А. Поспелова «Оркестр грає без диригента», вперше вийшла в світ в 1984 р. В основі базових понять і моделей теорії колективної поведінки і управління лежить гіпотеза простоти, висловлена М.Л. Цетлін: «будь-яка досить складна поведінка складається із сукупності простих поведінкових актів». На цій гіпотезі побудовані автоматні моделі: автомат з лінійною тактикою М.Л. Цетлін, «довірливий» автомат В.І. Крінского. Учень М.Л. Цетлін В.Ю. Крилов, працюючи зі своїм учителем, з ентузіазмом вносив нові ідеї. Ними була розроблена нова модель автомата - «обережний» автомат Крилова. Були також розроблені модель автомата зі змінною структурою, призначеного для функціонування в динамічних середовищах, загальна концептуальна схема колективної поведінки автоматів, що взаємодіють з середовищем [1, 2].

У роботах Гінзбурга, Крилова, Цетліна була запропонована мова і апарат моделювання – стохастичні автомати, за допомогою яких вдалося побудувати нормативні моделі прийняття рішень.

Стохастичний автомат – математична модель автомата, який володіє пам'яттю про прийняті раніше рішення, і здатного приймати рішення на підставі ймовірності виграшу при тій чи іншій дії.

В якості об'єктів, поведінка яких буде вивчатись, використовуються скінченні автомати, тобто об'єкти, що мають кінцеве число внутрішніх станів φ_i, $i = 1, 2, \ldots, n$, кінцеве число вхідних сигналів S_l, $l = 1, 2, \ldots, g$ і кінцеве

число вихідних сигналів f_m, $m = 1, 2, \ldots, k$. Автомат задається канонічними рівняннями:

$$\varphi(t+1) = \Phi[\varphi(t), S(t+1)], \tag{12.1}$$

$$f(t) = \mathrm{F}[\varphi(t)]. \tag{12.2}$$

Рівняння 12.1 визначає зміну внутрішніх станів автомата під впливом вхідної змінної $S(t)$, а рівняння 12.2 – залежність вихідного сигналу від внутрішнього стану автомата.

Позначимо через n глибину пам'яті автомата. Сенс цього параметра полягає в наступному. Чим більше n, тим більше інерційним є автомат, бо тим більша послідовність програшів змушує його до зміни дій. Інтуїтивно ясно, що, чим більше інерційність автомата, тим ближче він до того, щоб, вибравши найкращу в даному середовищі дію, продовжувати виконувати лише її. З ростом глибини пам'яті зростає доцільність поведінки автомата в стаціонарних середовищах. І, навпаки, при малому значенні n функціонування автомата більш схильне до дії програшів, які можуть перевести автомат в нову дію. Конструкція автомата, розглянута нами вище, була названа М.Л. Цетлін автоматом з лінійної тактикою. Ця проста в технічній реалізації система вирішує складну задачу про доцільну поведінку в будь-якому заздалегідь не фіксованому стаціонарному середовищі. Факт цей викликає глибоке здивування. Наскільки ж простими виявляються конструкції, здатні виконувати процедури адаптації, що представляються на перший погляд досить складними. Але виявляється, що доцільна поведінка це ще не все. М.Л. Цетлін показав, що якщо мінімальна імовірність P_{min} не перевищує 0.5, то при зростанні величини n ми отримаємо послідовність автоматів з лінійною тактикою з усе дедалі більшою глибиною пам'яті, яка є асимптотично оптимальною. Це означає, що при $n \to \infty$ має місце умова $M(q,E) \to M$, де M – мінімальний сумарний штраф, який можна отримати в даному стаціонарному випадковому середовищі. Таким чином, у багатьох таких середовищах конструкція, запропонована М. Л. Цетлін,

забезпечує при досить великих значеннях поведінку, яка скільки завгодно близька до найкращої [1, 2].

Наведемо ще одну конструкцію автомата (запропонованого В.Ю. Криловим), що забезпечує доцільну поведінку в будь-якому стаціонарному середовищі. При цьому є можливість побудови асимптотично оптимальної послідовності автоматів, що дозволяє отримувати мінімальний можливий штраф в даному середовищі з будь-якою наперед заданою точністю. На відміну від раніше розглянутих конструкцій цей автомат буде не детермінованим, а імовірнісним. Влаштований він подібно до автомату з лінійної тактикою. При надходженні сигналу «виграш» зміна станів в ньому відбувається аналогічно автомату Цетліна.

Але при сигналі «програш» такий автомат не поспішає міняти стан. Спочатку він «підкидає монетку» і по результату підкидання або переходить в інший стан, або зберігає той стан, в якому автомат отримав сигнал «штраф». Ця конструкція, запропонована В.Ю. Криловим, може бути названа «обережним» автоматом.

Автомат Крилова має n положень (обсяг пам’яті). Перехід від положення до положення залежить від виграшу або втрати і номера положення на попередньому кроці. Такий автомат є асимптотично оптимальним в стохастичному середовищі, що задається за допомогою ймовірностей виграшу (P_1, ..., P_m). Це означає, що граничне математичне очікування виграшу автомата дорівнює максимальній ймовірності P_i в середовищі f_i, $i = 1, ..., m$:

$$lim\ M = P_{imax}.$$

В.Ю. Криловим була вперше розглянута однорідна гра багатьох однакових автоматів. Тим самим було відкрито новий напрям в застосуванні автоматних моделей: моделювання колективної поведінки за допомогою автоматів, асимптотично оптимальних в стаціонарному випадковому середовищі. Приклад колективної поведінки автоматів автори назвали грою в розміщення [1, 2].

Ігри в розміщення виявилися зручною моделлю для знаходження можливості організації складної колективної поведінки автоматів [1]. Сьогоднішня лекція присвячена, зокрема, вивченню надійності колективної поведінки і опису прикладу застосування методів моделювання гри в розміщення для вирішення так званої задачі про розподіл обчислювальної техніки.

Нагадаємо визначення *гри в розміщення*. В цій грі беруть участь n однакових автоматів. Кожен автомат A_α ($\alpha = 1, 2, \ldots, n$) має $k > n$ стратегій f_1, f_2, …, f_k. Гра задається набором k чисел $1 \geq \alpha_1 \geq \alpha_2 \geq \ldots \geq \alpha_k \geq 0$, величина α_j називається потужністю стратегії f_j.

Якщо автомат A_α в момент t вибрав стратегію $f_{i\alpha}$ ($\alpha = 1, 2, \ldots, n$), то набір $f(t)$ = (f_{i1}, f_{i2} ,…, f_{in}) стратегій, що вибрані автоматом, називається партією гри. Математичне очікування виграшу автомата A_α в цій партії визначається за формулою:

$$M_\alpha(f_{i1}, f_{i2}, \ldots, f_{iv}) = \frac{a_{i\alpha}}{m_{i\alpha}}, \tag{12.3}$$

де $a_{i\alpha}$ – потужність використовуваної автоматом A_α стратегії, а $m_{i\alpha}$ – число автоматів, що обрали цю стратегію.

Критерієм доцільності поведінки автомата A_α може бути середній виграш за T партій:

$$W_\alpha(t) = \frac{1}{T}\sum_{i=1}^{T} U_\alpha(t), \tag{12.4}$$

де $U_\alpha(t) = 1$ якщо автомат виграв і $U_\alpha(t) = -1$ якщо автомат програв в момент t.

Нехай стратегію f_i в партії $f(t)$ вибрало m_i^0 автоматів ($m_1^0 + m_2^0 + \cdots + m_k^0 = n$), і нехай для довільних i, j = 1, 2, …, k мають місце нерівності:

$$\frac{a_i}{m_i^0} \geq \frac{a_j}{m_j^0 + 1}. \tag{12.5}$$

В цьому випадку ні одному з автоматів не вигідно змінювати стратегію. Набір $m^0 = \left(m_1^0 + m_2^0 + \cdots + m_k^0 \right)$ називається точкою рівноваги (точка Неша).

З грою в розміщення B пов'язана ще одна гра – гра в розміщення з загальною касою B^c. Визначається ця гра так само як і гра в розміщення, а виграш автоматів задається формулою:

$$M_{\alpha^c}(f_{i1}, f_{i2}, \ldots, f_{iv}) = \frac{1}{v} \sum_{\alpha=1}^{v} M_\alpha(f_{i1}, f_{i2}, \ldots, f_{iv}), \qquad (12.6)$$

де $M_\alpha(f_{i1}, f_{i2}, \ldots, f_{iv})$ відомі з (12.3). Точкою Неша для неї являється набір $m_c^0 = (1, 1, \ldots, 1, 0, \ldots, 0)$, де $m_1^0 = m_2^0 = \cdots = m_n^0 = 1$, інші $m_1^0 = 0$. Партії визначені такими наборами, є партіями Мора, оскільки ціни цих партій максимальні.

Приклад 12.1. В іграх брали участь п'ять автоматів з пам'яттю n = 10 і сімома стратегіями. Потужності стратегій обирались наступними: a_1 = 0,9; a_2 = a_3 = … = a_7 = 0,33. При моделюванні виявилося, що в більшості партій першу стратегію обирали два автомати, інші шість стратегій майже в половині партій обираються одним автомат, в інших – вільні. В грі в розміщення з загальною касою в більшості партій автомати розприділялися по одному на перших n стратегіях (партії Мора).

Поведінка автоматів в грі з розміщенням володіє надійністю. Розглянемо гру в розміщення з загальною касою. В цій грі автомати так розприділяються по стратегіям, щоб отримати максимальний сумарний виграш. В кожний момент часу частка, що приноситься в сумарний виграш різними автоматами, різна (найбільший вклад вносить автомат, що обере стратегію f_1 і т.д.).

Будемо вважати, що автомати можуть виходити з ладу (закінчувати брати участь у грі). Тоді вихід з ладу автомата переводить гру $B^c(n)$ в гру $B^c(n-1)$, що відрізняється від $B^c(n)$ тільки числом n учасників. Так як для гри $B^c(n-1)$ зайнятими виявляються (якщо реалізується точка Мора) перші (n-1) стратегій, то через деякий час після виходу автомата з ладу незайнятою виявиться n-та стратегія незалежно від того, який саме автомат вийшов з ладу. Таким чином,

навіть при виході з ладу автомата, що обрав стратегію f_1 з максимальною потужністю a_1, сумарний виграш зменшиться лише на величину a_n.

Експеримент заключається в тому, що час від часу, але достатньо рідко, після установки середнього виграшу автоматів, що беруть участь у грі, виводить з ладу автомат, що обрав стратегію f_1. Коли ця стратегія скоро знову виявляється зайнятою, середній виграш кожного автомата, що бере участь у грі, збільшується: число використовуваних стратегій зменшується за рахунок найменш вигідної.

Приклад 12.2. У грі беруть участь п’ять автоматів з величиною пам’яті $n = 10$. Задано сім стратегій з потужностями: $a_1 = 0{,}9$; $a_2 = a_3 = \ldots = a_7 = 0{,}33$. Результат моделювання наведений в табл. 12.1, де W і W^c – середні виграші учасників в іграх B і B^c відповідно, коли число учасників рівне n.

Таблиця 12.1. Результат моделювання

v	5	4	3	2	1
W	0,38	0,39	0,41	0,45	0,9
W^c	0,44	0,47	0,52	0,61	0,9

З наведеного прикладу видно, що ріст середнього виграшу автомата при виході з ладу автоматів спостерігається і в грі B і в грі B^c. Зі зменшенням числа учасників гри при незмінній величині пам’яті середній виграш кожного автомата збільшується. Це відбувається, як уже зазначалося, за рахунок досягнення оптимального розподілу автоматів, що залишились в грі по стратегіям. При цьому оптимальний розподіл досягається більш точно для автоматів з більшою пам’яттю. При зменшенні числа учасників гра полегшується, і при незмінній величині пам’яті час існування оптимального розподілу збільшується.

Приклад 12.3. $k = 7$; $a_1 = 0{,}9$; $a_2 = 0{,}45$; $a_3 = 0{,}225$; $a_4 = 0{,}112$; $a_5 = 0{,}056$; $a_6 = 0{,}028$; $a_7 = 0{,}014$. В цій грі з загальною касою брали участь спочатку п’ять автоматів, потім чотири і три.

Таблиця 12.2. Результат машинного експерименту

v	Теоретичний виграш	*n*		
		7	10	12
5	0,35	0,28	0,29	0,3
4	0,42	0,32	0,37	0,4
3	0,52	0,43	0,47	0,52

Результат машинного експерименту наведено в табл. 12.2, по якій можна прослідити, як позначається на середньому виграші автомата зміна і його пам'яті, і числа *n* автоматів, що беруть участь у грі.

Опишемо гру багатьох автоматів, що допускає різноманітні змістовні інтерпретації. Розглянемо наступну «задачу про розподіл обчислювальної техніки». Нехай є деяка множина задач $C = c^i(a_1^i, \ldots, a_p^i)$, $i = 1, 2, \ldots, N$. Рішення задачі c^i заключається в виборі числа β із множини {1, 2, ..., *p*}, яке забезпечує виграш a_β^i. Цей вибір будемо називати вибором стратегії. Припустимо, що ці задачі виникають випадково і p_i – імовірність появи задачі c^i. Ми аж ніяк не припускаємо, що параметри a_β^i і p_i раніше задані.

Припустимо тепер, що є *M* обчислювальних машин R_k, k = 1, 2, ..., *M*. Кожна з них вмикається при появі індексу *i* (мітки) тої задачі, на яку вона налаштована. При вмиканні обчислювальна машина обирає одне з чисел $\beta \in \{1, 2, \ldots, p\}$, при цьому математичне очікування її виграшу рівне a_β^i.

У випадку, коли *M* = *N*, було б логічно налаштувати кожну з машин на свою мітку, тоді робота кожної з них звелася б до функціонування в стаціонарному середовищі *i*, якби обчислювальна машина належала до асимптотично-оптимальних послідовностей автоматів, можна було б гарантувати виграш:

$$W = \sum_{i=1}^{N} p_i \max_{\beta} a_\beta^i . \qquad (12.7)$$

Якщо ж число *M* обчислювальних машин менше загального числа *N* різних задач, то виникає питання про те, як налаштувати обчислювальну

машину на мітку, щоб забезпечити найбільший можливий виграш. Оскільки при цьому неминуче будуть з'являться задачі, на мітки яких не налаштована жодна з обчислювальних машин, будемо припускати, що в таких випадках вибір числа β відбувається випадково: числа 1, 2, …, p обираються з однаковою імовірністю 1/p. Тоді будемо говорити про вмикання машини випадкового вибору.

Іноді будемо вважати, що є обчислювальна машина, яка може бути налаштована і на мітку 0; при цьому вона вмикається при появі будь-якої задачі, на мітку якої не налаштована жодна з інших обчислювальних машин.

Будемо вважати, що виграш всієї системи S обчислювальних машин складається з виграшів всіх обчислювальних машин і виграшу машин випадкового вибору.

Наведемо найпростіший приклад. Нехай існує дві задачі $c^1 = (0,6; 0,9)$ і $c^2 = (0,45; -0,45)$, що виникають з імовірністю 0,5 кожна, і є одна обчислювальна машина. Якщо вона налаштована на мітку 1 і обирає β = 2, його виграш при появі задачі c^1 рівний 0,9; при появі задачі c^2 вмикається машина випадкового вибору і середній виграш системи буде дорівнювати $\frac{1}{2}[0,45 + (-0,45)] = 0$. Таким чином, враховуючи, що задачі c^1 і c^2 з'являються з імовірністю 0,5, середній виграш системи S становитиме 0,9·0,5 + 0·0,5 = 0,45. Аналогічно при налаштуванні обчислювальної машини на мітку 2 математичне очікування виграшу системи дорівнює [(0,6+0,9)/2]·0,5 + 0,45·0,5=0,6.

Уявімо тепер, що обчислювальна машина налаштована на мітку 0. В цьому випадку вона буде вмикатися при появі будь-якої задачі. Так як математичне очікування виграшу при виборі першої стратегії дорівнює 0,6·0,5 + 0,45·0,5 = 0,525, при виборі другої стратегії 0,9·0,5 – 0,45·0,5 = 0,225, обчислювальна машина при налаштуванні на мітку 0 буде обирати лише першу стратегію, забезпечуючи виграш 0,525. Таким чином, в цьому прикладі найбільш вигідним є налаштування обчислювальної машини на мітку 2.

Задачу про розподілення обчислювальної техніки можна розглядати як задачу про організацію колективної поведінки обчислювальних машин. Потрібно намагатися організувати їх взаємодію так, щоб доцільна поведінка окремих обчислювальних машин привела до оптимальної поведінки всієї системи, що виконує задачу. Для цього потрібно, щоб найбільш вигідним для кожної з обчислювальних машин був такий спосіб настройки, який забезпечив би найбільший сумарний виграш. Це досягається процедурою введення загальної каси. З іншого боку, в кожен такт виграш кожної обчислювальної машини залежить не тільки від мітки, обраної ним для вирішення завдання, а й від того, наскільки точно він це завдання вирішує. З цього випливає, що при будь-якій фіксованій настройці обчислювальних машин на мітки повинен пройти деякий час τ для того, щоб обчислювальні машини зуміли здійснити досить точний вибір стратегій. Тому характеристикою якості тієї чи іншої настройки на мітки може служити лише середній сумарний виграш обчислювальних машин за час τ, достатній для установки. Величина τ повинна обиратися з таким розрахунком, щоб за цей час можна було грубо оцінити доцільність тої чи іншої настройки на мітку. Звичайно, немає необхідності обирати τ достатнім для того, щоб вплив перехідного режиму зовсім припинив позначатися.

Ця особливість задачі про розподіл обчислювальної техніки породжує необхідність організації двох рівнів для її рішення. А саме в кожний такт t = 1, 2, ... , пред'являється та чи інша задача з множини $c^1, c^2, \ldots, c^N$, вмикається відповідна обчислювальна машина, відбувається вибір стратегії і оцінка цього вибору. В моменти τ, 2τ відбувається оцінка якості вибору міток і зміна налаштування на мітки.

В кожний момент t = 1, 2, … дія кожної обчислювальної машини характеризується двома числами (β, i) – стратегією, що обирається β $\beta = (1, 2, \ldots, p)$ і номером мітки i (i = 0, 1, … , N). При виборі числа β

обчислювальні машини діють незалежно один від одного, робота кожної з них є просто функціонування в випадковому стаціонарному середовищі.

В якості обчислювальних машин R_k *(k* = 1, 2, ..., *M)* будемо використовувати пари автоматів з лінійною тактикою $(A_k^{(1)}, A_k^{(2)})$ $k = 1, 2, \ldots, M$. Діями автомата $A_k^{(1)}$ є різні стратегії $\beta = (1, 2, \ldots, p)$; діями автомата $A_k^{(2)}$ є вибір мітки *i* = 0, 1, …, *N*. При цьому в інтервалі $t = (r-1)\tau + 1, \ldots, r\tau - 1$, стани автоматів $A_k^{(2)}$ і їх дія не змінюються.

Автомат $A_k^{(1)}$ $k = 1, 2, \ldots, M$ вмикається лиш при появі середовища з тією міткою *i*, на яку налаштований автомат $A_k^{(2)}$, і робить дію β, що відповідає його стану а даний момент. При цьому автомат виграє з імовірністю $q_\beta^i = \dfrac{1 + a_\beta^i}{2}$ і програє з імовірністю $p_\beta^i = \dfrac{1 - a_\beta^i}{2}$. Виграш і програш автомата $A_k^{(1)}$ змінюють його стан (у відповідності з його структурою).

При появі середовища *i*, на мітку якого не налаштований ні один з автоматів $A_k^{(2)}$ $k = 1, 2, \ldots, M$, або включається такий автомат $A_k^{(1)}$ (якщо $A_k^{(2)}$ має мітку 0), або пристрій випадкового вибору. Значить, в кожному такті роботи вмикається або один з автоматів $A_k^{(1)}$ $k = 1, 2, \ldots, M$, або пристрій випадкового вибору; інші автомати не змінюють своїх станів. Виграш або програш автоматів і пристроїв випадкового вибору сумуються за час τ, утворюючи сумарний виграш всієї системи *S* за цей час. Цей виграш, ділений на число тактів τ, визначає виграш *W* для будь-якого автомата в момент $A_k^{(2)}$ $k = 1, 2, \ldots, M$. Автомати $A_k^{(2)}$, що мають по *N* + 1 дії (0, 1, …, *M*) в моменти $r\tau$, $r = 1, 2, \ldots$ виграють з імовірністю *q* = (1 + *W*)/2 і програють з імовірністю *p* = (1 - *W*)/2, що призводить до зміни їх станів і, отже, дій. Вибір мітки реалізується таким способом, що кожну мітку обирає не більше ніж один автомат. Нагадаємо, що автомати $A_k^{(2)}$ також мають лінійну тактику.

Дамо опис роботи програми, що моделює на електронно-обчислювальній машині систему *S*. З імовірністю p_i, в кожний момент часу $t = 1, 2, \ldots$ на виході з'являється середовище $c^i = c^i(a_1^i, a_2^i, \ldots, a_p^i)$ де a_p^i – математичне очікування виграшу при виборі β-ї стратегії для i-го середовища ($i = 1, 2, \ldots, N$, $\beta = 1, 2, \ldots, p$). В грі беруть участь пари автоматів $(A_k^{(1)}, A_k^{(2)})$ $k = 1, 2, \ldots, M$ і пристрій випадкового вибору. Стан автомата $A_k^{(1)}$ $k = 1, 2, \ldots, M$ позначається через величину φ_j^β, де $\beta = 1, 2, \ldots, p$ – номер обраної автоматом стратегії, а j ($j = 1, 2, \ldots, n$) – глибина його стану. Стан автомата $A_k^{(2)}$ $k = 1, 2, \ldots, M$ позначається через φ_γ^i, де i ($i = 0, 1, \ldots, N$) – номер середовища (мітки), на яку автомат налаштований, а γ ($\gamma = 1, 2, \ldots, m$) – також глибина стану.

Нехай в момент t з'являється середовище c^i. Тоді шукається такий автомат $A_k^{(2)}$, який налаштований на мітку i. Якщо такий автомат є, то вмикається автомат $A_k^{(1)}$, якщо стан автомата в момент t був φ_j^β, то він обирає стратегію β. При цьому автомат $A_k^{(1)}$ виграє з імовірністю $q_\beta^i = \dfrac{1 + a_\beta^i}{2}$ і програє з імовірністю $p_\beta^i = \dfrac{1 - a_\beta^i}{2}$.

У випадку виграшу ($U_k(t) = 1$) автомат $A_k^{(1)}$ переходить з стану φ_j^β в стан φ_{j+1}^β, якщо $j \neq n$ і залишається в стані φ_n^β, якщо $j = n$. У випадку програшу ($U_k(t) = -1$) автомат $A_k^{(1)}$ переходить з стану φ_j^β в стан φ_{j-1}^β якщо $j \neq 1$ і переходить з імовірністю $1/p$ в один з станів $\varphi_j^{\beta'}$, $\beta' = 1, 2, \ldots, p$, якщо $j = 1$. Величини $U_k(t)$ сумуються за час $t = (r-1)\tau + 1, \ldots, r\tau - 1$, в лічильнику, що називається загальною касою. Якщо в момент t жоден з автоматів $A_k^{(2)}$ не має мітки i, але є автомат $A_k^{(2)}$ з нульовою міткою, рахунок проводиться так, якби у цього автомата була мітка i. Але можливо трапиться, що немає автомата ні з міткою i, ні з міткою 0. В цьому випадку працює пристрій випадкового вибору:

з імовірністю *1/p* обирається будь-яка зі стратегій $\beta = 1, 2, \ldots, p$, тоді математичне очікування виграшу рівне a_β^i. Через τ тактів обрахується виграш *W* кожного автомата $A_k^{(2)}$ системи *S* (загальний сумарний виграш системи за τ тактів, що поділений на число тактів автомата), а вміст загальної каси зникає. Після цього розігруються виграші для автоматів $A_k^{(2)}$ $k = 1, 2, \ldots, M$. Будь-який автомат $A_k^{(2)}$ виграє з імовірністю $q = (1 + W)/2$ і програє з імовірністю $p = 1 - q$. У випадку виграшу автомат $A_k^{(2)}$ переходить з стану φ_γ^i в стан $\varphi_{\gamma+1}^i$, якщо $\gamma \neq m$, і залишається в стані φ_m^i, якщо $\gamma = m$. У випадку програшу автомат $A_k^{(2)}$ переходить в стан $\varphi_{\gamma-1}^i$, якщо $\gamma \neq 1$. Якщо ж $\gamma = 1$, то автомат з рівною імовірністю переходить в стан φ_1^l, де *l* приймає будь-яке значення, крім зайнятих. Така процедура виключає випадковий вибір однакової мітки двома або більше автоматами.

Крім накоплення сумарного виграшу системи *S* за τ тактів, розрахуємо середній виграш кожної пари автоматів і пристроїв випадкового вибору за весь час рахунку *T*. Ці «особисті» виграші ніяк не впливають на останній рахунок, але по їх сумі (виграші системи *S*) судимо про доцільність поведінки автоматів.

Якщо автомати $(A_k^{(1)}, A_k^{(2)})$ належать до асимптотичної оптимальної послідовності автоматів, то можна допускати, що в межі (при $m, n \to \infty$) вони так обирають мітку і стратегії, щоб отримати максимально можливий виграш W_{max}^S.

Розглянемо декілька окремих випадків.

1) система *S* складається з однієї пари автоматів $(A^{(1)}, A^{(2)})$ нульова мітка відсутня, тобто $i = 1, 2, \ldots, N$, тоді

$$W_{\max}^S = \max_{i=1, 2, \ldots, N} \left\{ p_i \left[\max(a_1^i, a_2^i, \ldots, a_p^i) \right] + \sum_{r \neq i}^{N} \frac{a_1^\tau + \cdots + a_p^\tau}{p} p_r \right\}. \qquad (12.8)$$

2) система S складається з однієї пари автоматів $(A^{(1)}, A^{(2)})$ але $A^{(2)}$ може мати мітки $i = 0, 1, 2 \ldots, N$. Тоді в цьому випадку $W_{max}{}^S = \max W_i$ для $i = 0, 1, 2, \ldots, N$

$$W_0 = \max\left[\sum_{i=1}^{N} p_i a_1^i, \sum_{i=1}^{N} p_i a_2^i, \ldots, \sum_{i=1}^{N} p_i a_p^i\right], \tag{12.9}$$

$$W_i = p_i\left[\max(a_1^i, a_2^i, \ldots, a_p^i)\right] + \sum_{r\neq i}^{N} \frac{a_1^\tau + \cdots + a_p^\tau}{p} p_r\,. \tag{12.10}$$

3) система S складається з M пар автоматів $(A_k^{(1)}, A_k^{(2)})$ $k = 1, 2, \ldots, M$, автомати $A_k^{(2)}$ можуть мати лише мітки, відмінні від 0. В цьому випадку має місце формула:

$$W_{\max}^S = \max_{i_1, i_2, \ldots, i_M} W(i_1, i_2, \ldots, i_M),$$

де

$$W(i_1, i_2, \ldots, i_M) = \sum_{\delta=1}^{M} p_{i_\delta}\left[\max(a_1^{i_\delta}, a_2^{i_\delta}, \ldots, a_p^{i_\delta})\right] + \sum_{\substack{r\neq i_1 \\ \cdots \\ r\neq i_M}}^{N} \frac{a_1^r + a_2^r \cdots + a_p^r}{p} p_r\,. \tag{12.11}$$

4) система S складається з M пар автоматів $(A_k^{(1)}, A_k^{(2)})$ $k = 1, 2, \ldots, M$, автомати $A_k^{(2)}$ можуть мати мітки $i = 0, 1, 2 \ldots, N$, тоді виграш W системи при тій чи іншій настройці на мітки визначається формулою (12.11), якщо немає автомата $A_k^{(2)}$ з нульовою міткою, а якщо один з автоматів $A_k^{(2)}$ налаштований на мітку 0, то

$$W(i_\alpha = 0, i_1, \ldots, i_M) = \sum_{\substack{\delta=1 \\ \delta\neq\alpha}}^{M} p_{i_\delta}\left[\max(a_1^{i_\delta}, a_2^{i_\delta}, \ldots, a_p^{i_\delta})\right] + \left(1 - \sum_{\substack{\delta=1 \\ \delta\neq\alpha}}^{M} p_{i_\delta}\right) \max_{\substack{r\neq i_1 \\ r\neq i_M}} \left[\sum_{r=1}^{N} p_r a_1^r, \ldots, \sum_{r=1}^{N} p_r a_p^r\right],$$

$$W_{\max}^S = \max_{i_1, i_2, \ldots, i_M}\left[W(i_{\alpha=0}, i_1, \ldots, i_M), W(i_1, i_2, \ldots, i_M)\right]. \tag{12.12}$$

Нижче наведені результати моделювання декількох прикладів задач про розподіл обчислювальної техніки.

Приклад 12.4. $c^1 = (0,9; 0)$, $c^2 = (-0,9; 0,9)$.

Середовища з'являються з ймовірністю 0,5, в грі бере участь одна пара автоматів $(A^{(1)}, A^{(2)})$ з пам'яттю $n = m = 5$. Налаштування на мітки відбувається кожні 20 тактів. Виграш W_{max}^S розраховується за формулами (12.9), (12.10) і виходить таким: $W_0 = 0,45$, $W_1 = 0,45$, $W_2 = 0,45$. Результати моделювання для випадку $i = 1; 2$ і для $i = 0; 1; 2$ приведені в табл. 12.3, 12.4.

Таблиця 12.3. Результати моделювання для випадку $i = 1; 2$

Число тактів	200	1000	2000	3000	4000
Виграш W^s	0,44	0,67	0,69	0,66	0,67
Загальне число тактів на мітці 1	200	540	540	540	540
Загальне число тактів на мітці 2	0	460	1460	2460	3460

Таблиця 12.4. Результати моделювання для випадку $i = 0; 1; 2$

Число тактів	200	1000	2000	3000	4000	5000	6000
Виграш W^s	0,44	0,45	0,66	0,67	0,67	0,69	0,66
Загальне число тактів на мітці 0	0	490	1200	1200	1200	1200	1200
Загальне число тактів на мітці 1	200	510	560	560	560	560	560
Загальне число тактів на мітці 2	0	0	240	1240	2240	3240	4240

З приведених результатів видно, що автомат $A^{(2)}$ в обох варіантах через достатнє число тактів обирає мітку 2, середній виграш системи в цьому випадку 0,67.

Приклад 12.5. З імовірністю 0,5 на вході з'являються середовища $c^1 = (0,6; 0,9)$, $c^2 = (0,45; -0,45)$. В грі бере участь одна пара автоматів $A^{(1)}$, $A^{(2)}$ пам'ять n автомата $A^{(1)}$ рівна п'яти, пам'ять $A^{(2)}$ – семи. Налаштування на мітки перевіряється кожні 20 тактів ($\tau = 20$). Мітка може бути будь-якою від 0 до 2.

Із формул (12.9), (12.10) отримуємо наступні теоретичні виграші: $W_0 = 0,525$, $W_1 = 0,45$, $W_2 = 0,6$. Автомат $A^{(2)}$ обирає мітку 2. Виграш системи W^S в середньому дорівнює 0,6.

Приклад 12.6. З однаковою ймовірністю 0,2 з'являється одне з середовищ: $c^1 = (0,2; 0,2)$, $c^2 = (-0,9; 0,9)$, $c^3 = (-0,5; 0,5)$, $c^4 = (-0,3; 0,3)$, $c^5 = (-$

0,3; 0,3). Неважко побачити, що для першого середовища вигідна стратегія $\beta = 1$, а для інших $\beta = 2$.

Система S складається з двох пар автоматів $(A_k^{(1)}, A_k^{(2)})$ $k = 1, 2, \ldots, M$ і пристрою випадкового вибору. В цьому випадку легко вичислити середній виграш системи при тому чи іншому налаштуванні на мітки (див. табл. 12.5). В цій таблиці парою чисел *i, j* позначені налаштування однієї обчислювальної машини на мітку *i*, а іншої – на мітку *j*.

Таблиця 12.5. Середній виграш системи при тому чи іншому налаштуванні на мітки

Налаштування	0,1	0,2	0,3	0,4	0,5	1,2	1,3	1,4	1,5	2,3	2,4	2,5	3,4	3,5	3,6
Виграш	0,44	0,28	0,28	0,28	0,28	0,22	0,14	0,1	0,1	0,24	0,2	0,2	0,12	0,12	0,08

Таблиця 12.6. Результати моделювання

		Розподіл автомата $A_k^{(2)}$ за мітками											
		$A_1^{(2)}$						$A_2^{(2)}$					
m	W^s	0	1	2	3	4	5	0	1	2	3	4	5
5	0,2	34	18	13	12	17	6	17	31	24	10	10	8
10	0,42	0	85	4	0	6	5	93	0	0	0	0	7

Із таблиці 12.6 видно, що найбільш вигідною є настройка одного автомата $A_k^{(2)}$ на мітку 0, а іншого – на мітку 1, виграш системи при цьому дорівнює 0,44. середній виграш системи при налаштуванні на будь-яку мітку рівний 0,18; тому, якщо виграш системи рівний 0,42, це означає, що комбінація (0, 1) зустрічається приблизно в 12 разів частіше, ніж всі інші.

Пам’ять всіх автоматів $A_k^{(1)}$ $k=1,2,\ldots,M$ рівна 5. Були прораховані варіанти з різною величиною m пам’яті автоматів $A_k^{(2)}$ $k=1,2,\ldots,M$ і з різним інтервалом τ. В табл. 12.7 приведені середні виграші і розподіли автоматів $A_k^{(2)}$ по середовищам до моменту 10000 інтервалів τ.

Таблиця 12.7. Середні виграші і розподіли автоматів $A_k^{(2)}$ по середовищам

Число інтервалів τ	$A_1^{(2)}$							$A_2^{(2)}$					
	W^s	0	1	2	3	4	5	0	1	2	3	4	5
1000	0,14	0	16	40	0	44	0	28	0	0	0	0	72
2000	0,30	0	24	20	0	32	24	24	0	0	0	0	36
4000	0,42	0	62	10	0	16	12	82	0	0	0	0	18
8000	0,42	0	81	5	0	8	6	91	0	0	0	0	9
12000	0,42	0	87,4	3,3	0	5,2	4,1	94	0	0	0	0	6

З приведеної таблиці видно, що при m = 5 виходить розмита картина розподілу автоматів по середовищам, при m = 10 чітко виділяється комбінація (0, 1).

Наскільки швидко така комбінація обирається, можна прослідкувати по табл. 12.7 (m = 10, $\tau = 40$).

Таким чином, практично після інтервалу $2000\,\tau$ автомати обирають лише комбінацію міток (0,1).

Перелік посилань

1. Neal S. Widmer, Gregory L. Moss, Ronald J. Tocci. Digital systems. Principles and applications. Pearson, 2016. – 1004 p.

2. Robert Boylestad. Introductory Circuit Analysis. Pearson, 2016. – 1224 p.

Лекція №13

Теорія ігор

Теорія ігор – сукупність методів математичного аналізу та оцінки правил поведінки учасників конфліктної ситуації [1]. Кожна конфліктна ситуація передбачає взаємодію двох чи більше учасників цієї ситуації (гравців) для досягнення кожним з них своєї мети. Протилежні інтереси учасників гри створюють конфліктну ситуацію. Рівень досягнення мети гравцем, тобто кінцевий результат його участі у грі, залежить як від непередбачуваних ситуацій, так і від поведінки інших учасників гри. Кожен з учасників гри, зрозуміло, хоче отримати для себе найбільшу вигоду. Результат гри, яку частково контролює кожен з учасників, певною мірою залежить від випадку, а найголовніше – від кмітливості і майстерності гравців. Конфліктні ситуації, до яких можна застосувати теорію ігор, існують як у класичних іграх (шахах, покері та інших), так і в економіці, біології, геології, політиці, військовій справі.

Теорію ігор як математичну дисципліну започатковано одночасно з теорією ймовірностей у середині XVII ст., однак її розвитку практично не відбувалося. Поштовхом до нових розробок у цій галузі була книга Дж. Неймана «До теорії стратегічних ігор» (1928) і книга 1944 Дж. Неймана та О. Моргенштерна «Теорія ігор і економічна поведінка» (1944). Теорія ігор з самого початку була спрямована на розв’язок задач щодо прийняття рішень у конкурентній економіці. У зазначених працях містились переважно економічні приклади, економічні конфліктні ситуації.

З часу другої світової війни теорія ігор серйозно зацікавила військових, які наявний математичний апарат застосували для аналізу військових стратегій. Зараз істотно розширилась сфера застосування теорії ігор, до неї тепер належать і соціальні науки.

Теорію ігор використовують з метою визначення вибору стратегій у конфліктних ситуаціях. Сьогодні теорія ігор все ще залишається чистою

теорією, сфера використання теорії ігор обмежена, оскільки за умови, що суперників більше двох і правила їхньої поведінки ускладнено, труднощі аналізу різко зростають.

Хоча теорія ігор ще не має практичного значення при вирішенні реальних завдань державного управління, однак уявлення, які вона розвиває, дають змогу керівництву краще розуміти конкурентну обстановку.

Процес аналізу, який спонукає керівника розглядати всі можливі альтернативи як своїх дій, так і дій конкурентів, є надзвичайно корисним.

Ігрове імітування можна здійснювати у формі спеціальних ігор або ігор людини і машини. Операційна гра – це спільна робота об'єднаних у групи експертів, які розподіляють між собою ролі, що відповідають діям учасників реальних процесів, та приймають рішення у відповідь на конкретні дії супротивників чи союзників за грою.

Головна мета гри – вироблення практичних рекомендацій щодо дій у різних ситуаціях під час навчання експертів.

Людино-машинні ігри полягають у створенні структури, в якій органічно взаємодіють імітаційна модель певного процесу на ЕОМ та учасники гри.

Загалом операційні та людино-машинні ігри сьогодні широко застосовують з метою моделювання економічних процесів у випадках, коли процес недостатньо вивчений для того, щоб цілковито сформувати його щодо реалізації на ЕОМ.

Теорія ігор користується методами теорії ймовірностей, функціонального аналізу, теорії диференціальних та інтегральних рівнянь [1]. Вона використовує ті самі методи, що й усі галузі математики: формулювання принципу оптимальності аксіоматичне, пошук розв'язків здійснюють на базі аналітичних розрахунків. Завдяки експериментальним методам дослідження гри може відбуватись на основі багаторазового відтворення гри людьми (ділові ігри) чи з допомогою цифрового моделювання на ЕОМ (ігри автоматів).

Відшукання розв’язків ігор можна здійснити поки що лише у досить вузьких окремих класах ігор. В окремих випадках це можна описати за допомогою формул, а здебільшого його представляють 4 у вигляді алгоритмів.

Кількість учасників гри, або кількість груп інтересів у грі є одним з базових понять теорії ігор. Ігри двох гравців займають центральне місце в теорії ігор. Їх ще називають іграми двох гравців з нульовою сумою, в яких один гравець виграє стільки, скільки програє другий. До таких ігор шляхом відповідних перетворень можна звести ігри двох гравців з постійною сумою, у яких конкуренти намагаються зі зазначеної суми забрати якомога більшу частку. Усі інші ігри кваліфікують як такі, що мають понад два учасники.

Учасники гри можуть вступати у коаліції дій, множину яких позначають через R. Кожну дію коаліції у грі називають стратегією. Це поняття є центральним у теорії ігор. Стратегією гравця є повний перелік усіх дій, які цей гравець застосовує у кожному випадку, що виникає під час гри унаслідок ходу конкурента, свого ходу чи цілком випадково. Виходячи з конкретних обставин, що виникають, стратегія визначає гравцеві хід, який він повинен зробити у даному випадку.

Функція виграшу H_k, яка фактично є правилом, визначення очікуваного виграшу першого гравця у випадку вибору ним будь-якої стратегії зі своєї множини допустимої стратегії, а інший гравець вибере будь-яку стратегію зі своєї множини допустимих стратегій. Для коаліції k множину всіх стратегій позначають через S_k.

Вибір кожним учасником гри чи коаліцією дій своєї стратегії створює ситуації, множину яких позначають через S. Зацікавленість у результатах гри стимулює учасників до формування коаліції інтересів, множину яких позначають через R_i. Перевагу ситуації $S’$ над ситуацією $S’’$ для коаліції k записують як $S’ > S’’$.

У нашому випадку справедливе $H_k^{(S')} > H_k^{(S'')}$. Перелік усіх коаліцій дій, множина їхніх стратегій і ситуацій, коаліції інтересів визначають гру. Очевидно, що у будь-якої гри є понад дві коаліції інтересів.

Класифікація ігор

Якщо існує сідлова точка, то дві стратегії, зумовлені нею, називають оптимальними. Можлива наявність обмежень на допустимі змішані стратегії (гра з обмеженнями). Нескінченні ігри використовують нескінченні множини чистих стратегій [1].

Багатокрокові ігри задаються на повному наборі інтервалів, кожний з яких є кроком гри. У цьому випадку обоє гравців володіють повною інформацією.

Стохастичні ігри мають ту особливість, що на кожному кроці існує позитивна імовірність зупинки і, незалежно від вибраної стратегії, імовірність того, що через n кроків гра буде продовжуватися, не більше S_n (де S_n – наперед задана, менша від одиниці величина).

У рекурсивних іграх імовірність закінчення гри на фіксованому кроці може бути нульовою. Багатокрокові ігри з неперервним часом, у яких перехід від одного стану до іншого описується диференціальними рівняннями, називають диференціальними іграми.

В іграх двох гравців з нульовою сумою інтереси гравців не обов’язково є протилежними, що може дати вигоду обом гравцям за умови узгодженості їхніх дій. Якщо у таких іграх допускають обмін інформацією, спільні стратегії, другорядні платежі, угоди – такі ігри називають кооперативними. Якщо такі взаємодії категорично заборонені – це некооперативні ігри.

Якщо у грі є лише одна коаліція дії, вважають, що множина ситуацій збігається з множиною стратегій і такі ігри називаються нестратегічними.

Якщо множини коаліцій дій і коаліцій інтересів збігаються, то обидва види коаліцій називають гравцями. Вироблення рішення в теорії ігор полягає у виборі коаліції дії чи якоїсь своєї стратегії гравцем, тобто вибору елемента з

певної множини. У випадку динамічних ігор вибір стратегії відбувається на певному проміжку часу.

В залежності від числа стратегій, по числу гравців, в залежності від взаємовідносин гравців і т.п. В *іграх з нульовою сумою* одні гравці виграють за рахунок інших, тобто сумарний виграш всіх гравців дорівнює нулю. *Парні ігри з нульовою сумою* називаються антагоністичними. Кінцеві антагоністичні ігри називаються *матричними іграми*.

Матричні ігри двох осіб

У випадку скінченності множини S_1 і S_2 (гра двох гравців) нормальну форму гри можна записати у вигляді матриці $A = (a_{ij})$, у якій елементи рядків представляють стратегії першого гравця, а елементи стовпчиків представляють стратегії другого гравця. Елементи a_{ij} матриці A є платою другого гравця першому у випадку вибору першим гравцем i-го рядка, а другим гравцем – j-го стовпчика. Якщо відомі всі значення a_{ij}, то можна записати платіжну матрицю гри (табличне представлення функції виграшу, табл. 13.1).

Таблиця 13.1. Табличне представлення функції виграшу

	B_1	B_2	…	B_j	…	B_n	α
A_1	a_{11}	a_{12}	…	a_{1j}	…	a_{1n}	α_1
A_2	a_{21}	a_{22}	…	a_{2j}	…	a_{2n}	α_2
…	…	…	…	…	…	…	…
A_i	a_{i1}	a_{i2}	…	a_{ij}	…	a_{in}	α_i
…	…	…	…	…	…	…	…
A_m	a_{m1}	a_{m2}	…	a_{mj}	…	a_{mn}	α_m
β	β_1	β_2	…	β_j	…	β_n	

Очевидно, перший гравець вибирає рядок з метою максимізувати плату (свій виграш), а другий гравець вибирає стовпчик з метою мінімізувати плату (свій програш). Таку гру називають *матричною*.

Тобто перший гравець намагатиметься спочатку вибрати мінімальне значення виграшу по кожній стратегії:

$$\alpha_i = \min_j a_{ij},\ i = \overline{1, m}.$$

А потім серед цим виграшів шукатиме той, який забезпечує максимум:

$$\alpha = \max_i \alpha_i = \max_i \min_j a_{ij}.$$

Величину α називають *нижньою чистою ціною гри* або *максіміном*. Їй відповідає максимінна стратегія, дотримання якої першим гравцем при будь-яких стратегіях другого гравця, гарантує для нього виграш, не менший, ніж α. Тобто це гарантований виграш першого гравця за умови будь-яких стратегій другого гравця.

У свою чергу, другий учасник максимізує свій програш по кожній стратегії:

$$\beta_j = \max_i a_{ij}, \ j = \overline{1, n}.$$

А потім серед усіх стратегій вибирає ту, при якій програш найменший:

$$\beta = \max_j \beta_j = \min_j \max_i a_{ij}.$$

Величину β називають *верхньою чистою ціною гри* (*мінімаксом*). Їй відповідає мінімаксна стратегія другого гравця, дотримання якої за умови будь-якої стратегії першого гравця, забезпечує йому програш, не більший, ніж β. Тобто це гарантований програш другого гравця за умови будь-яких стратегій першого гравця.

Оптимальним розв'язком цієї гри є *сідлова точка,* тобто елемент a_{kl} матриці *A*, який задовольняє умову $a_{il} \le a_{kl} \le a_{kj}$ для будь-яких *i* та *j*.

Якщо $\beta = \alpha$, тобто верхня ціна гри дорівнює нижній ціні гри, то відповідні чисті стратегії називають оптимальними, а про гру кажуть, що вона має сідлову точку.

У більшості матричних ігор відсутні сідлові точки, тобто не виконується умова:

$$\max_i \min_j a_{ij} \le \min_j \max_i a_{ij}.$$

У таких випадках використовують змішані стратегії, які є схемою випадкового вибору чистих стратегій.

Змішаною стратегією S_A першого гравця називають застосування чистих стратегій $A_1, A_2, \ldots, A_m$ з ймовірностями $p_1, p_2, \ldots, p_m$, причому:

$$\sum_{i=1}^{n} p_i = 1,\ p_i \geq 0,\ i = \overline{1, m}.$$

Змішані стратегії гравця A запишемо у вигляді матриці:

$$S_A = \begin{pmatrix} A_1 & A_2 & \cdots & A_i & \cdots & A_m \\ p_1 & p_2 & \cdots & p_i & \cdots & p_m \end{pmatrix}.$$

Або у вигляді рядка $S_A = (p_1, p_2, \ldots, p_m)$.

Аналогічно змішані стратегії гравця B позначимо

$$S_B = \begin{pmatrix} B_1 & B_2 & \cdots & B_i & \cdots & B_m \\ q_1 & q_2 & \cdots & q_i & \cdots & q_m \end{pmatrix}.$$

Або у вигляді рядка $S_B = (q_1, q_2, \ldots, q_m)$, де $\sum_{j=1}^{n} p_j = 1,\ q_j \geq 0,\ j = \overline{1, n}$.

Гравець A вибирає стратегію так, щоб максимізувати найменший очікуваний виграш по стовпцях платіжної матриці, а гравець B вибирає стратегію так, щоб мінімізувати найбільший очікуваний виграш по стовпцях. Отже, гравець A вибирає стратегію p_i:

$$\max_{p_i} \left\{ \min \left(\sum_{i=1}^{m} a_{i1} \cdot p_i, \sum_{i=1}^{m} a_{i2} \cdot p_i, \ldots, \sum_{i=1}^{m} a_{in} \cdot p_i \right) \right\}.$$

Гравець B вибирає стратегію q_i:

$$\min_{q_j} \left\{ \max \left(\sum_{j=1}^{n} a_{j1} \cdot q_j, \sum_{j=1}^{n} a_{j2} \cdot q_j, \ldots, \sum_{j=1}^{n} a_{jm} \cdot q_j \right) \right\}.$$

Якщо стратегії оптимальні, то виконується рівність між максимінним очікуваним виграшем та мінімаксним очікуваним програшем.

Теорема (про активні стратегії). Якщо один гравець дотримується своєї оптимальної змішаної стратегії, то виграш залишається незмінним і рівним ціні гри, якщо інший гравець не виходить за межі своїх активних стратегій.

Теорема фон Неймана (основна теорема матричних ігор). Будь-яка матрична гра має принаймні одне рішення в змішаних стратегіях – дві оптимальні стратегії і відповідну їм ціну:

$$\left\langle \bar{x}^*, \bar{y}^*, v = f\left(\bar{x}^*, \bar{y}^*\right)\right\rangle. \qquad (13.1)$$

Графічний метод розв'язання матричної гри

Гра 2 × *N* задається платіжною матрицею, що подана у таблиці 13.2.

Таблиця 13.2. Платіжна матриця

	B_1	B_2	…	B_j	…	B_n
A_1	a_{11}	a_{12}	…	a_{1j}	…	a_{1n}
A_2	a_{21}	a_{22}	…	a_{2j}	…	a_{2n}

Гравець *A* має дві чисті стратегії, а гравець *B* – *n* чистих стратегій.

Алгоритм геометричного (графічного) знаходження оптимальних стратегій гравця *A* та ціни гри *v*:

1) Відміряємо горизонтальний відрізок [0, 1]. Через кінці відрізка проводимо два перпендикуляри. На лівому перпендикулярі як на числовій осі відкладаємо елементи першого рядка платіжної матриці. На правому перпендикулярі аналогічно відкладаємо всі елементи другого рядка платіжної матриці. Масштаб на лівому та правому перпендикулярах однаковий, хоча може не співпадати із масштабом горизонтального одиничного відрізка.

2) З'єднуємо відрізками кожну пару точок, що відповідають стратегії гравця *B*, тобто в платіжній матриці відображені у стовпцях.

3) Виділяємо нижню частину сімейства відрізків, яка в загальному випадку буде мати вигляд ламаної, а у частковому випадку може бути прямою.

4) Знаходимо на цій ламаній максимальну (найвищу) точку (точки). Абсциса цієї точки є ймовірністю p^0 вибору гравцем *A* чистої стратегії A_2 в оптимальній змішаній стратегії $P^0 = (1 - p^0, p^0)$. Ордината точки є ціною гри *v*. Приклад зображено на рис. 13.1.

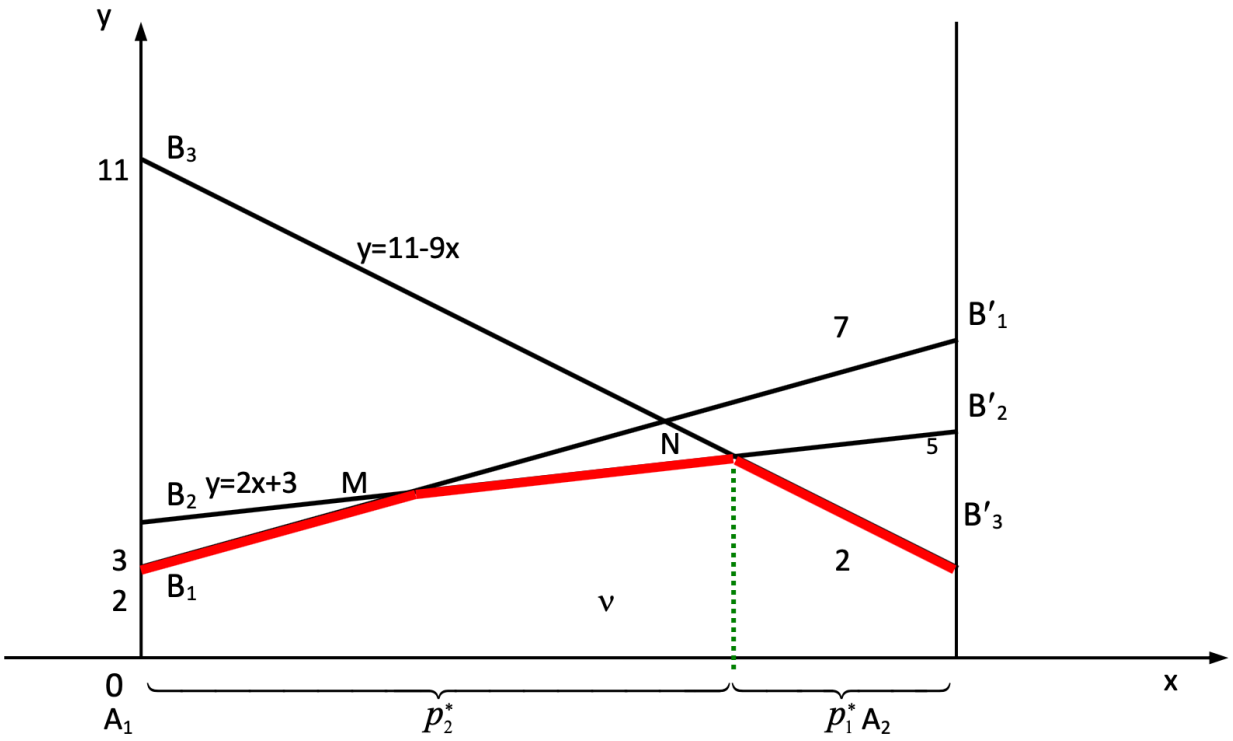


Рис. 13.1. Знаходження оптимальної стратегії гравця A

Гра N × 2 задається платіжною матрицею, що подана у таблиці 13.3.

У цьому випадку гравець A має m чистих стратегій, а гравець B – дві чисті стратегії.

Таблиця 13.3. Платіжна матриця

	B_1	B_2
A_1	a_{11}	a_{12}
A_2	a_{21}	a_{22}
…	…	…
A_i	a_{i1}	a_{i2}
…	…	…
A_m	a_{m1}	a_{m2}

Алгоритм геометричного (графічного) знаходження оптимальних стратегій гравця B та ціни гри v:

1) Відміряємо горизонтальний відрізок [0,1]. Через кінці відрізка проводимо два перпендикуляри. На лівому перпендикулярі як на числовій осі відкладаємо елементи першого стовпця платіжної матриці. На правому

перпендикулярі аналогічно відкладаємо всі елементи другого стовпця платіжної матриці. Масштаб по обох перпендикулярах, знову ж таки, однаковий.

2) З'єднуємо відрізками кожну пару точок, що відповідають стратегії гравця *A*, тобто в платіжній матриці відображені у рядках. Отримуємо *m* відрізків.

3) Виділяємо верхню частину сімейства відрізків, яка в загальному випадку буде мати вигляд ламаної, а у частковому випадку може бути прямою.

4) Знаходимо на цій ламаній мінімальну (найнижчу) точку (точки). Абсциса мінімальної точки вказує ймовірністю q^0 вибору гравцем *B* чистої стратегії B_2 в оптимальній змішаній стратегії $Q^0 = (1 - q^0, q^0)$. Ордината мінімальної точки є ціною гри *v*. Приклад зображено на рис. 13.2.

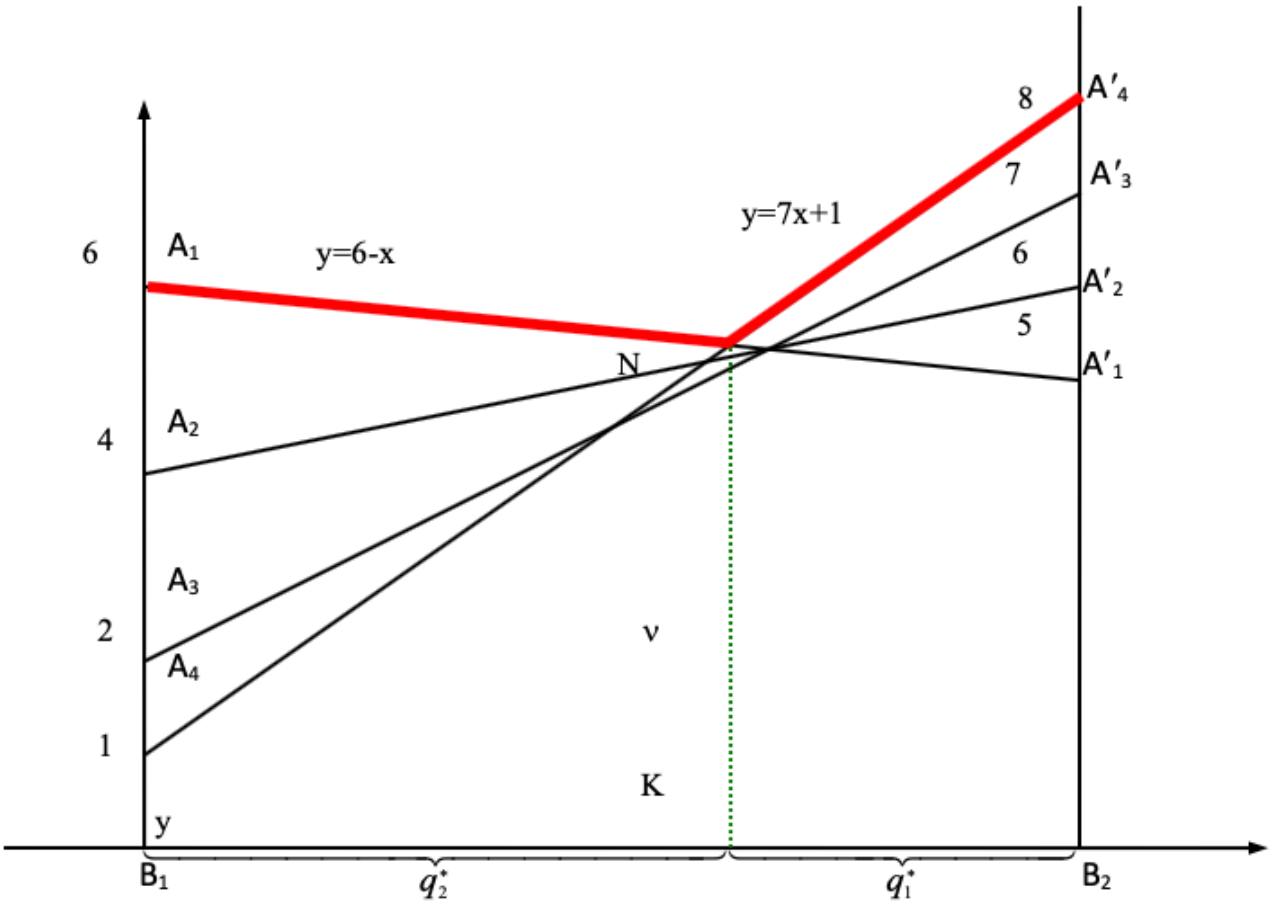


Рис. 13.2. Знаходження оптимальної стратегії гравця *B*

Розв’язання матричної гри шляхом зведення до задачі лінійного програмування

Розглянемо матричну гру (m×n), яка задана платіжною матрицею A з елементами $\|a_{ij}\|_{m\times n}$, $a_{ij} \geq 0$ [2]. Оптимальна змішана стратегія гравця A – $p = (p_1, p_2, \ldots, p_m)$, а гравця B – $q = (q_1, q_2, \ldots, q_n)$. Відомо, що для оптимальної стратегії першого гравця та ціни гри v виконується нерівність $\sum_{i=1}^{m} a_{ij} p_i \geq v$, $j = \overline{1, n}$. Можемо припустити, що ціни гри v невід’ємна, чого завжди можна досягти, збільшивши ціну гри на деяке стале число. Поділивши обидві частини останньої нерівності на ціну гри, отримаємо:

$$\sum_{i=1}^{m} a_{ij} \frac{p_i}{v} \geq 1, \; j = \overline{1, n}.$$

Нехай $\frac{p_i}{v} = y_i$, тоді

$$\sum_{i=1}^{m} a_{ij} y_i \geq 1, \; j = \overline{1, n}, \; y_i \geq 0, \; i = \overline{1, m}.$$

Оскільки для змішаних стратегій виконувалась умова $\sum_{i=1}^{m} p_i = 1$, то тепер умова набуде вигляду

$$\sum_{i=1}^{m} y_i = \frac{1}{v}.$$

Оскільки перший гравець прагне максимізувати свій виграш, то він повинен забезпечити мінімум величини $\frac{1}{v}$. З урахуванням цього, визначення оптимальної стратегії першого гравця зводиться до розв’язування оптимізаційної задачі лінійного програмування:

$$\min F; \; F = \sum_{i=1}^{m} y_i,$$

$$\sum_{i=1}^{m} a_{ij} y_i \geq 1,\ j = \overline{1,n},\ y_i \geq 0,\ i = \overline{1,m}.$$

За аналогією визначення оптимальної стратегії другого гравця зводиться до знаходження розв'язку задачі лінійного програмування:

$$\max G;\ G = \sum_{j=1}^{n} x_j,$$

$$\sum_{j=1}^{n} a_{ij} x_j \leq 1,\ i = \overline{1,m},\ x_j \geq 0,\ j = \overline{1,n}.$$

Таким чином, щоб знайти розв'язок гри, що задана матрицею *A*, необхідно розв'язати пару двоїстих задач.

Оптимальна змішана стратегія гравця *A*: $p_i^* = v \cdot y_i$, ($i = \overline{1,m}$).

Оптимальна змішана стратегія гравця *B*: $q_j^* = v \cdot x_j$, ($j = \overline{1,n}$).

Етапи розв'язування матричної гри шляхом зведення до задачі лінійного програмування:

1) записати пару двоїстих задач лінійного програмування, еквівалентних заданій матричній грі;

2) визначити оптимальні плани пари двоїстих задач;

3) використовуючи співвідношення між планами пари двоїстих задач та оптимальними стратегіями і ціною гри, знайти її оптимальний розв'язок.

Статичні ігри з повною інформацією

Під *статичною* розуміють таку гру, в якій всі її учасники приймають рішення, не знаючи, які саме рішення приймають інші [2].

Під *іграми з повною інформацією* розуміють гри, в яких кожен з гравців точно знає характеристики інших гравців.

Гра в нормальній формі задається наступною сукупністю об'єктів $G = \{S_i, u_i, i \in N\}$, де $N = \{1,2,...,n\}$ – множина гравців, S_i – множина стратегій *i*-го гравця, $u_i : \prod_{i \in N} S_i \to R$ – функція виграшу гравця *i*.

Профіль стратегій всіх гравців: $s = (s_1,...,s_n) \in S = \prod_{i \in N} S_i$.

Профіль стратегій всіх гравців, окрім i – го (тобто інших гравців): $s_{-i} = \left(s_j, j \neq i\right) \in S_{-i} = \prod_{j \neq i} S_j$. При цьому очевидно, що $s = \left(s_i, s_{-i}\right) = \left(s_1, ..., s_n\right)$.

Для гри в нормальній формі $G = \left\{S_i, u_i, i \in N\right\}$ стратегія s_i' строго домінує стратегію s_i'', якщо $u_i\left(s_i', s_{-i}\right) > u_i\left(s_i'', s_{-i}\right), \quad \forall s_{-i} \in S_{-i}$.

Рівновагою в (строго) домінуючих стратегіях називається профіль (строго) домінуючих стратегій, якщо такі стратегії існують для кожного гравця. Очевидним шляхом використання домінування є послідовне видалення строго домінуючих стратегій (СДС). Наприклад, в грі з матрицею процес послідовного видалення СДС не завжди призводить до отримання результату. Тут ні у одного з гравців немає жодної домінуючої стратегії. Тут будь-яка стратегія є найкращою при деякому виборі іншого гравця. У цьому випадку підхід, заснований на домінуванні, безсилий, тому потрібно більш універсальний підхід.

Рівновагою Неша в грі в нормальній формі $G = \left\{S_i, u_i, i \in N\right\}$ називається такий профіль стратегій s^* всіх гравців, що $u_i\left(s^*\right) \geq u_i\left(s_i, s_{-i}^*\right), \quad \forall i \in N, \quad \forall s_i \in S_i$. Сенс рівноваги Неша в тому, що жодному з гравців невигідно відхилятися від своєї рівноважної стратегії поодинці.

Перелік посилань

1. Neal S. Widmer, Gregory L. Moss, Ronald J. Tocci. Digital systems. Principles and applications. Pearson, 2016. – 1004 p.

2. Robert Boylestad. Introductory Circuit Analysis. Pearson, 2016. – 1224 p.

Лекція №14

Типи лічильників та особливості їх роботи

Лічильники імпульсів – це своєрідні скінченні автомати, які можуть проектуватись у відповідності до деяких правил і необхідного алгоритму функціонування [1].

У загальному плані *лічильниками* називаються цифрові пристрої (автомати), призначені для підрахунку і фіксації кількості імпульсів, що подаються на їх інформаційні входи або синхровходи. Назва *"лічильники"* використовується до будь-яких послідовнісних цифрових пристроїв із замкнутим циклом діаграми станів.

Як пристрої цифрової схемотехніки, лічильники характеризуються наступними основними параметрами:

1. статичний параметр – *коефіцієнт перерахунку* (*модуль перерахунку*) *M* – характеризує максимальну кількість імпульсів, яка може бути подана на лічильник, щоб привести його до початкового стану.

2. динамічні параметри лічильників характеризують їх швидкодію,

2.1. *час установлення вихідного коду* ($t_{уст}$) – це інтервал часу з моменту подачі вхідного імпульсу до моменту встановлення коду на виходах лічильника;

2.2. *дозволяюча спроможність лічильника* ($t_{сд}$) визначається як мінімальний інтервал часу між двома вхідними імпульсами. Величина

$$f_{max} = t_{сд}^{-1} \tag{14.1}$$

називається *максимальною частотою роботи лічильника*.

Оскільки скінченні автомати можуть бути синхронними і асинхронними, то і, відповідно, ті групи автоматів, які спеціально призначені для підрахунку кіль кості імпульсів, можуть також бути синхронними і асинхронними.

У синхронних лічильниках синхросигнал на всі тригери подається одночасно, а зміна стану тригера відбувається лише тоді, коли на інформаційних входах будь-якого тригера підготовлені відповідні дані. В

асинхронних лічильниках, які часто називають *послідовними*, вхідна послідовність імпульсів подається лише на перший тригер, а решта тригерів спрацьовує в залежності від зміни стану попереднього.

Асинхронні лічильники

Прикладом асинхронних (послідовних) лічильників є схема, що приведена на рис. 14.1, ***а***.

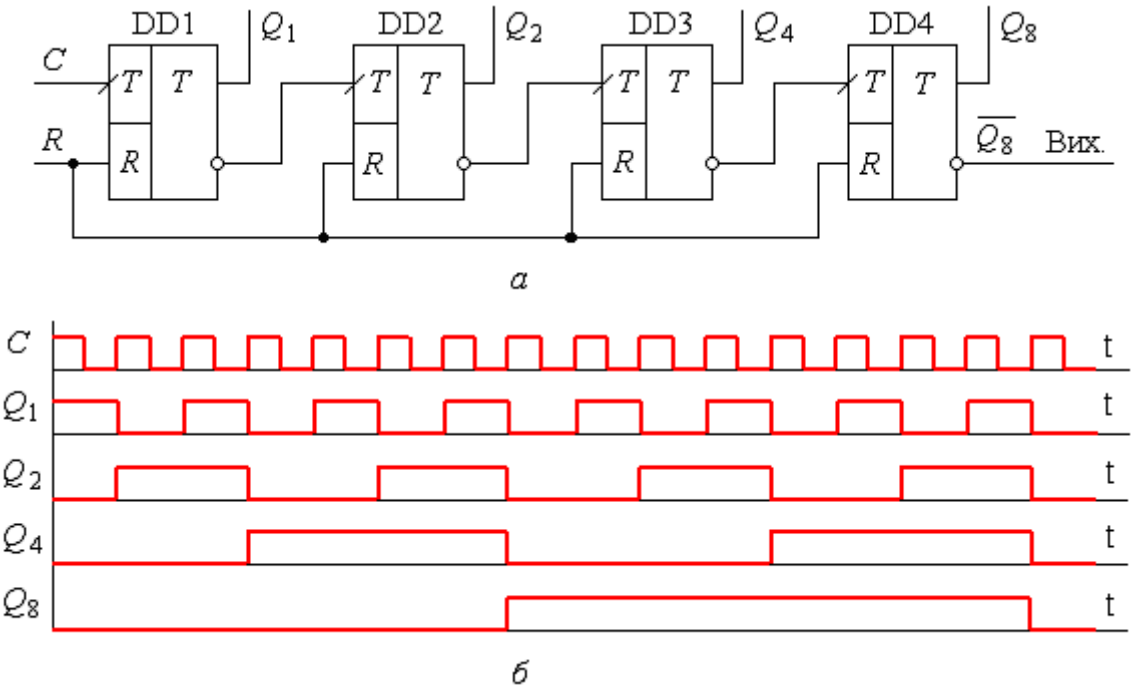


Рис. 14.1. Схема та часові діаграми асинхронного (послідовного) лічильника

Такі лічильники в своїй роботі використовують властивості *T*-тригера, оскільки він може як зберігати свій стан, так і додавати за модулем **2** вхідний сигнал до інформації, записаної попередньо. Одиночний *T*-тригер ділить на **2** частоту вхідної послідовності імпульсів. Послідовне включення *m* таких тригерів дає можливість поділити частоту вхідних імпульсів у $M = 2^m$ разів, або утворює лічильник з коефіцієнтом перерахунку *M* (модуль рахунку, ємність лічильника).

Робота найпростішого двійкового лічильника (рис. 14.1, ***а***) пояснюється даними, приведеними у табл. 14.1 і часовими діаграмами, зображеними на рис. 14.1, ***б***.

Таблиця 14.1. Таблиця станів найпростішого двійкового лічильника

Вхідні імпульси (N)	Q_8	Q_4	Q_2	Q_1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Перед подачею вхідних імпульсів тригери лічильника обнуляються. При подачі першого імпульсу на вхід *C* тригер, виконаний на мікросхемі DD1, у відповідності до його алгоритму роботи, змінює свій стан на протилежний. При цьому на його прямому виході з'явиться сигнал, еквівалентний логічній **"1"**, а на інверсному виході встановлюється логічний **"0"**. У відповідності до схеми, тригери з динамічною синхронізацією спрацьовують за фронтом вхідного імпульсу. Стани решти тригерів не зміняться, оскільки другий тригер DD2 по своєму тактовому входу сприйме перепад з **"1"** в **"0"**, що матиме місце на інверсному виході першого тригера, а тригери DD3 і DD4 на своїх входах не матимуть ніяких змін. У результаті після першого вхідного імпульсу значення виходів тригерів відповідатимуть значенням, приведеним у табл. 6.1 в другому рядку. Другий вхідний імпульс призведе до повторної зміни стану тригера DD1. Тепер на його інверсному виході матиме місце зміна стану з логічного **"0"** на логічну **"1"**, що сприймається тригером DD2 як фронт вхідного імпульсу. В результаті тригер DD2 змінить свій стан на протилежний, і на його виході Q_2 з'явиться сигнал логічної **"1"**, у той час як вихід Q_1 тригера DD1 перейде в

нульовий стан. Цей стан відповідатиме третьому рядку таблиці, відповідно до якого виходи тригерів зафіксують число **2** у двійковому коді. Третій імпульс знову змінить лише стан тригера DD1, прямий вихід якого відповідає молодшому розряду двійкового коду, що відображає кількість імпульсів, що були подані на вхід *C*-.

Рис. 14.1, ***б*** ілюструє часові співвідношення між станами тригерів, що зафіксовані їх прямими виходами. Зміни станів тригерів прийняті миттєвими.

З табл. 14.1 і часової діаграми (рис. 14.1, ***б***) бачимо, що виходи тригерів лічильника в двійковому коді відображають кількість поданих на вхід імпульсів *N*, якщо ця кількість менша числа *M*.

Якщо на вхід лічильника подано *N* імпульсів, то їх кількість, підрахована за допомогою лічильника, відповідає формулі:

$$N = K\,M + a_0\,2^0 + a_1\,2^1 + \ldots + a_{n-1}\,2^{n-1} = n\,M + \sum_{i=0}^{n-1} a_i\,2^i\,, \qquad (14.2)$$

де K – кількість вихідних імпульсів лічильника; $a_i \in 1, 0$ – рівень сигналу на i-ому виході лічильника; 2^i – вагові коефіцієнти кожного прямого виходу. Для урахування вагових коефіцієнтів виходи лічильника нумерують відповідними індексами. Це дає можливість з послідовності значень виходів $Q_8\,Q_4\,Q_2\,Q_1$, рівних, наприклад, **1011**, одразу ж зчитувати вміст лічильника, тобто кількість імпульсів, яка менша *M*.

При $N = M$ усі тригери лічильника обнуляються, і такий перехід фіксується зміною стану інверсного виходу тригера DD4 з **“0”** в **“1”**, що наступними аналогічними схемами повинно сприйматись як фронт вихідного імпульсу. Це дає можливість безпосереднього нарощування розрядності лічильників шляхом прямого з’єднання входу наступного лічильника з виходом попереднього. При однотипних лічильниках кількість тригерів подвоюється, і загальний коефіцієнт перерахунку визначатиметься формулою:

$$M_З = 2^{2m} = 2^m \cdot 2^m = M \cdot M\,.$$

Звідси витікає, що при безпосередньому нарощуванні кількості лічильників загальний коефіцієнт перерахунку визначається добутком відповідних коефіцієнтів окремих лічильників.

Зазвичай код, що визначається вихідними станами тригерів, змінюється у зростаючому напрямку (*додаючі лічильники*). Операція збільшення вмісту лічильника на одиницю називається *інкрементуванням*. При зворотній зміні станів лічильники звуться *віднімаючими*, а якщо напрямок рахунку може змінюватися, то лічильник має назву *реверсивного*. Операція зменшення вмісту лічильника на одиницю називається *декрементуванням*. Напрямок рахунку визначається як динамічними властивостями тригерів (перемикання за фронтом чи за спадом), так і способом з'єднання виходів попереднього тригера зі входом наступного. Якщо, наприклад, у лічильнику використовуються тригери, які змінюють свій стан за фронтом синхроімпульсу, то для організації додаючого лічильника необхідно з'єднати вхід наступного тригера з інверсним виходом попереднього, а для створення віднімаючого лічильника для зв'язку використовуються прямі виходи тригерів. При застосуванні тригерів, що змінюють свій стан за спадом синхроімпульсу, вказані зв'язки повинні бути протилежними.

Прикладом віднімаючого лічильника може служити схема, що приведена на рис. 14.2.

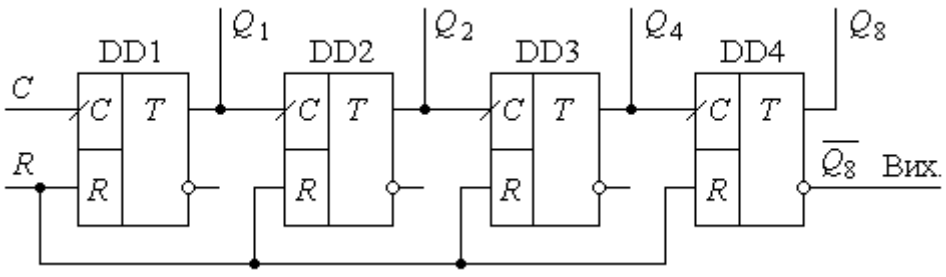


Рис. 14.2. Схема віднімаючого лічильника

Якщо всі тригери лічильника встановлені в одиничний стан, то при подачі першого вхідного імпульсу тригер DD1 змінить свій стан на протилежний, і на

виході Q_1 запишеться значення $Q_1 = 0$. Тригер DD2 при цьому не змінить свій стан, оскільки на його вході матиме місце спад імпульсу, на який він не реагує. Наступний вхідний імпульс призведе до появи на Q_1 сигналу одиничного рівня, при цьому тригер DD2 змінить свій стан, і на виходах тригерів зафіксується число $Q_8\,Q_4\,Q_2\,Q_1 = 1101_2 = 13_{10}$. Подібний процес продовжуватиметься до повного обнуління тригерів, а шістнадцятий імпульс забезпечить встановлення всіх тригерів в одиничний стан.

На рис. 14.3 приведена схема асинхронного реверсивного лічильника.

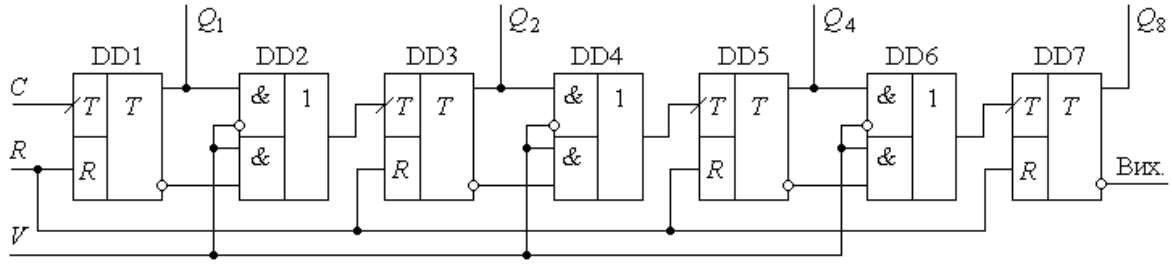


Рис. 14.3. Схема асинхронного реверсивного лічильника

Керуючий вхід V за допомогою двохвходових мультиплексорів DD2, DD4, DD6 забезпечує комутацію виходів тригерів DD1, DD3, DD5 на входи наступних – відповідно, DD3, DD5, DD7. При значенні сигналу на цьому вході $V = 1$ забезпечуватиметься підключення інверсних виходів і лічильник працюватиме на інкрементування вмісту, при $V = 0$ забезпечуватиметься режим декрементування.

R – асинхронний статичний вхід загального скидання, за яким прямі виходи всіх тригерів встановлюються в **“0”**.

За один цикл роботи лічильник приймає 2^m станів. Тому, з точки зору теорії скінченних автоматів, розглянутий лічильник – це асинхронний скінченний автомат з замкнутим циклом роботи, в якому перехід з одного стану до іншого забезпечується вхідними інформаційними імпульсами.

Кожен стан кодується m-розрядним двійковим словом. Граф-схема лічильника за модулем 8 як скінченного автомата (діаграма станів) приведена на рис. 14.4.

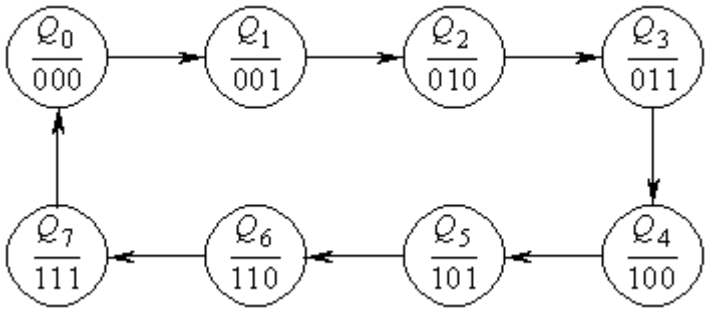


Рис. 14.4. Граф-схема лічильника за модулем 8 як скінченного автомата

Для зміни значення одного з старших розрядів кодового слова повинні змінюватись і всі молодші розряди. Перед зміною стану тригера, що зберігає інформацію про вміст старших розрядів кодового слова, повинні змінити свій стан і всі попередні тригери. Якщо прийняти, що зміна стану одного тригера проходить за інтервал часу затримки $t_З$, то між моментом подачі вхідного імпульсу і зміною стану тригера старшого розряду має місце часова затримка $t_{ЗМ} = m\, t_З$. Цей параметр називається *часовою спроможністю лічильника*, $t_{ЗМ}$ визначає мінімальний інтервал часу між двома вхідними імпульсами, при якому лічильник в найгіршому режимі перемикання встигає їх відпрацювати. Ця максимально можлива величина часової затримки визначає швидкодію лічильників при перерахунку імпульсних послідовностей. Максимально можлива частота вхідної послідовності імпульсів визначається за формулою

$$f_М = \frac{1}{m\, t_З}. \qquad (14.3)$$

Тому послідовний спосіб забезпечення перерахунку суттєво обмежує швидкодію двійкових лічильників. Пристрої такого типу називаються *лічильниками з послідовним переносом*.

Існують лічильники з іншими видами кодування станів виходів тригерів – наприклад, *унітарним* – коли стан лічильника представляється кількістю записаних в ньому одиниць; *одинарним* – коли стан лічильника визначається розташуванням однієї одиниці.

Лічильники з довільним модулем рахунку

Лічильники з довільним модулем рахунку мають значення M, що відрізняється від цілого ступеню числа **2** [1]. Прикладами таких лічильників можуть служити пристрої з $M = 10$; $M = 12$; $M = 24$; $M = 60$ і т. д. На практиці доводиться мати справу з лічильниками, призначеними для ділення частоти вхідних послідовностей імпульсів у сотні, тисячі і десятки тисяч разів, і далеко не завжди коефіцієнт ділення може бути кратним 2^m (m – ціле число).

При побудові лічильників цього типу використовують такі способи:

- виключення зайвих станів;
- зворотного зв’язку;
- кратних модулів.

Найбільшого розповсюдження набув спосіб виключення зайвих станів. Його реалізація переважно здійснюється наступним чином:

- застосуванням попередньої установки лічильника;
- використанням примусового його обнуління.

На рис. 14.5 приведена схема лічильника, в якому попереднє завантаження початкового стану забезпечується через асинхронні *S*-входи *T*-тригерів за допомогою логіки DD1…DD4 і керуючого входу *PE* (паралельного завантаження). Через входи D_1, D_2, D_4, D_8 у лічильник може бути записаний будь-який двійковий код у діапазоні **0000** – **1111**, значення якого буде зафіксоване на виходах Q_1, Q_2, Q_4, Q_8. Запис коду забезпечується до початку подачі вхідної послідовності імпульсів на *C*-вхід. Тому з моменту подачі вхідних імпульсів лічильник рахуватиме, починаючи не з нуля, а з занесеного коду.

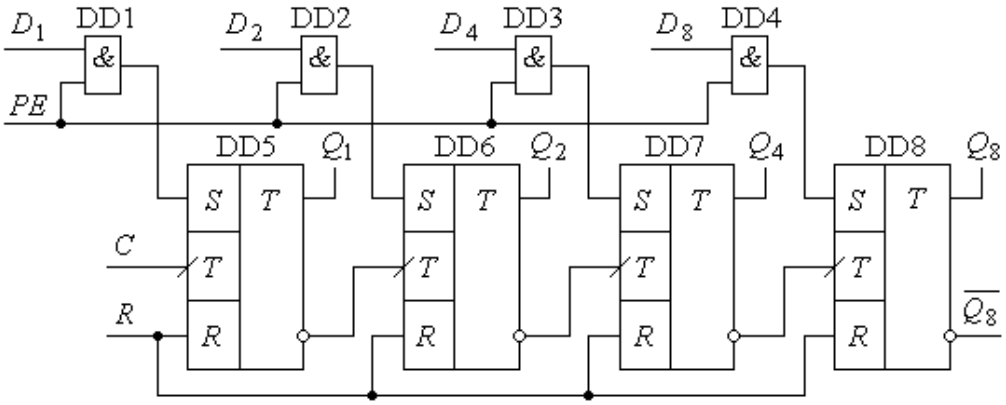


Рис. 14.5. Схема лічильника, в якому попереднє завантаження початкового стану забезпечується через асинхронні *S*-входи

Кількість імпульсів N, що може бути подана на *C*-вхід лічильника до переповнення, обчислюється за формулою $N = M - D$, тобто з M станів лічильника виключається D перших станів.

На рис. 14.6 приведені умовні позначення двійкового лічильника (рис. 14.6, ***а***) та лічильника з попередньою установкою (рис. 14.6, ***б***).

На відміну від двійкового лічильника, в позначенні лічильника з попередньою установкою зображені входи D ($D_1 \div D_8$) і допоміжний вхід паралельного завантаження *PE*. Особливість таких лічильників полягає в тому, що при циклічній роботі після завершення кожного циклу необхідно перезаписувати початковий код. Крім того, необхідно враховувати, що двійковий вихідний код у такому лічильнику при $D \neq 0$ не відповідає кількості імпульсів, поданих на вхід *C*.

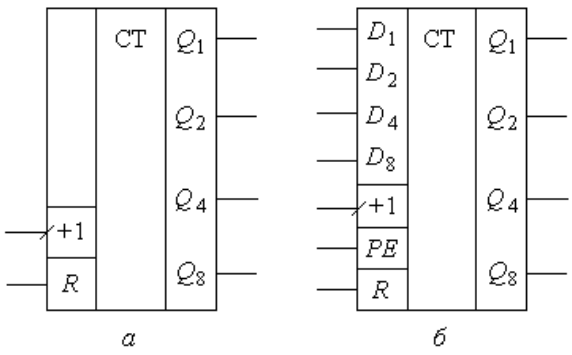


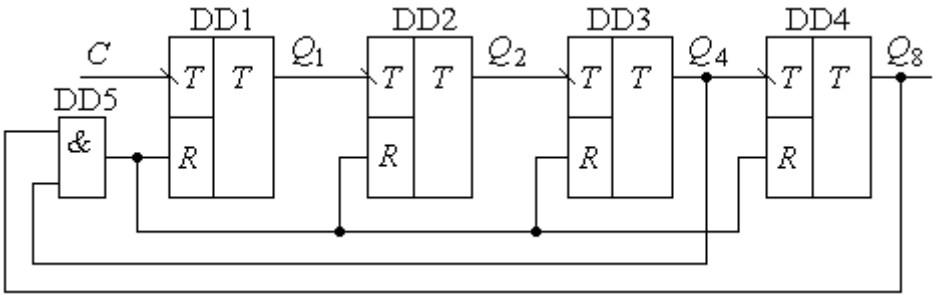
Рис. 14.6. Умовні позначення двійкового лічильника (а) та лічильника з попередньою установкою (б)

Примусове обнуління характеризується тим, що із загальної кількості станів M виключаються ті, які своїми значеннями перевищують встановлений модуль перерахунку $M_В$. Якщо, наприклад, лічильник з $M = 16$ повинен мати $M_В = 12$, то всі значення, котрі перевищують 12, повинні бути виключені. Тобто створення лічильника з модулем перерахунку $M_В$ досягається виконанням двох умов:

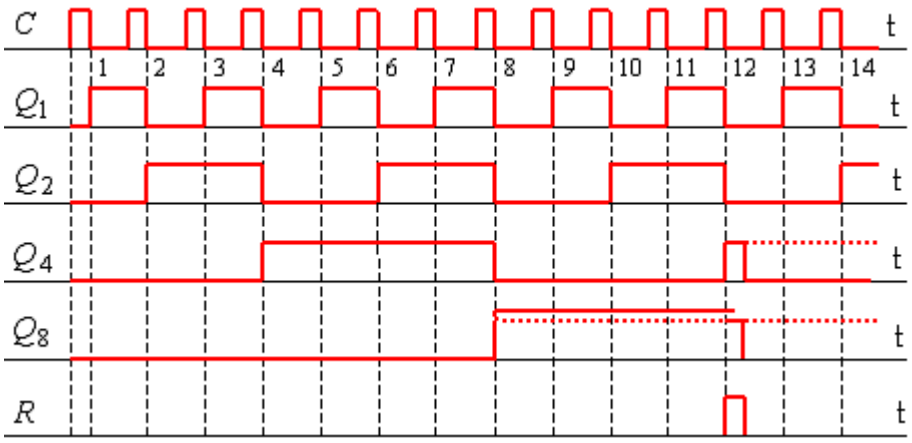
- обнуління лічильника при досягненні поточним кодом значення $M_В$;
- виключення всіх станів, що перевищують $M_В$.

Перша умова забезпечується, якщо сигнал обнуління: $R = M_В$.

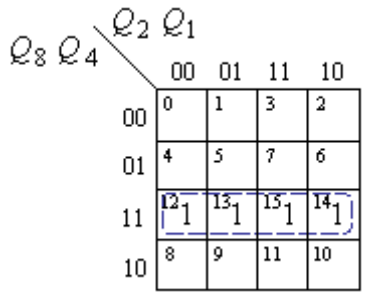
Для лічильника з $M_В = 12$ маємо: $R = Q_8\, Q_4\, \overline{Q_2}\, \overline{Q_1}$.



а



б



в

Рис. 14.7. Схема двійкового лічильника з $M_В = 12$ (а), часові діаграми, що пояснюють його роботу (б), карта Карно (в)

На рис. 14.7, ***а*** приводиться приклад схеми двійкового лічильника з $M_В = 12$, а на рис. 14.7, ***б*** – часові діаграми, що пояснюють його роботу. Як витікає з часових діаграм, за зрізом дванадцятого вхідного імпульсу на виходах Q_4 та Q_8 з'являються одиничні сигнали, які створюють сигнал обнуління R усіх тригерів. Враховуючи те, що стан лічильника Q_1, Q_2, Q_3, $Q_4 = 0011$ є короткочасним (він існує протягом часу обнуління тригерів лічильника). Його можна вважати перехідним і віднести до неробочих.

Таким чином функція виключення зайвих станів, яка забезпечує обнуління лічильника, виглядає наступним чином:

$$Y_В = \vee 12, 13, 14, 15 ,$$

яка після мінімізації за допомогою карти Карно (рис. 12.7, ***в***) прийме вигляд

$$Y_В = R = Q_8\ Q_4 .$$

Спосіб зворотного зв'язку використовується досить рідко. Особливість цього способу полягає у наступному. Припустимо, що лічильник містить у собі n тригерів. Виділимо з цієї групи меншу – з n_2 тригерами, тоді $n = n_1 + n_2$. У виділеній групі заводиться зворотній зв'язок так, щоб при появі одиниць на виходах тригерів перший тригер скидався в нуль. Тоді коефіцієнт перерахунку в цій групі становитиме $2^{n_2} - 1$, а загальний коефіцієнт перерахунку:

$$M = 2^{n_1} \cdot (2^{n_2} - 1) = 2^n - 2^{n_2} . \qquad (14.4)$$

Спосіб кратних модулів пояснюється схемою, що приведена на рис. 14.8.

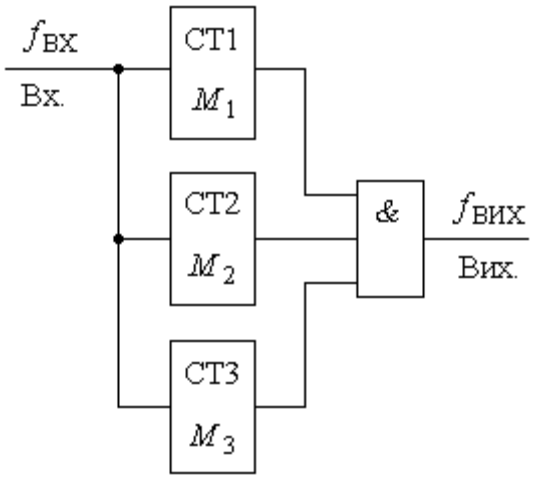


Рис. 14.8. Схема способу кратних модулів

Лічильники СТ1, СТ2, СТ3 з модулями рахунку, відповідно, M_1, M_2, M_3 з'єднані по входу паралельно, а їх виходи об'єднані елементом **3І**. Імпульс на виході з'явиться лише тоді, коли одночасно на виходах всіх лічильників з'являться вихідні імпульси. У такій схемі частота імпульсів вхідної послідовності $f_{вх}$ буде ділитися на коефіцієнт $M = M_1 M_2 M_3$, і частота вихідних імпульсів матиме значення:

$$f_{вих} = \frac{f_{вх}}{M}. \qquad (14.5)$$

Синхронні лічильники

У попередніх лекціях в якості скінченних автоматів розглядалися лічильники, в яких синхросигнали та інформаційні сигнали, що записуються в тригери, розділені по входу. В таких схемах інформаційний сигнал, що записується у тригер, підготовляється одразу після встановлення попереднього значення, а синхросигнал подається паралельно на всі тригери [1]. Час підготовки $t_П$ – це час, який необхідно враховувати як час розповсюдження сигналу через допоміжні логічні елементи, і він завжди має величину, значно меншу, ніж час перемикання одиночного тригера. Тому швидкодія подібного типу лічильників має бути значно більшою, ніж швидкодія лічильників з

послідовним переносом. Лічильники такого типу називаються *лічильниками з паралельним переносом*.

Виходячи з цього, розглянемо синтез синхронного лічильника з паралельним переносом з використанням *JK*-тригерів, який забезпечить коефіцієнт $M = 8$.

Таблиця станів лічильника, а також сигнали управління *J* та *K* входів тригерів приведені у табл. 14.2 (умовним позначенням * зображений байдужий стан відповідного входу).

Таблиця 14.2. Таблиця станів лічильника, а також сигнали управління *J* та *K* входів тригерів

C_n	Q_{4n}	Q_{2n}	Q_{1n}	$Q_{4(n+1)}$	$Q_{2(n+1)}$	$Q_{1(n+1)}$	J_4	K_4	J_2	K_2	J_1	K_1
0	0	0	0	0	0	1	0	*	0	*	1	*
1	0	0	1	0	1	0	0	*	1	*	*	1
2	0	1	0	0	1	1	0	*	*	0	1	*
3	0	1	1	1	0	0	1	*	*	1	*	1
4	1	0	0	1	0	1	*	0	0	*	1	*
5	1	0	1	1	1	0	*	0	1	*	*	1
6	1	1	0	1	1	1	*	0	*	0	1	*
7	1	1	1	0	0	0	*	1	*	1	*	1

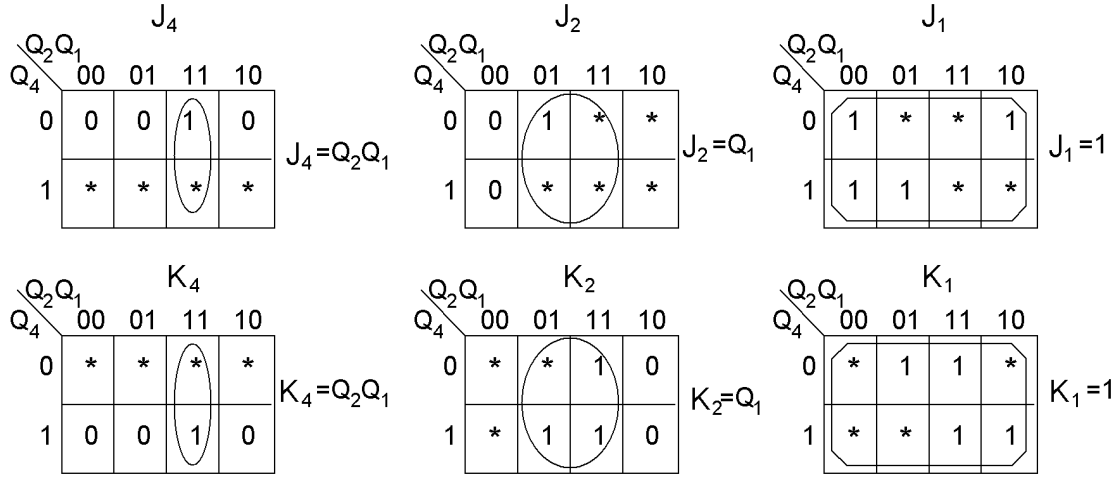


Рис. 14. 9. Карти Карно для входів J_4, K_4, J_2, K_2, J_1, K_1

Використовуючи карти Карно для входів J_4, K_4, J_2, K_2, J_1, K_1 (рис. 12.9), знаходимо вирази для сигналів їх збудження. Відповідно, схема лічильника приведена на рис. 14.10.

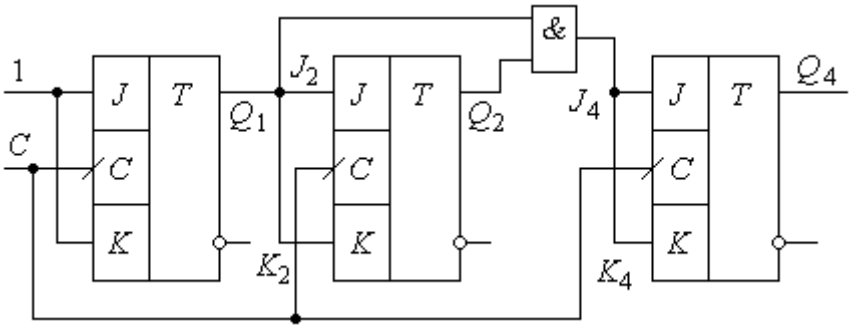


Рис. 14.10. Схема лічильника

Узагальнюючи отриманий результат, можна стверджувати, що, якщо для лічильника з $M = 2$:

$J_1 = 1$; $K_1 = 1$;

для лічильника з $M = 4$:

$J_2 = Q_1$; $K_2 = Q_1$;

для лічильника з $M = 8$:

$J_4 = Q_1 Q_2 = J_2 Q_2$; $K_4 = Q_1 Q_2 = K_2 Q_2$;

то для лічильника з $M = 16$ матимемо:

$J_8 = Q_1 Q_2 Q_4 = J_4 Q_4$; $K_8 = Q_1 Q_2 Q_4 = K_4 Q_4$.

Відповідно, для лічильника з $M = N$:

$J_N = Q_1 Q_2 \ldots Q_{(N-1)} = J_{(N-1)} Q_{(N-1)}$; $K_N = Q_1 Q_2 \ldots Q_{(N-1)} = K_{(N-1)} Q_{(N-1)}$.

Для синхронних лічильників зворотного рахунку (віднімаючих) аналогічно можемо записати:

$$\begin{cases} J_1 = K_1 = 1; \\ J_2 = K_2 = \overline{Q_1}; \\ J_4 = K_4 = \overline{Q_1}\,\overline{Q_2} = J_2\,\overline{Q_2}; \\ J_8 = K_8 = \overline{Q_1}\,\overline{Q_2}\,\overline{Q_4} = J_4\,\overline{Q_4}; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ J_N = K_N = \overline{Q_1}\,\overline{Q_2}\ldots\overline{Q_{(N-1)}} = J_{(N-1)}\,\overline{Q_{(N-1)}}. \end{cases}$$

За аналогією з асинхронними лічильниками, з використанням допоміжної логіки синхронні лічильники можуть бути реверсивними.

Прикладом лічильника з паралельним переносом є мікросхема КМОН К1561ИЕ10, спрощена схема якої приведена на рис. 1.11.

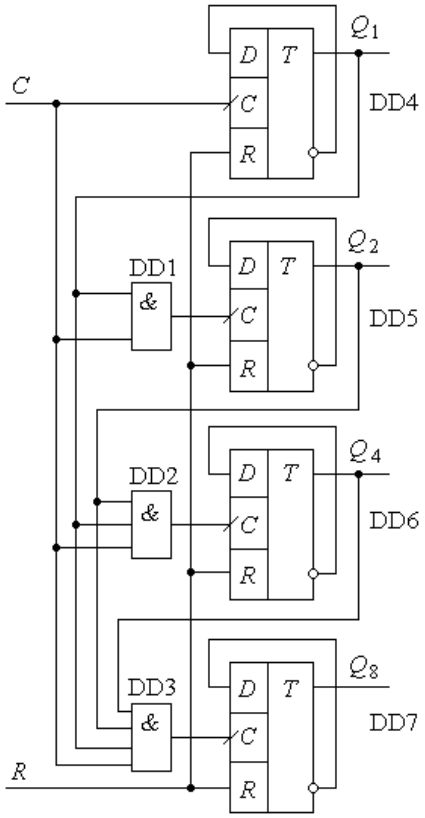


Рис. 14.11. Спрощена схема мікросхеми К1561ИЕ10

Лічильник виготовлений на *D*-тригерах DD4…DD7, які працюють у режимі *T*-тригерів. Допоміжна логіка DD1…DD3 забезпечує швидкий доступ послідовності імпульсів, що підраховуються, до синхровходу. При подачі сигналу високого рівня на *R*-вхід усі прямі виходи тригерів Q_1, Q_2, Q_4, Q_8 встановлюються у нульовий стан. За зрізом першого тактового сигналу вихід Q_1 тригера DD4 встановлюється в **“1”** і цим підготовлює доступ синхросигналу до синхровходу тригера DD5. Другий імпульс своїм зрізом встановить тригер DD4 у початковий стан, а DD5 – в **“1”**.

Цей процес триватиме доти, поки всі тригери не будуть встановлені у стан, при якому $Q_1 = Q_2 = Q_4 = Q_8 = 1$. Наступний за цим (шістнадцятий) синхроімпульс встановить всі тригери в нуль, і цикл закінчиться.

Як випливає з роботи лічильника, всі тригери сприймають імпульси вхідної послідовності практично одночасно, аналогічно відбувається зміна їх станів, що забезпечує високу швидкодію лічильника. Фактично вона визначається часом перемикання тригера і незначною, порівняно з ним, затримкою вхідної логіки.

З аналізу роботи лічильника бачимо, що перемикання кожного наступного тригера при приході чергового імпульсу має місце лише тоді, коли всі попередні тригери встановлені в **“1”**. Математично це можна виразити наступною формулою:

$$Q_{i(n+1)} = \overline{Q_{in}}\ p_i + Q_{in}\ \overline{p_i} = Q_{in} \oplus p_i, \qquad (14.6)$$

де Q_{in} – стан i-го тригера до подачі імпульсу; $p_i = Q_{0n} \cdot Q_{1n} \cdot \ldots \cdot Q_{(i-1)n}$ – узагальнений сигнал перенесення.

У розглянутих лічильниках, на відміну від пристроїв з послідовним перенесенням, напрямок рахунку не залежить від типу динамічного входу тригера, а визначається виключно тим, який з виходів тригера (прямий чи інверсний) використовується для формування сигналу перенесення. Оскільки ми впевнились, що лічильник, схема якого приведена на рис. 14.11, є додаючим, то при використанні інверсних виходів тригерів отримаємо лічильник віднімаючий. Таким чином алгоритм перемикання в реверсивних лічильниках (додавання/віднімання) залишається незмінним.

Недоліки лічильників з паралельним перенесенням полягають, перш за все, у критичності до тривалості вхідних імпульсів та їх фронтів. Тому при використанні таких пристроїв слід звертати увагу на довідкові дані по цим параметрам. Обумовлено це тим, що, незважаючи на те, що тригери лічильника перемикаються майже одночасно і час установки двійкового коду $t_{КУ}$ на виході лічильника фактично визначається часом установки тригера $t_Т$, але для

підготовки до наступного перемикання необхідно, щоб пройшов деякий час, який називається часом підготовки $t_П$. За цей час послідовно формуються сигнали перенесення на всіх входах елементів **І**. Величина $t_П$ залежить від встановленого в попередньому такті коду.

Складність практичної реалізації лічильника з великим значенням модуля M є другим їх недоліком. Це обумовлено ускладненням логіки переносу з великою кількістю входів.

Задача нарощування лічильників розв'язується декількома шляхами. Наприклад, у мікросхемі К1561ИЕ10 синхровхід запаралелюється через елемент **АБО** допоміжним інверсним входом $\overline{ES}$. Цей вхід дає можливість організувати рахунок як за фронтом імпульсу, так і за зрізом, а при послідовному нарощуванні використовується як вхід асинхронного вводу від виходу Q_8 попереднього лічильника. Такий спосіб нарощування називається *комбінованим* (*послідовно-паралельним*).

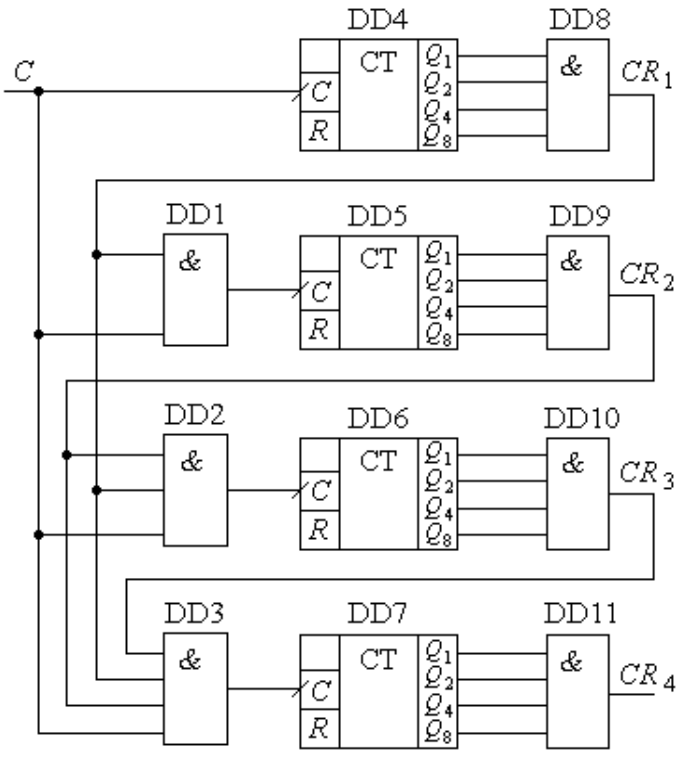


Рис. 14.12. Структура з комбінованим паралельно-паралельним перенесенням

На практиці часто використовуються структури з *комбінованим паралельно-паралельним перенесенням*. Особливість такого перенесення ілюструється рис. 14.12. За аналогією зі схемою рис. 14.11, логічні елементи DD1…DD3 забезпечують паралельне перенесення від лічильників DD4…DD7. Формування сигналів перенесення $CR_1 \ldots CR_4$ з кожного лічильника виконують логічні елементи DD8…DD11 при умові, що виходи $Q_1 \ldots Q_8$ кожного з лічильників встановлені в **"1"**. У такій структурі лічильника швидкодія визначається часом установки коду $t_{КУ}$. Ця величина практично визначається часом установки коду в межах одного лічильника. Час підготовки $t_{П\,max}$ у лічильнику визначається сумою затримок у логічних елементах **І**. Тому максимальна частота перемикання визначатиметься як:

$$f_{max} = \frac{1}{t_{КУ} + t_{П\,max}}. \qquad (14.7)$$

Скінченні автомати на основі лічильників

Лічильники можуть розглядатись як скінченні автомати з циклічним режимом роботи. Але іноді виникає необхідність побудови автомата, що працює в майже циклічному режимі. У таких випадках для побудови автоматів корисно використати готові мікросхеми лічильників з базовим циклічним режимом роботи і доповнити комбінаційною логікою для забезпечення нециклічних переходів.

На рис. 14.13, як приклад, приводиться діаграма станів автомата з майже циклічним режимом роботи. В такому випадку необхідно вибрати мікросхему лічильника і за допомогою логічних елементів забезпечити необхідні переходи.

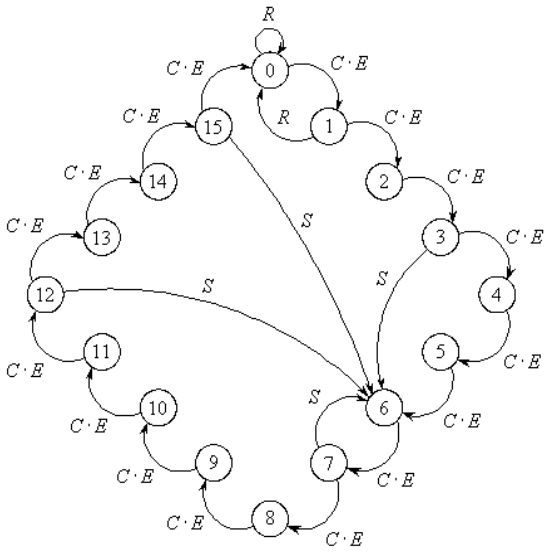


Рис. 14.13. Діаграма станів автомата з майже циклічним режимом роботи

Функціональна схема автомата, що відповідає приведеній діаграмі станів, показана на рис. 14.14.

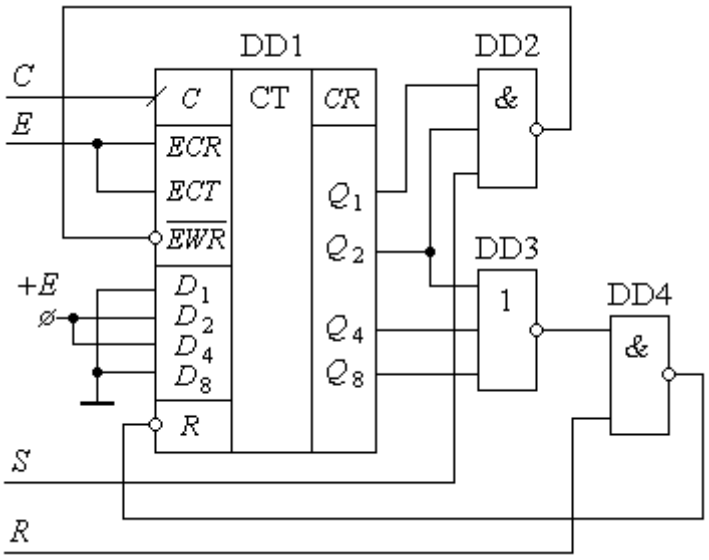


Рис. 14.14. Функціональна схема автомата, що відповідає приведеній діаграмі станів

Вхід автомата *R* фактично призначений для примусового встановлення автомата в стан **0**, а сигнал *S* – аналогічно, для забезпечення примусового встановлення автомата в стан **6**.

Розглянемо особливості проектування таких автоматів на прикладі лічильника з $M = 8$, який за сигналом керування G повинен працювати у двійковому коді ($G = 0$) або в режимі відліку відповідно до коду Грея ($G = 1$). Граф-схема автомата матиме вигляд, зображений на рис. 14.15.

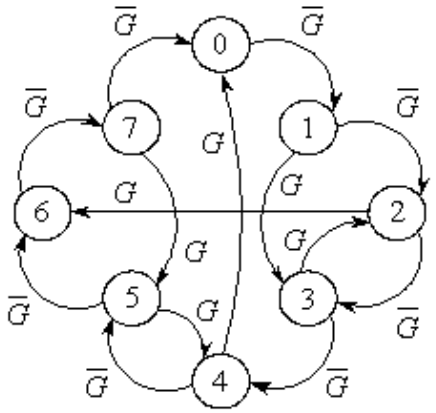


Рис. 14.15. Граф-схема автомата

Виходячи з граф-схеми, можемо визначитись, що для побудови такого автомата необхідно використати двійковий лічильник з попередньою асинхронною установкою.

Оскільки зв'язок між розрядами бінарного коду b_0, b_1, b_2 і коду Грея g_0, g_1, g_2 встановлюється залежностями:

$$g_0 = b_0\,\overline{b_1} + \overline{b_0}\,b_1 = b_0 \oplus b_1;$$

$$g_1 = b_2\,\overline{b_1} + \overline{b_2}\,b_1 = b_2 \oplus b_1;$$

то матимемо функціональну схему автомата (рис. 12.16).

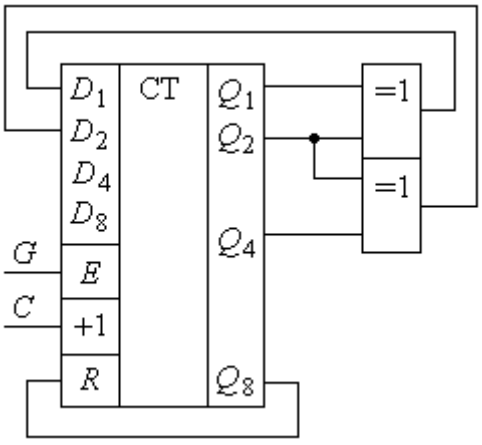


Рис. 14.16. Функціональна схема автомата

Із функціональної схеми витікає, що для побудови автомата для $M = 16$ досить зняти подачу сигналу на вхід R і додати, аналогічно попередньому, ще один логічний елемент **“ВИКЛ. АБО”**.

Використання лічильників при побудові цифрових автоматів дозволяє значно спрощувати апаратну реалізацію, але вимагає від інженерів-розробників високої кваліфікації.

Перелік посилань

1. Рябенький В.М., Жуйков В.Я., Гулий В.Д. Цифрова схемотехніка: Навч. посібник. – Львів: Новий світ – 2000, 2009, – 736 с.

Лекція №15

Загальне поняття про регістри

Регістрами називають цифрові пристрої, призначені для тимчасового зберігання та перетворення інформації, що подається на них у вигляді багаторозрядних двійкових чисел [1]. Кількість тригерів, розміщених паралельно або з'єднаних послідовно, визначає *розрядність* регістрів.

Регістри можуть класифікуватися за різними ознаками, але основними є способи введення та виведення інформації. Виходячи з цього, вони розділяються на дві групи: *накопичувальні* (або *регістри пам'яті*) та *послідовні* (або *регістри зсуву*). У свою чергу, послідовні регістри можуть забезпечувати *послідовний, паралельний* та *комбінований* способи введення та виведення інформації.

Записана у тригери інформація може виводитись у прямому коді, зворотному або одночасно в прямому та зворотному.

Регістри зсуву можуть бути *однонаправленими*, тобто забезпечувати зсув лише в одному напрямку, або *реверсивними* (*двонаправленими*).

Регістри пам'яті

Регістри пам'яті призначені для запису, тимчасового зберігання та передачі невеликих обсягів інформації, представленої у двійковому коді [1].

Елементами пам'яті в них виступають здебільшого *D*-тригери, тому регістри подібного типу можуть виготовлятись з використанням мікросхем, що містять набори синхронних тригерів, – наприклад, мікросхеми КР1533ТМ2 (SN74ALS74A), ТМ7, ТМ8, ТМ9. Керування режимами запису / зчитування може бути забезпечено вхідною та вихідною логікою, побудованою, наприклад, з використанням ЛЕ **2I** (ЛИ1 (ALS08)).

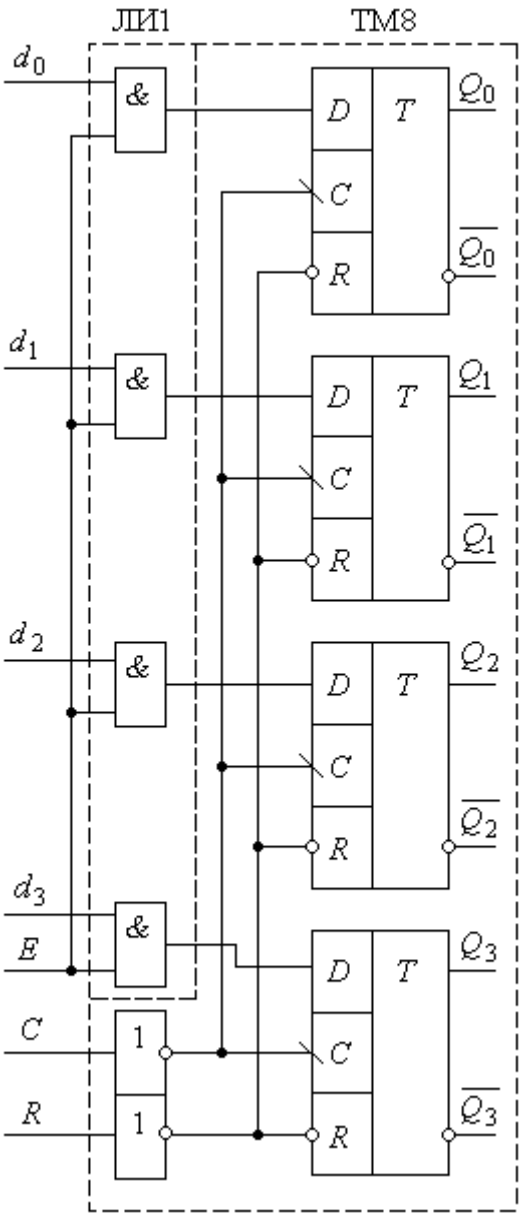


Рис. 15.1. Функціональна схема регістра

На рис. 15.1 наведений приклад реалізації функціональної схеми регістра з дозволяючим входом запису *E*, прямими та інверсними виходами. Запис інформації в *D*-тригери може виконуватись при високому рівні сигналу на вході *E* за фронтом синхросигналу *C*. Зчитування інформації в прямому або зворотному кодах виконується безпосередньо з виходів тригерів Q_i та $\overline{Q_i}$. Для забезпечення керування процесом зчитування може бути використана та ж мікросхема або створений режим мультиплексування виходів тригера на один вихід регістра. При використанні такого пристрою у структурах з великою

кількістю регістрів вхід *E* може використовуватись для адресного звернення до одного з них через відповідні дешифратори адреси.

Реальним прикладом є мікросхема ИР5. Це чотирьохрозрядний паралельний регістр даних, умовне зображення якого приведене на рис. 15.2. Він має по два входи D_{iA} та D_{iB} для одного запам'ятовуючого елемента. Вибір входу забезпечується рівнем сигналу, що подається на вхід *S*, а запис – за зрізом синхросигналу. Враховуючи наявність логіки на входах та виходах тригерів, режими роботи регістрів описуються таблицями станів.

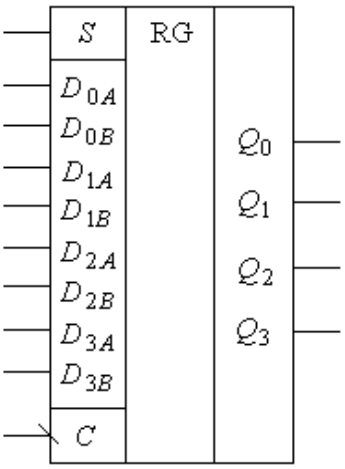


Рис. 15.2. Мікросхема ИР5

Приклад таблиці станів для регістра ИР5 приводиться у табл. 15.1. Таблиці станів повинні відображати всі режими роботи регістра, і для кожного з них вказують на однозначну відповідність між рівнями логічних сигналів на його вході та виході.

Таблиця 15.1. Таблиця станів регістра пам'яті

Входи				Вихід
D_{iA}	D_{iB}	S	C	Q_i
0/1	x	0	⌊	0/1
x	0/1	1	⌊	0/1

На рис. 15.3 приводиться функціональна схема, що містить чотири регістри, які об'єднані єдиною шиною даних ($d_0 \ldots d_3$). Дешифратори DD1 і

DD2 на основі адресних сигналів ($a_0 \ldots a_3$) генерують сигнали дозволу на читання інформації з регістра *ER* та запису *EW*. При виборі двох регістрів по їх адресах на відповідних входах *ER* і *EW* встановлюються сигнали високого рівня, що дозволяє за фронтом синхросигналу *C* зчитати інформацію з одного регістра та записати в інший.

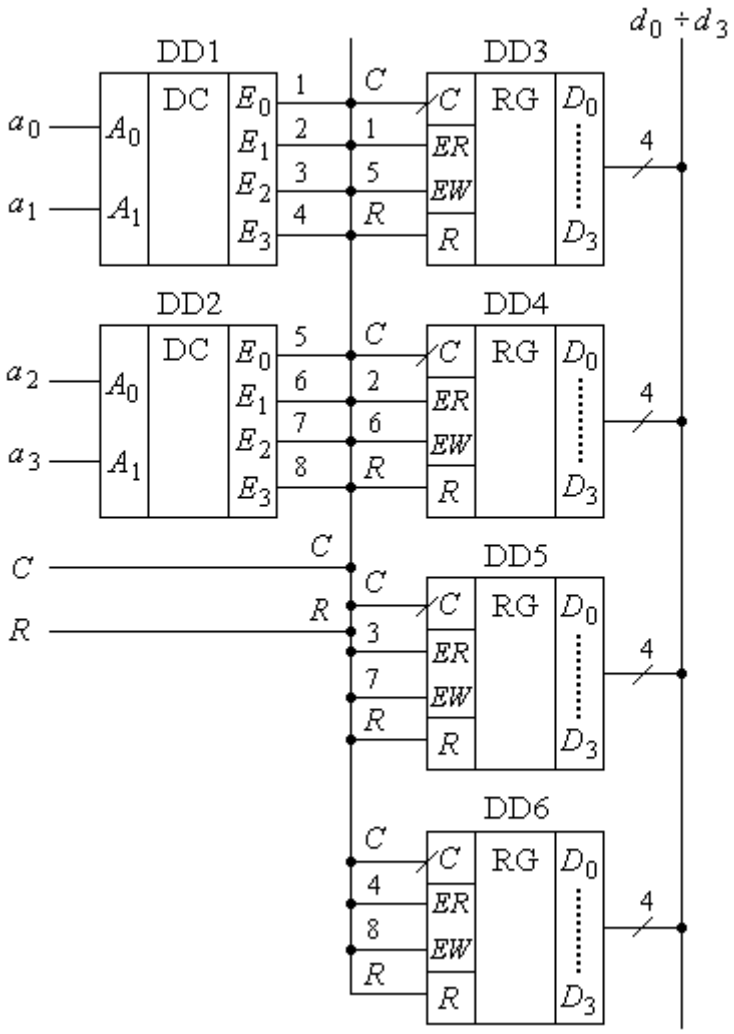
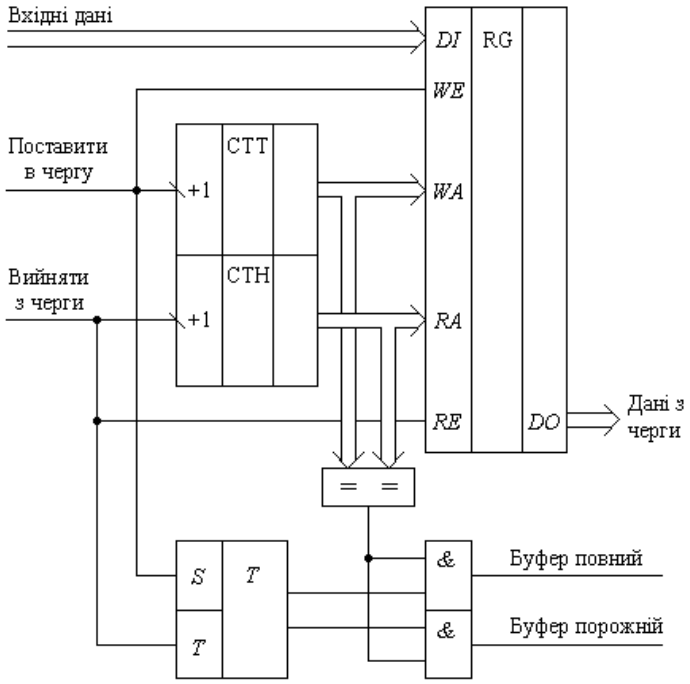


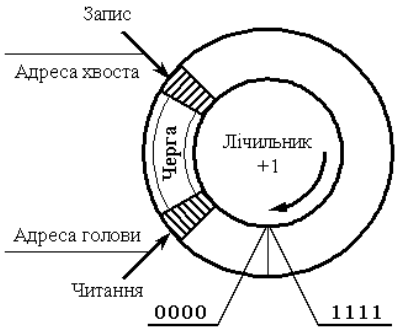
Рис. 15.3. Функціональна схема, що містить чотири регістри, які об’єднані єдиною шиною даних

Окрім адресованих регістрів загального призначення, у цифровій апаратурі використовуються допоміжні запам’ятовуючі пристрої з “неявно” вираженою адресацією, які служать для зберігання черг і називаються іноді *“буферами даних”*. Ці пристрої часто будують на основі регістрової пам’яті.

Необхідність у такому буфері виникає тоді, коли джерело даних постачає приймачу слова, що розподілені у часі нерівномірно, причому інтервали часу між двома словами можуть іноді бути меншими, ніж час, необхідний приймачу для обробки одного слова. Подібна ситуація може приводити до того, що частина даних втрачатиметься. Якщо такі втрати недопустимі, то між джерелом та приймачем включається *буфер "черги"* для створення і зберігання черги слів для їх послідуючої обробки, або *буфер типу FIFO* (***F**irst **I**n - **F**irst **O**ut* – перший ввійшов - перший вийшов).



а)



б)

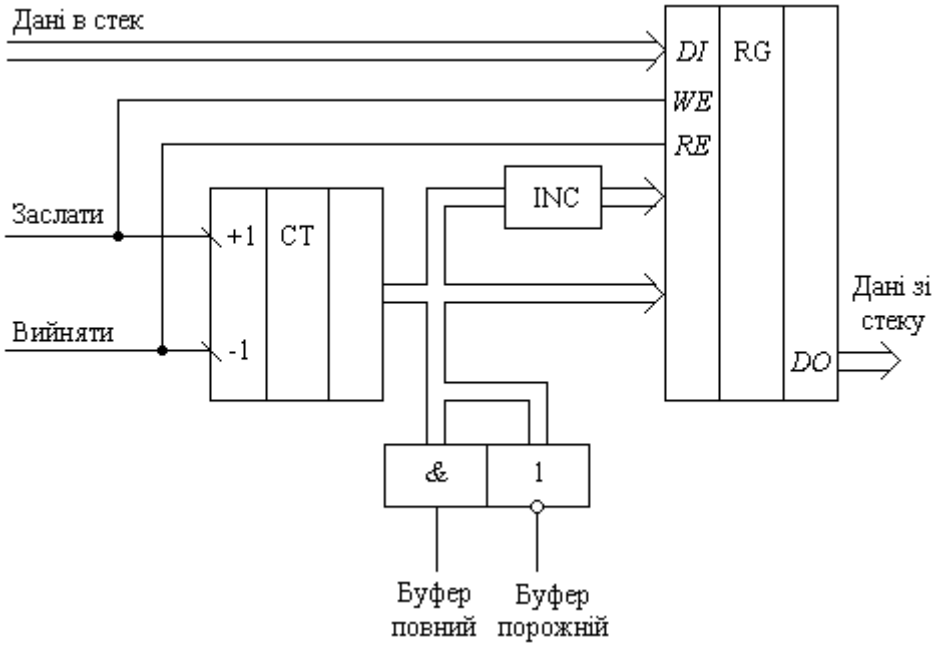
Рис. 15.4. Функціональна схема буфера, побудованого на основі регістрової пам'яті (***а***); набір регістрів пам'яті у вигляді кільця (***б***)

Функціональна схема буфера, побудованого на основі регістрової пам'яті, зображена на рис. 15.4, ***а***. На рис. 15.4, ***б*** набір регістрів пам'яті, тобто адреси регістрів пам'яті, зображені у вигляді кільця. Частина регістрів заповнена словами черги – решта вільні. Адреса запису при постановці в чергу задається лічильником хвоста черги (СТТ). Сигнал "Поставити в чергу" подається на вхід *WE* дозволу запису і забезпечує запис слова даних, що поступили по вхідній шини *DI* (***D****ata* ***I****n*) у той регістр пам'яті, номер якого зберігається в СТТ. За зрізом сигналу "Поставити в чергу" вихідний код СТТ збільшується на **1**, підготовляючи адресу запису для чергового слова. При появі сигналу "Вийняти з черги" на вихідній шині *DO* (***D****ata* ***O****ut*) з'явиться слово, яке зберігалося в тому регістрі пам'яті, номер якого задається кодом лічильника голови СТН. За зрізом сигналу вихідний код лічильника збільшується на **1**, підготовлюючи тим самим до видачі наступне слово, яке стало першим у черзі. Переповнення лічильника СТТ не призводить до ускладнення ситуації, оскільки після максимально можливого коду **1111…11** у ньому автоматично з'явиться код **0000…00**. Черга в кільці просто переміститься хвостом через нульову відмітку лічильника. Так само з часом матиме місце переміщення голови. В процесі нормальної роботи черга рухатиметься по кільцю значень адрес за часовою стрілкою хвостом вперед, зростаючи чи скорочуючись у відповідності до швидкості передавача. Схема буфера *FIFO* повинна сигналізувати про появу двох особливих ситуацій. Перша – "буфер повний". Це означає, що в нього неможливо більше записувати і необхідно зупинити передачу даних. Друга – "буфер порожній". Це означає, що з нього неможливо брати дані і необхідно зупинити приймач. Обидві ситуації мають спільну ознаку: рівність кодів обох лічильників після зникнення вхідного сигналу. Така ознака виявляється компаратором. Якщо коди лічильників стали рівними після чергової вибірки з черги, то це означає, що черги немає і буфер порожній. Якщо ж коди лічильників стали рівними після чергового запису в чергу, то буфер повний. Характер останнього звернення до буфера запам'ятовується в *RS*-тригері, який при появі особливих ситуацій через

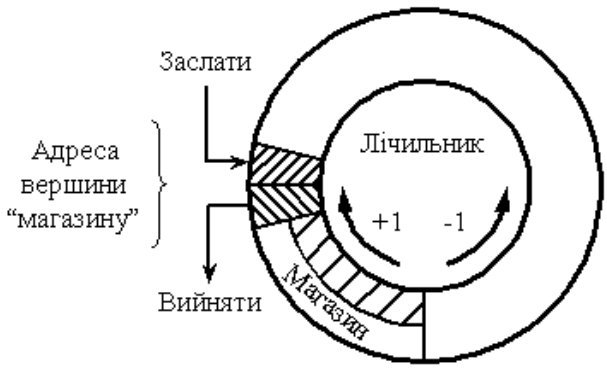
елементи **2І** видає відповідні інформаційні сигнали, призначені для подальшого прийняття рішень.

Іншим широко використовуваним буфером є *буфер типу "магазин"*, або *"стек"* (*Stack*), або *буфер типу LIFO* (***L**ast **I**n - **F**irst **O**ut* – останній ввійшов - перший вийшов).

Стекові структури даних використовуються у цифрових пристроях, в яких процес виконання завдання переривається більш терміновим, тому всі дані, що були пов'язані з перерваним завданням, розміщуються у буфері типу "магазин" для тимчасового зберігання. Виконання термінового завдання також може бути перервано появою більш термінового, і т. д. Внаслідок таких переривань у буфері накопичуватимуться нові й нові дані, створюючи чергу слів, очікуючих обробки. Вибірка з буфера слів відбувається у відповідності до вимог їх черговості, внаслідок чого першими виймаються слова, які були занесені в буфер останніми.



а



б

Рис. 15.5. Функціональна схема буфера типу “магазин” (***а***);
реверсивний лічильник адреси (***б***)

На рис. 15.5, ***а*** приведена функціональна схема буфера типу “магазин”. Основою його є регістрова пам’ять, а також реверсивний лічильник адреси, в якому зберігається номер регістра вершини стеку (рис. 13.5, ***б***). Коли подається команда “Заслати у стек” (*Push*), то вміст лічильника інкрементується на **1**; при команді “Вийняти зі стеку” (*Pop*) його вміст декрементується. Як видно з діаграми, адреса, за якою засилається слово у стек, завжди на **1** більше адреси, за якою слово зчитується зі стеку. Постійну різницю на **1** між адресами запису та зчитування підтримує пристрій, який називається *інкременатором* (INC). Два особливі стани – “буфер повний” та “буфер порожній” – визначаються за максимально можливим і нульовим значенням вмісту лічильника.

Регістри зсуву

Регістрами зсуву називаються такі регістри, вміст яких при подачі керуючих сигналів може змінюватись у бік старших чи молодших розрядів [1]. Такі пристрої складаються з ланки динамічних тригерів, спрацьовування яких виконується за фронтом або спадом синхронізуючого імпульсу *C*-.

Під терміном *“прозорість”* мається на увазі властивість тригера при активному рівні сигналу на вході *D*- адекватно з його типом відслідковувати на виході всі перемикання керуючих входів; якщо це *D*-тригер, то просто

повторюється стан *D*-входу. Іншими словами, цим терміном охоплюються тригери зі статичним входом.

“Непрозорість” тригерів – це їх властивість навіть при активному рівні сигналу на *C*-вході не передавати на вихід змін керуючих входів, що з’являються вслід за перемикаючим фронтом. Цією властивістю охоплюються всі тригери, які перемикаються за фронтом або зрізом синхроімпульсу, що поступає на вхід *C*-.

Складність застосування статичних тригерів у зсувних регістрах обумовлена тим, що при активному рівні сигналу на синхронізуючих входах всіх тригерів перемикання одного з них приведе до перемикання послідуючих, а їх кількість визначатиметься тривалістю синхросигналу та швидкодією тригерів. Однак під цим не слід розуміти, що застосування статичних тригерів у схемах з послідовним зсувом принципово неможливе. Така можливість з’являється, наприклад, при використанні двофазної синхронізації, яка дозволяє розділити у часі процеси перемикання на окремих ділянках схеми. Прикладом ефективного застосування двофазної синхронізації може служити мікропроцесорний комплект ВІС серії К580. В інтегральних схемах середнього ступеню інтеграції використовується однофазна синхронізація.

На рис. 15.6 наведена схема зсувного регістра, побудованого на *D*-тригерах. Інформація, призначена для запису в регістр і представлена у послідовному форматі, поступає на *D*-вхід першого тригера. Її запис у перший та наступні тригери здійснюється за фронтом синхроімпульсу, що поступає одночасно на входи *C*- всіх тригерів. Зчитування інформації виконується також у послідовному форматі з виходу *Q* останнього тригера шляхом подання на синхровходи серії імпульсів. У паралельному форматі вона може бути зчитана з виводів *Q* тригерів, якщо такі виводи передбачені в мікросхемі регістра. За подібною схемою побудований регістр 564ИР2, що містить два незалежні чотирьохрозрядні однонаправлені регістри.

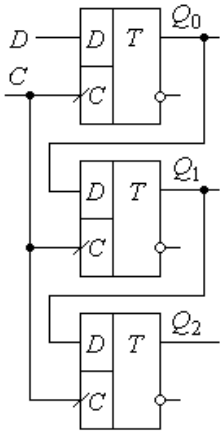


Рис. 15.6. Схема регістра зсувну, побудованого на *D*-тригерах

На рис. 15.7 приводиться варіант схеми зсувного регістра на *JK*-тригерах, кожен з яких працює в режимі *D*-тригера. Ці режими забезпечуються для тригерів DD1…DD3 протифазними виходами попередніх, а *D*-режим тригера DD0 забезпечується інвертором DD5.1. Доповнений логічними елементами DD4.1…DD4.4, він дозволяє виконувати послідовне та паралельне введення інформації, а також послідовне або паралельне її виведення.

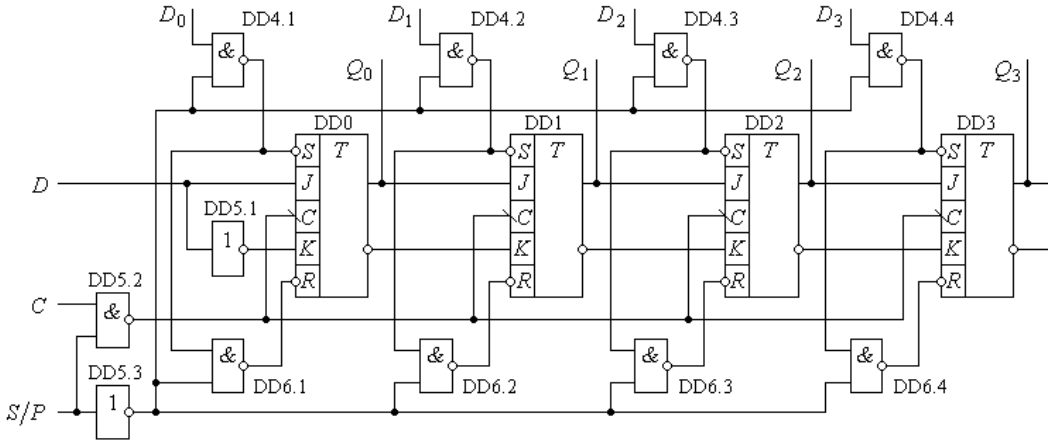


Рис. 15.7. Схема регістра зсуву на *JK*-тригерах, кожен з яких працює в режимі *D*-тригера

Часова діаграма, яку наведено на рис. 15.8, ілюструє процедуру послідовного введення по входу *S*- за наявності тактових імпульсів *C*- та логічної одиниці на вході вибору режимів введення: $S/P = 1$.

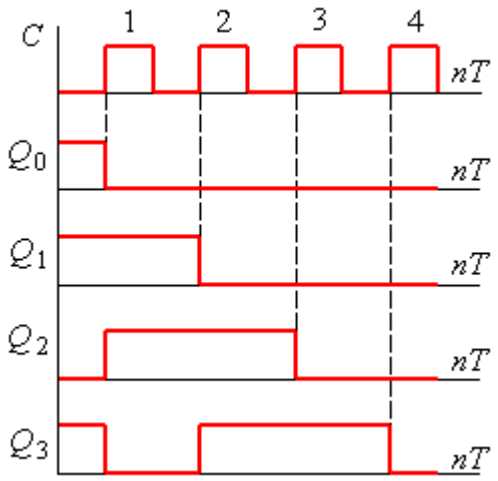


Рис. 15.8. Часова діаграма, яка ілюструє процедуру послідовного введення

Розглянемо особливості роботи регістра при послідовному введенні інформації. Припустимо, що в регістр послідовно вводиться, починаючи з молодшого розряду, двійковий код **1101**, який подається синхронно з тактовими імпульсами *C*-. Враховуючи, що тригери повинні спрацьовувати за зрізом вхідного сигналу, який подається на їх *C*-входи, а синхросигнал регістра інвертується елементом DD5.2, запис інформації забезпечуватиметься за фронтом синхросигналу регістра. При цьому записувана у тригери інформація повинна бути підготовленою до подачі синхросигналу.

За фронтом першого тактового імпульсу *C*- в DD0 буде записана одиниця молодшого розряду, а у решту тригерів перезапишуться нулі з прямих виходів попередніх тригерів (див. рис. 13.9). Перед приходом другого синхросигналу на входах DD0 і DD1 будуть присутні сигнали високого рівня, а на входах DD2 і DD3 – низького. Тому за фронтом другого синхросигналу в перший тригер DD0 знову перезапишеться **“1”**, в другий DD1 – також **“1”** з виходу Q_0, а в DD2 і DD3 будуть записані нулі відповідно з виходів Q_1 і Q_2. Аналогічно, за третім синхросигналом **“1”** з виходів Q_0 і Q_1 перемістяться на виходи тригерів DD1 і DD2, а в DD0 буде записаний **“0”** з *D*-входу регістра. Так за кожним синхросигналом у регістрі матиме місце зсув інформації на один розряд вправо. Після чотирьох тактових імпульсів код на виходах $Q_0 \ldots Q_3$ відповідатиме

вхідному коду і може бути зчитаний зовнішнім пристроєм. Розрядність регістра може бути підвищена нарощуванням подібних пристроїв. Нарощування здійснюється шляхом під єднання виходу Q_3 попереднього регістра з входом *D*-послідуючого з відповідним об'єднанням синхронізуючих та керуючих входів.

При наявності в регістрі *JK*-тригерів з асинхронними *R*- і *S*- входами можна забезпечити за допомогою допоміжної логіки (ЛЕ DD4) паралельний запис інформації $D_0 \ldots D_3$ у кожен з тригерів. Запис забезпечується асинхронно за низьким рівнем керуючого сигналу S/P (*Serial*/*Parallel*). Тобто записати інформацію в паралельному форматі можна в інтервалі часу між двома синхроімпульсами.

Зчитування інформації у послідовному форматі забезпечується подачею на *C*-вхід серії синхросигналів у кількості, що дорівнює розрядності регістра. Зі сказаного витікає, що регістр зсуву може виконувати функцію мультиплексора з запам'ятовуванням.

Для забезпечення зсуву інформації вліво необхідно в регістрі створити зв'язки, які могли б дозволити переміщувати інформацію з правого тригера в лівий, а також передбачити два послідовних входи: *DR* – вхід для прийому інформації, що поступає в регістр зі сторони молодшого розряду та зсувається вправо, та *DL* – вхід для прийому інформації, що поступає в регістр зі сторони молодшого розряду та зсувається вліво. Ці входи використовуються і при нарощуванні регістрів: *DR* приєднуються до входу старшого розряду сусідньої молодшої секції загального регістру; *DL* – приєднується до виходу Q_0 молодшого розряду сусідньої старшої секції загального регістра.

Функціональна схема одного розряду двонаправленого регістра зсуву з можливістю паралельного завантаженні приведена на рис. 15.9.

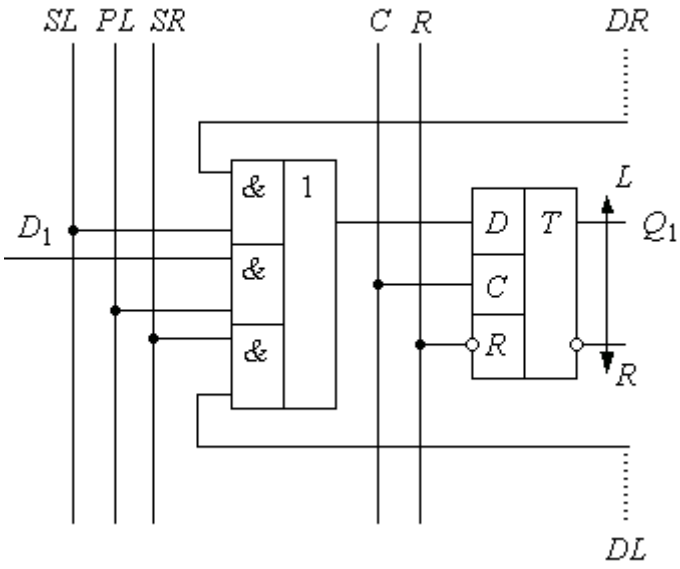


Рис. 15.9. Функціональна схема одного розряду двонаправленого регістра зсуву

До *D*-входу кожного тригера регістра приєднується вихід мультиплексора, який за допомогою керуючих сигналів *SL* (*Shift Left* – зсув ліворуч), *SR* (*Shift Right* – зсув праворуч) і *PL* (*Parallel Load* – паралельне завантаження) приєднує вхід *i*-го тригера відповідно до виходу послідуючого тригера, до виходу попереднього тригера або до *i*-го розряду паралельного завантаження. За сигналом синхронізації у тригер завантажуються відповідні дані. Вихід *i*-го тригера приєднується до відповідних входів мультиплексорів сусідніх розрядів (напрямки *L*, *R*). У мікросхемах регістрів входи *SL*, *PL*, *SR* не виводяться з корпусу мікросхеми, а керування ними здійснюється через дешифратор режимів.

Забезпечення обміну інформацією у послідовному форматі

Задача забезпечення обміну інформацією у послідовному форматі є досить актуальною, оскільки у цьому форматі передається більшість цифрової інформації.

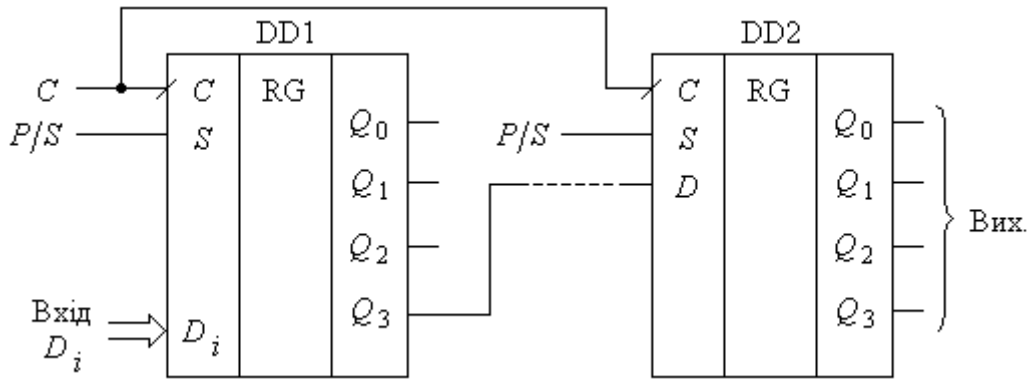


Рис. 15.10. Вхідний регістр

Вхідний регістр DD1 (рис. 15.10) забезпечує перетворення інформації, поданої в паралельному форматі, у послідовний формат і з виходу Q_3 передає її в лінію зв'язку. Вихідний регістр DD2 виконує зворотне перетворення. У регістр-приймач необхідно передавати не тільки послідовний код інформації, що передається, а також і синхроімпульси, необхідні для керування зсувом на приймальній стороні лінії зв'язку. Така система зв'язку є *однофазною трипровідною*, і їй властиві недоліки однофазних цифрових систем. Найсерйозніший недолік полягає у розфазуванні сигналів, що передаються.

Забезпечення стійкості до розфазування досягається різними шляхами. Один з них полягає в застосуванні вирівнюючого регістру, що вмикається перед входом регістра-приймача DD2. Особливість використання вирівнюючого регістра наступна. Якщо інформація, що передається, відновлюється за зрізом синхроімпульсу *C*-, то в якості вирівнюючого використовується регістр, що перемикається за фронтом, який виставляє на своєму виході інформацію за фронтом синхросигналу лінії зв'язку, тобто інформацію, яка з гарантією відноситься саме до даного такту. Можливі й інші засоби синхронізації.

Також використовуються й асинхронні способи послідовної передачі даних, в яких синхросигнал не передається по окремому провіднику, а за рахунок спеціального кодування передається по тому ж каналу, що й дані, розмежовуючись з ними у часі. Канал зв'язку спрощується за рахунок зниження швидкодії інформаційного обміну і ускладнення передавальної та приймальної

апаратури. Деякі аспекти такої передачі розглядались у попередньому розділі. Такий канал зв'язку є *двопровідним* (в практиці зв'язку називається *однопровідним*, оскільки один із провідників – загальна шина).

Використовуються різні способи такої однопровідної передачі. За першим з них синхронізуючий перепад супроводжує кожен біт інформації (*фазове* і *частотне кодування*). За іншим синхронізуючий перепад, який зветься стартовим бітом, супроводжує один байт. Це стандартний послідовний інтерфейс, який використовується у COM-портах ЕОМ, в USB-шині. У послідовному форматі забезпечується запис інформації на жорсткі носії. Для передачі та прийому інформації у послідовному форматі випускаються спеціальні мікросхеми. У більшості мікроконтролерів і мікропроцесорів для обробки інформації вмонтовується спеціальний пристрій – *UART* (*Universal Asynchronous Receiver / Transmitter*). Передача інформації у послідовному форматі з підвищенням швидкодії лінії зв'язку стає домінуючою в промислових системах автоматики.

Регістрові лічильники імпульсів (розподілювачі)

Розподілювачами називають функціональні вузли, які циклічно розподіляють потік імпульсів по декількох каналах [1]. Прикладом розподілювача може служити схема двійкового лічильника, на вхід якого подаються імпульси для підрахунку, а виходи приєднані до дешифратора. Збільшуваний або зменшуваний вихідний двійковий код лічильника дешифрується дешифратором по ряду каналів, створюючи в кожному з них послідовність імпульсів, що задається циклом роботи схеми. Аналіз такої схеми без урахування динамічних процесів показує, що створювані імпульсні послідовності можуть бути симетричними, якщо лічильник має модуль 2^m. Якщо ж модуль перерахунку не відповідає такій кратності, то не всі вихідні послідовності матимуть симетричний вид. При врахуванні динамічних режимів перемикання лічильника на виході дешифратора виникають короткі сплески напруги – гонки, які ще більше спотворюють вихідні послідовності.

Простіше функцію розподілювача може виконувати регістр зсуву, якщо лише в один його розряд помістити одиничний сигнал. При подачі синхроімпульсів записана одиниця переміщуватиметься по розрядах регістра, відображаючи на лінійній шкалі кількість поданих синхроімпульсів (наприклад, загоряння лампочки номеру поверху в ліфті). Значно більші можливості мають регістри зсуву, що з'єднані в кільце, а точніше, мають зворотний зв'язок з виходів на вхід. У літературі такі схеми, у залежності від характеру зворотних зв'язків, називають *кільцевими лічильниками*, *лічильниками Джонсона*, *лічильниками Мебіуса*, *лічильниками в коді Лібау-Крейга*, *кільцями Реженера*.

Загальна структура кільцевого лічильника з використанням регістра зсуву приведена на рис. 15.11.

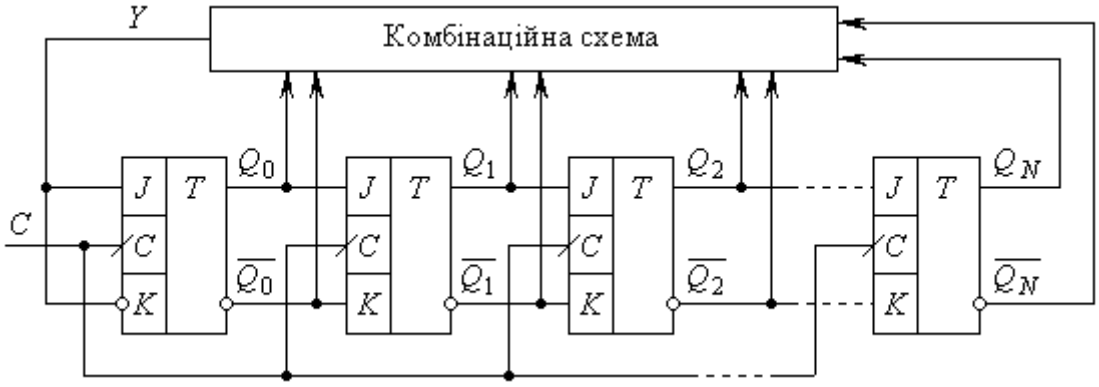


Рис. 15.11. Загальна структура кільцевого лічильника з використанням регістра зсуву

Особливість її полягає в тому, що допоміжно вона має комбінаційну схему, яка обробляє вихідні сигнали кожного тригера і генерує черговий сигнал на вхід регістра молодших розрядів:

$$Y = f(Q_0 \ldots Q_N, \overline{Q_0} \ldots \overline{Q_N}).$$

Тому можемо взаємозв'язок між входами/виходами кожного розряду зобразити у вигляді систем рівнянь:

$$Q_{0\,(n+1)} = f(Q_0 \ldots Q_N, \overline{Q_0} \ldots \overline{Q_N})_n ;$$
$$Q_{1\,(n+1)} = Q_{0n} ;$$
……
$$Q_{N\,(n+1)} = Q_{(N-1)\,n} .$$

Ланка зворотного зв’язку формує сигнал **0** або**1**, задаючи тим самим черговий стан схем лічильника.

Лічильники Джонсона

Недолік кільцевого лічильника, обумовлений великою кількістю тригерів, необхідних для реалізації необхідного коефіцієнта перерахунку, може бути усунений, якщо один із зв’язків між тригерами зробити перехресним, тобто вхід одного з тригерів з’єднати з інверсним виходом попереднього. При цьому принципово не має значення той факт, між якими з тригерів установлений перехресний зв’язок (*лічильник Мебіуса*).

Після установки всіх тригерів у нульовий стан на вході першого тригера матимемо сигнал логічної одиниці, який з кожним синхроімпульсом передаватиметься вправо, не зникаючи в попередньому розряді до заповнення всіх розрядів. За рахунок зворотного зв’язку у наступному циклі по лічильнику нарощуватиметься кількість нулів.

На рис. 15.12 приводиться приклад такого десяткового лічильника, що має п’ять тригерів.

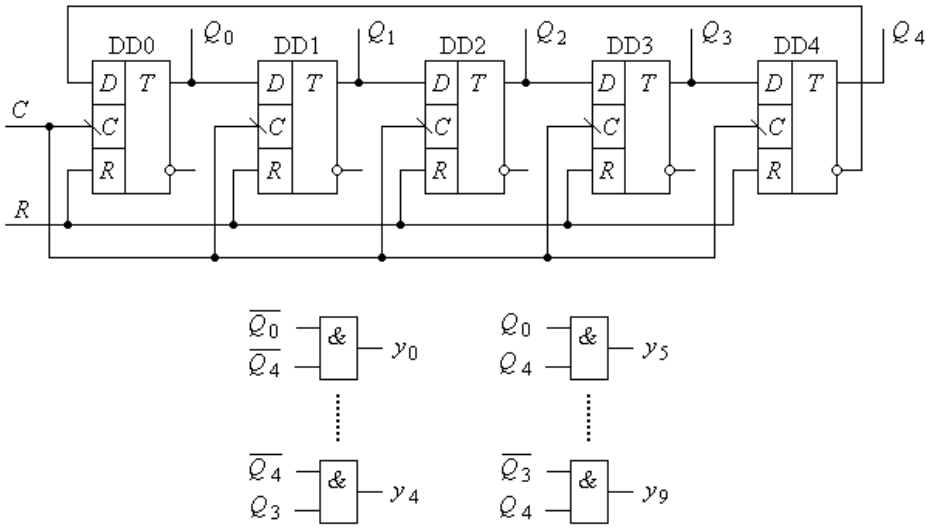


Рис. 15.12. Приклад десяткового лічильника Джонсона, що має п’ять тригерів

Прийнявши, що в нульовому стані всі тригери лічильника встановлюються в нуль, отримаємо послідовність відліку, що приводиться у табл. 15.2.

Таблиця 15.2. Послідовність відліку

N тактового імпульсу	Q_0	Q_1	Q_2	Q_3	Q_4	Спосіб дешифрації стану	*DO*
0	0	0	0	0	0	$\overline{Q_0}\cdot\overline{Q_4}=\overline{Q_0\vee Q_4}$	1
1	1	0	0	0	0	$Q_0\cdot\overline{Q_1}=\overline{\overline{Q_0}\vee Q_1}$	1
2	1	1	0	0	0	$Q_1\cdot\overline{Q_2}=\overline{\overline{Q_1}\vee Q_2}$	1
3	1	1	1	0	0	$Q_2\cdot\overline{Q_3}=\overline{\overline{Q_2}\vee Q_3}$	1
4	1	1	1	1	0	$Q_3\cdot\overline{Q_4}=\overline{\overline{Q_3}\vee Q_4}$	1
5	1	1	1	1	1	$Q_4\cdot Q_0=\overline{\overline{Q_4}\vee\overline{Q_0}}$	0
6	0	1	1	1	1	$\overline{Q_0}\cdot Q_1=\overline{Q_0\vee\overline{Q_1}}$	0
7	0	0	1	1	1	$\overline{Q_1}\cdot Q_2=\overline{Q_1\vee\overline{Q_2}}$	0
8	0	0	0	1	1	$\overline{Q_2}\cdot Q_3=\overline{Q_2\vee\overline{Q_3}}$	0
9	0	0	0	0	1	$\overline{Q_3}\cdot Q_4=\overline{Q_3\vee\overline{Q_4}}$	0

Отримана послідовність має 10 станів, які можна вибрати з табл. 15.2. Логічна схема ланки зворотного зв’язку *DO* може бути отримана таким чином: у стовпець таблиці *DO* заносяться необхідні значення сигналу *DO* для забезпечення послідуючого стану.

В результаті отримуємо:

$$DO=\bigvee_{0}^{9}0,1,2,3,4.$$

Переходячи до логічних змінних $Q_0\ldots Q_4$ й мінімізуючи їх за допомогою карт Карно (рис. 13.13), отримуємо:

$$DO=\overline{Q_4},$$

що підтверджує відповідність схеми рис. 15.12 і табл. 15.2.

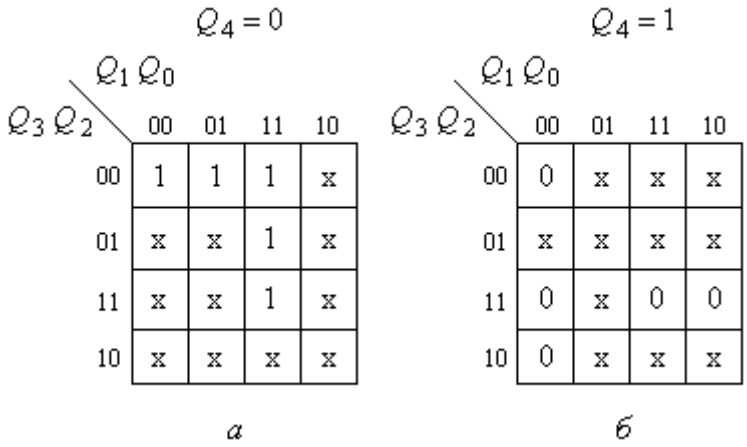


Рис. 15.13. Карти Карно для визначення і мінімізації логіки декодування станів

Організація десяткового відліку може бути забезпечена допоміжною логікою, яка призначена для декодування станів лічильника. Для визначення і мінімізації логіки декодування станів карти Карно, що зображені на рис. 15.13, дещо змінимо, вписавши до них номери станів (рис. 15.14).

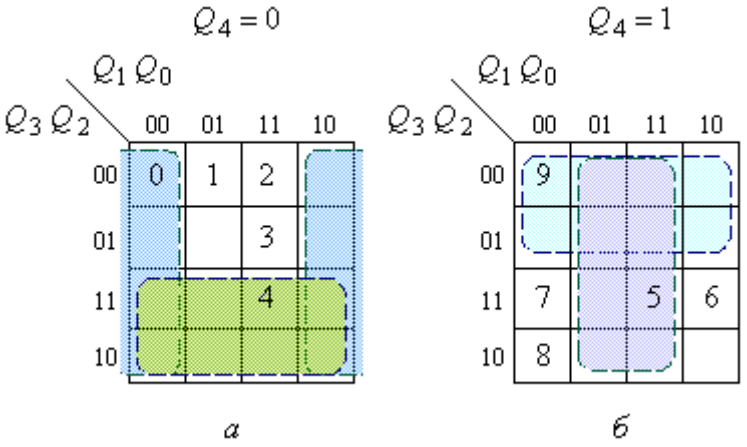


Рис. 15.14. Карти Карно зі вписаними номерами станів

Використовуючи відомі засоби мінімізації, для функцій, що залежить від Q_4, маємо:

$$y_0 = \overline{Q_0}\ \overline{Q_4}\ ; \qquad y_5 = Q_0\ Q_4\ ;$$
$$y_4 = Q_3\ \overline{Q_4}\ ; \qquad y_9 = \overline{Q_3}\ Q_4\ .$$

Аналогічно отримуються і функції, які не залежать від Q_4. Результати мінімізації представлені у табл. 15.2.

Для лічильника Джонсона, що розглядається, існують також три послідовності, відповідно до яких він може змінювати свої стани після дії перешкод:

1. $S_2 \to S_5 \to S_{11} \to S_{23} \to S_{14} \to S_{29} \to S_{26} \to S_{20} \to S_8 \to S_{17} \to S_2$;

2. $S_4 \to S_9 \to S_{19} \to S_6 \to S_{13} \to S_{27} \to S_{22} \to S_{12} \to S_{25} \to S_{18} \to S_4$;

3. $S_{10} \to S_{21} \to S_{10}$.

Вказані послідовності можуть бути відображені за допомогою карт Карно (рис. 15.15), цифри на яких відповідають вказаним послідовностям.

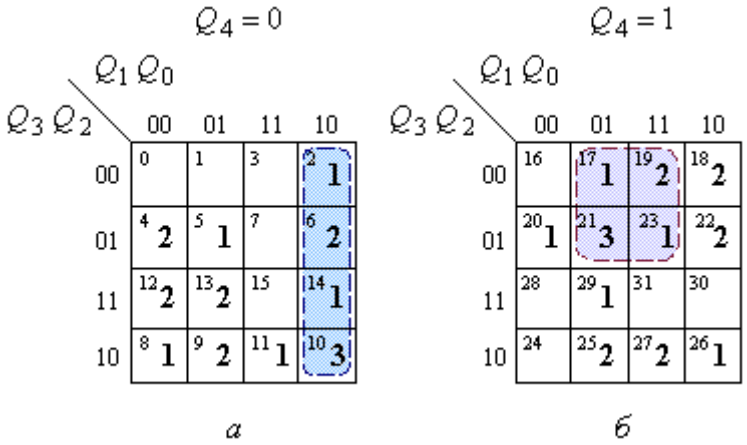


Рис. 15.15. Карти Карно

З карти, що приведена на рис. 15.15, ***а***, бачимо, що чотири суміжні клітини зі станами S_2, S_6, S_{14}, S_{10} охоплюють стани всіх небажаних послідовностей. Такою ж групою клітин на рис. 15.15, ***б*** є клітки з номерами S_{17}, S_{19}, S_{21}, S_{23}. Функції, що описують ці групи кліток, мають вигляд:

$$f_1 = \overline{Q_4}\, Q_1\, \overline{Q_0}\,; \qquad f_2 = Q_4\, \overline{Q_3}\, Q_0\,.$$

Є і інші варіанти таких функцій. Якщо використати одну з таких функцій для установки всіх тригерів, то через один цикл роботи лічильник відновить правильну послідовність зміни станів.

Системи контролю цифрової апаратури

Вузли ділення поліномів використовують також при контролі правильності роботи логічних схем методом сигнатурного аналізу [1]. На входи схеми, що перевіряється, з генератора псевдовипадкових чисел подається визначеної

довжини та завжди однакова послідовність псевдовипадкових чисел. Вихідна тестова послідовність проходить через поліноміальний розподілювач, який ділить її на достатньо складний поліном *G*. Поліноміальний розподілювач називають *сигнатурним аналізатором*. Залишок, який залишається в регістрі поліноміального розподілювача після закінчення вихідної тестової послідовності, називають *сигнатурою*. На виході справної цифрової системи сигнатура від заданої цифрової тестової послідовності завжди заздалегідь відома і може бути розпізнана, наприклад, за допомогою компаратора. Сигнатура несправної системи інша, причому, завдяки процедурі ділення та складному поліному *G*, будь-яка незначна відмінність значно посилюється. На теперішній час широке використання отримав шістнадцятирозрядний аналізатор. Частка помилок, котра ним не виявляється, оцінюється величиною 2^{-16}.

Сигнатурні аналізатори використовуються як при стендовому контролі цифрових систем, так і при створенні вбудованих пристроїв самоконтролю. В останньому випадку додатково застосовують так званий принцип *"наскрізного регістра"*, суть якого розкривається нижче.

Зупинимося коротко на ще одному методі тестової перевірки працездатності цифрових схем з використанням цифрових регістрів, який в вітчизняній практиці зветься *методом наскрізного зсувного регістра*, або *методом сканування*. Суть цього методу полягає у наступному. Для повного контролю складних цифрових схем необхідно мати доступ до великої кількості контрольних точок, які повинні бути виведені на роз'єми відповідних плат, або до виводів ВІС. Однак для цього повинна бути велика кількість виводів, що у більшості випадків забезпечити неможливо. У зв'язку з цим в якості контрольних точок зручно використовувати виводу тригерів. З'єднуючи всі тригери аналізованої схеми (а у ряді випадків – і додаткові зовнішні тригери) в один зсувний регістр, можливо послідовним кодом вводити тестову інформацію, що аналізує контрольні точки. Зазвичай для вирішення цієї задачі

використовується третій стан мікросхем, тригери відключаються від основної схеми та заводяться спеціальні серії синхросигналів. Перевірка працездатності наскрізного регістра здійснюється за допомогою генераторів псевдовипадкових послідовностей та сигнатурних аналізаторів.

Перелік посилань

1. Рябенький В.М., Жуйков В.Я., Гулий В.Д. Цифрова схемотехніка: Навч. посібник. – Львів: Новий світ – 2000, 2009, – 736 с.

Лекція №16

Постійні запам’ятовуючі пристрої

Розглянемо спочатку енергонезалежну пам’ять, яка здебільшого в найбільш загальній формі називається *ПЗП – постійні запам’ятовуючі пристрої* (***ROM*** – ***R****ead* ***O****nly* ***M****emory*) [1].

Одновимірні ПЗП. *ПЗП* – це пристрої пам’яті, вміст яких не може бути змінений процесором під час виконання програми і зберігається при відсутності живлення. Приклад структури ПЗП приводиться на рис. 16.1. Вона складається з дешифратора адреси $A_2\,A_1\,A_0$ і діодної матриці. У відповідності до значень сигналів на адресних входах, один з виходів дешифратора активізується (у даному випадку – на одному з виходів 0…7 з’являється сигнал високого рівня, який через відповідні діоди передається на виходи $D_0 \ldots D_3$, фактично відображаючи у двійковому коді значення слова, що створене в матриці за допомогою приєднаних діодів).

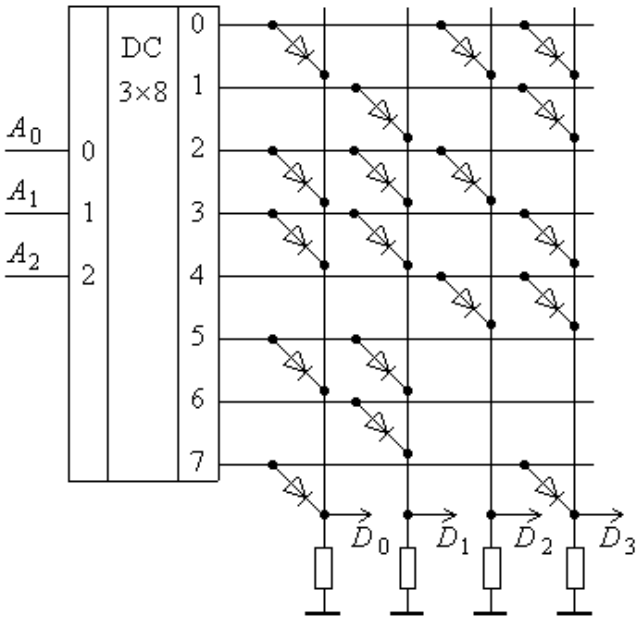


Рис. 16.1. Приклад структури ПЗП

Таблиця станів для кожного з виходів приведеної на рис. 16.1 схеми як функцію адресних входів приводиться в табл. 16.1.

Таблиця 16.1. Таблиця станів

N	Входи			Виходи			
	A_2	A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	1	1	0	1
1	0	0	1	1	0	1	0
2	0	1	0	0	1	1	1
3	0	1	1	1	0	1	1
4	1	0	0	1	1	0	0
5	1	0	1	0	0	1	1
6	1	1	0	0	0	1	0
7	1	1	1	1	0	0	1

З табл. 16.1 витікає, що ПЗП є комбінаційною схемою з n входами, які називаються *адресними*. Виходи комбінаційної схеми $D_0 \ldots D_3$ є виходами даних. Іншими словами, ПЗП зберігає в матричному пристрої пам'яті таблицю станів n-входової і m-виходової комбінаційної функції. Виходячи з даних таблиці, можемо стверджувати, що в ПЗП зберігається 2^n слів, кожне з яких має розмірність $m = 4$ розряди. Інформаційна ємність його складає $2^3 \times 4 = 32$ біти.

Нехтуючи прогнозованою затримкою, дані ПЗП визначаються бітами вихідного слова $D_0 \ldots D_m$, що зберігається у відповідному рядку таблиці станів, який задається адресними входами $A_0 \ldots A_n$. Умовне позначення ПЗП приводиться на рис. 16.2.

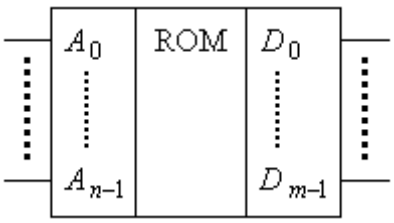


Рис. 16.2. Умовне позначення ПЗП

Оскільки ПЗП є комбінаційною схемою, то буде правильним сказати, що це взагалі не пам'ять, а група комбінаційних логічних елементів. Але, у той же час, якщо у ПЗП записується інформація, то зрозуміло, що вона звідти зчитуватиметься неодноразово і зберігатиметься навіть при відсутності

живлення. Це і мається на увазі, коли говорять, що ПЗП є енергонезалежною пам'яттю.

Аналізуючи стани табл. 16.1, можемо її розглядати як таблицю чотирьох функцій трьох змінних, тобто:

$$D_0 = f_0(A_0 \dots A_2); \qquad D_1 = f_1(A_0 \dots A_2);$$
$$D_2 = f_2(A_0 \dots A_2); \qquad D_3 = f_3(A_0 \dots A_2).$$

Така система функцій може бути реалізована на основі дешифратора 3 × 8 і чотирьох логічних елементів **АБО**, кількість входів кожного з яких відповідає кількості мінтермів відповідної логічної функції:

$$D_0 = \bigvee_0^7 0, 2, 3, 5, 7; \qquad D_1 = \bigvee_0^7 1, 2, 3, 5, 6;$$
$$D_2 = \bigvee_0^7 0, 2, 4; \qquad D_3 = \bigvee_0^7 0, 1, 3, 4, 7.$$

Таке представлення записаної у ПЗП інформації і забезпечує спосіб їх побудови на основі дешифраторів та багатовходових логічних елементів **АБО**.

З іншого боку, за кожною з адрес $A_2 \dots A_0$ маємо чотири біти даних, які можуть зчитуватись як окремі записані слова. Представивши їх у шістнадцятковій системі числення, табл. 16.1 можемо зобразити в іншій формі (див. табл. 16.2), тобто кожному значенню адреси, яка зображена у вигляді впорядкованого ряду мінтермів, відповідає конкретне значення вихідного слова, що записане в структурі пам'яті.

Таблиця 16.2. Таблиця (карта) прошивок

$A_2 \div A_0$	0	1	2	3	4	5	6	7
$D_3 \div D_0$	D	A	7	B	C	3	2	9

У практиці електронної техніки табл. 16.2 називається *таблицею* (*картою*) *прошивок* або *гексадецимальним лістінгом*.

Внутрішня структура ПЗП реально дещо відрізняється від приведеної на рис. 16.1 – перш за все, тим, що внутрішній дешифратор має інверсні виходи, а тому при активізації відповідного рядка його потенціал стає близьким до нуля (рис. 16.3). Кожен з виходів дешифратора називається *рядком слова*, оскільки

вибирається слово з матриці діодів. Вертикальні шини через резистори приєднані до джерела живлення. Вертикальні лінії називаються *бітовими стовпцями*, оскільки відповідають значенню одного біта записаного слова.

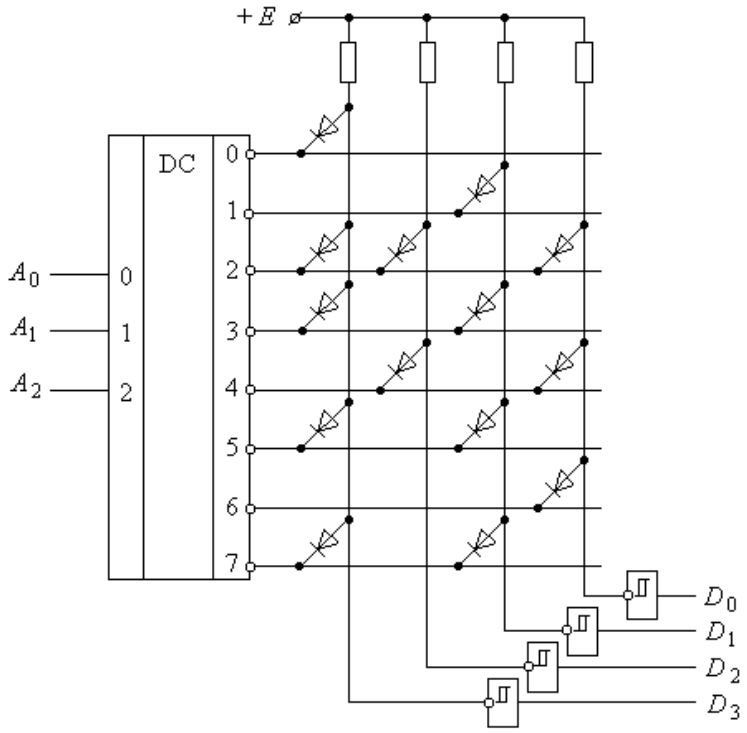


Рис. 16.3. Внутрішня структура ПЗ

У схемі, що розглядається, з’єднання через діод відповідних лінії слова і лінії біта забезпечує запис логічної одиниці.

Наявність інверторів з тригерами Шмідта покращує рівень захисту від перешкод, оскільки падіння напруги на діоді складає приблизно 0,5…0,6 В, і падіння напруги на відкритому виході дешифратора підвищують нульовий рівень перешкод.

Мікросхеми ПЗП виготовляються також із використанням МДН-технології, де в якості елементів пам’яті використовуються транзистори з n- або p- каналами, а також комплементарні ключі. При використанні МДН-транзисторів під час виготовлення ПЗП запис логічних одиниць або нулів забезпечується за рахунок величини порогової напруги стокозатворної характеристики транзистора. Збільшення або зменшення величини порогової напруги встановлюється товщиною ізоляційного прошарку між затвором та

напівпровідником. У матрицях з такими транзисторами всі транзистори, що відносяться до одного записаного слова, приєднуються затворами до вибраного рядка, а стоки кожного транзистора з'єднуються з відповідним розрядним стовпцем матриці. Витоки всіх транзисторів об'єднані і приєднуються до джерела живлення. При виборі відповідного рядка транзистори з низьким пороговим рівнем напруги відкриваються, і на відповідних стовпцях матриці отримується сигнал логічної одиниці. У транзисторах з високою порогової напругою буде записаний логічний нуль.

Двовимірне декодування в ПЗП. Припустимо, що нам необхідно побудувати ПЗП з організацією 128 × 1 за структурою, що була описана вище. Зрозуміло, що в такому випадку необхідно мати дешифратор **"з 7 на 128"**, котрий, як відомо, не так легко створити. Тому виникла інша ідея створення ПЗП такого типу – на основі діодної матриці з розділеною дешифрацією рядків та стовпців. Приклад її реалізації приводиться на рис. 16.4.

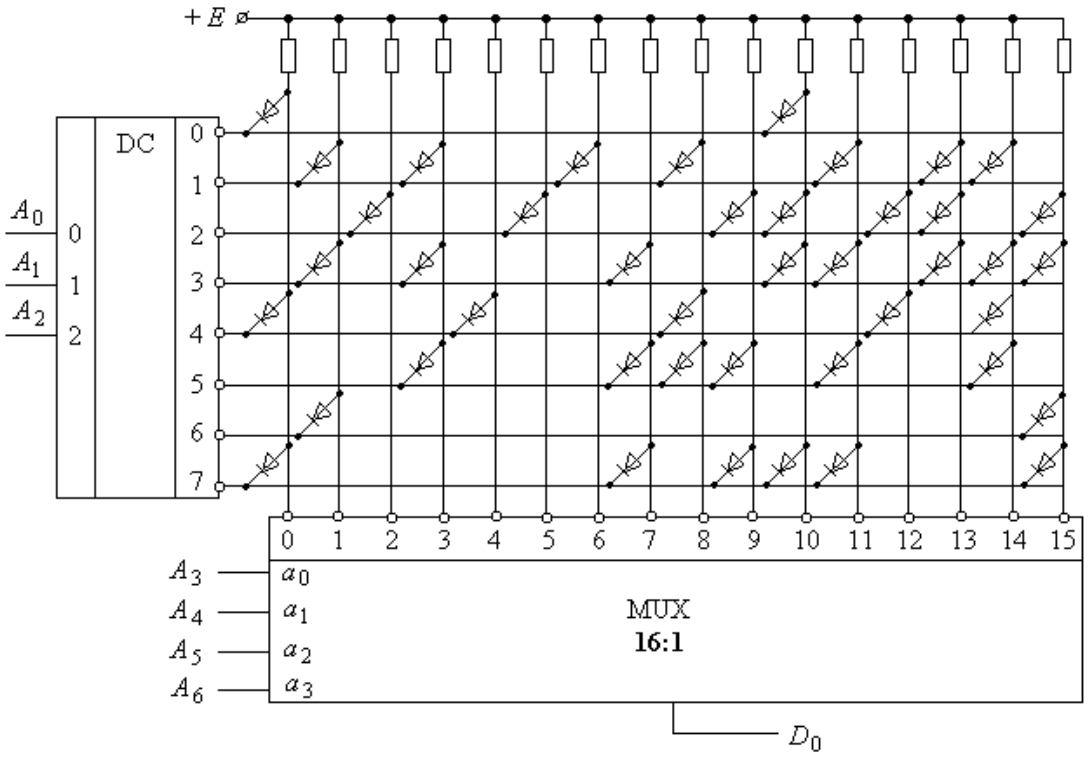


Рис. 16.4. Приклад реалізації діодної матриці з розділеною дешифрацією рядків та стовпців

Вся шина адресного простору розбивається у даному випадку на дві нерівні частини. Три старші розряди $A_6\,A_5\,A_4$ дешифруються у вісім рядків, а молодші розряди $A_3\,A_2\,A_1\,A_0$ використовуються для керування мультиплексором, що вибирає один з 16 бітових стовпців.

Така структура близька за формою до квадратної матриці, що забезпечує найбільш ефективне використання площі кристалу. Матрична структура побудови ПЗП відповідає ідеології фізичної реалізації на кристалах квадратної або майже квадратної форми.

У ПЗП з великою кількістю розрядів даних однобітна матриця може бути більш вузькою з тим, щоб структура всієї мікросхеми мала наближення до квадрата.

Як приклад, на рис. 16.5 приводиться можливий варіант ПЗП структури 64K × 8. Кожна матриця має 1024 рядки, які вибираються дешифратором старших розрядів адреси, і 64 стовпці, один з яких вибирається мультиплексором **64:1**. Звідси маємо 1024 × 64 = 65 536 слів з розрядністю 8 біт.

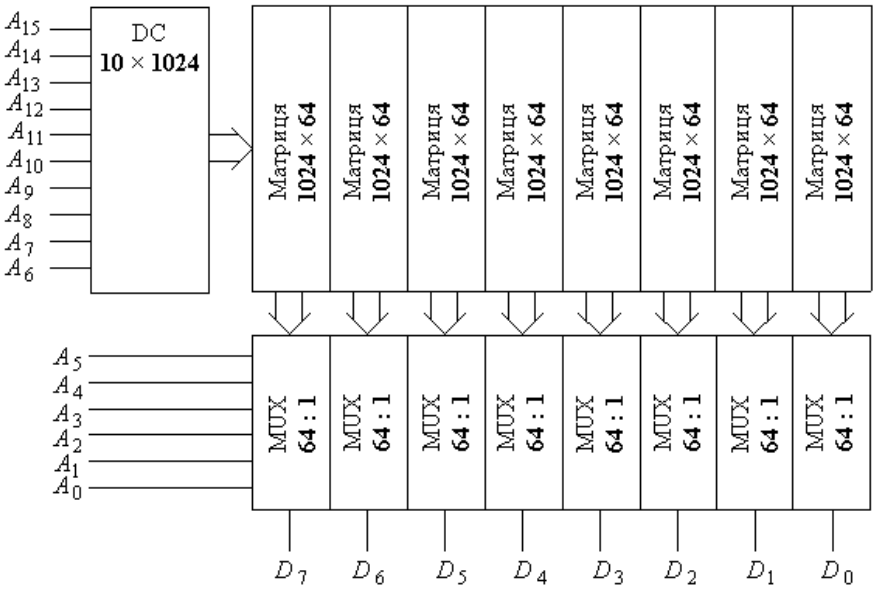


Рис. 16.5. Можливий варіант ПЗП структури 64K × 8

У більш загальному плані структура ПЗП має вигляд, приведений на рис. 16.6.

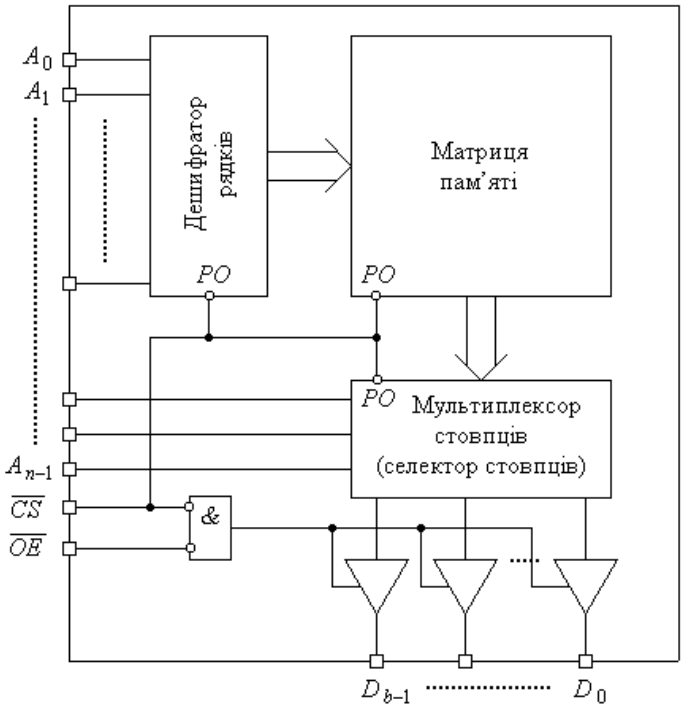


Рис. 16.6. Структура ПЗП

Більшість ПЗП, оскільки до мікропроцесорних систем вони приєднуються в великій кількості, мають вхід вибору мікросхеми $\overline{CS}$. Цей вхід керує процесом енергозбереження, адже при високому рівні сигналу на ньому внутрішні елементи ПЗП відключаються від живлення, і забезпечує переведення виходів шини даних у *Z*-стан. При $\overline{CS}=1$ енергоспоживання зменшується майже на 90%. Переведення виходів із *Z*-стану і виведення інформації з мікросхеми забезпечується входом дозволу виведення даних $\overline{OE}$ (***O**utput **E**nable*).

Репрограмовані ПЗП

Репрограмованими називають ПЗП, які є енергонезалежними приладами і інформація в яких може неодноразово перезаписуватись [1]. Ця група запам'ятовуючих пристроїв привертає до себе дуже велику увагу розробників, оскільки забезпечує високу надійність зберігання записаних у них даних. Тому вони мають високу динаміку розвитку і оновлення.

Принципи побудови репрограмованих ПЗП. Історично першими з них були розроблені *одноразово програмовані РПЗП* (***PROM** – **P**rogrammable **ROM***,

OTP ROM – One Time Programmable ROM) на основі запам'ятовуючих матриць, елементами пам'яті в яких використовувались перепалювані ніхромові перемички або p-n- переходи напівпровідникових приладів.

Сучасні РПЗП будуються на основі МОН-структур і мають значно більші можливості, оскільки інформація в них може бути замінена багаторазово. Вони розділяються на такі групи:

- *EPROM* (***Electrically Programmable ROM***) або РПЗП-УФ (репрограмовані ПЗП з ультрафіолетовим стиранням записаної інформації);
- *EEPROM* (***Electrically Erasable PROM***) або **РПЗП-ЕС** (репрограмовані ПЗП з електричним стиранням записаної інформації);
- пам'ять типу *flash* (флеш).

Особливість цих типів мікросхем полягає в тому, що запис інформації в них, який забезпечується електричними імпульсами, проходить значно повільніше, ніж читання, що є принциповою різницею між ними та пристроями оперативної пам'яті. Запам'ятовуючими елементами в них є спеціальні транзистори МОН-типу, в яких підзатворний ізоляційний діелектрик складається з двох шарів – тонкого (двоокис кремнію $Si\,O_2$) та товстого (нітрид кремнію $Si_3\,N_4$) (завдяки появі допоміжного шару на основі нітриду кремнію ці транзистори отримали назву МНОН).

Паралельні EEPROM. Як приклади паралельних EEPROM, можна привести мікросхеми сімейства AT28C фірми ATMEL.

Особливості приладів цього сімейства полягають у наступному:

- одна напруга живлення для всіх операцій;
- 8-розрядна організація;
- вбудовані фіксатори адреси та даних;
- вбудований автомат програмування;
- побайтове і секторне програмування;
- типова кількість циклів програмування – не менш 10000;

- виконання для побутової (діапазон робочих температур 0…+ 70° С) і промислової (– 40…+ 85° С) апаратури.

Сімейство 8-розрядних EEPROM AT28C має діапазон інформаційних ємностей від 16 Кбіт до 4 Мбіт. В усіх режимах прилади сімейства працюють від одного джерела живлення, що забезпечує просте внутрисистемне перепрограмування. Частина приладів забезпечують побайтове програмування як одного (кількох) байтів, так і всього обсягу основної пам'яті, інші мають секторну організацію і за одну операцію програмуються як в межах одного сектора, так і декількох байтів. Програмування керується вбудованим автоматом, не вимагає зовнішнього тактування і операцій попереднього стирання інформації. Звернення до пристроїв сімейства при операціях читання / запису аналогічне зверненню до статичних ОЗП. Мікросхеми містять вбудовані регістри на весь об'єм даних – байт або сторінку в залежності від типу приладу. На час циклу запису адреса і дані фіксуються вбудованими регістрами-фіксаторами, звільнюючи шину для інших операцій керуючого процесора. Закінчення циклу запису може бути виявлено опитуванням $\overline{DATA}$ і опитуванням біту чергування або за станом виводу $RDY/\overline{BUSY}$ (у залежності від типу приладу). Функції приладів сімейства розширені додатковими можливостями забезпечення високої якості та технологічності – застосована вбудована корекція помилок для підвищення стійкості приладів і підвищення характеристик схоронності даних. Додатковий механізм програмного захисту даних зберігає дані від випадкового перепрограмування. У приладах сімейства виділена особлива область пам'яті ємністю від 32 до 128 байт для зберігання користувальницьких даних ідентифікації.

Послідовні EEPROM. Серед мікросхем пам'яті все більшою популярністю користуються мікросхеми з послідовним протоколом запису та зчитування даних. Це мікросхеми з двохпровідною шиною типу IIC (I^2C) та з трьохпровідною шиною типу MICROWARE (SPI).

Таблиця 16. 3. Деякі типи мікросхем, що виготовляються фірмами ATMEL та MICROCHIP

Шина **I²C**			Шина **Microware (SPI)**		
Типи мікросхем, фірми		Організація	Типи мікросхем, фірми		Організація
ATMEL	MICROCHIP		ATMEL	MICROCHIP	
24C01	85C72 24C01A 24LC01		93C06		16 × 6
24C02 24LLC02	24C02	256 × 8	93C46	93C46 93LC46	64 × 16
24C04A 24LC04	24C04	2 × 256 × 8	93C46	93LC46	128 × 8
	24C08	4 × 256 × 8	93C56	93C56	128 × 16
24C16 24LC16	24C16	8 × 256 × 8	93C56	93C56 93LC56	256 × 8
			93C66	93C66 93LC66	256 × 16

У табл. 16.3 приведені деякі типи мікросхем, що виготовляються фірмами AMEL та MICROCHIP.

Основи побудови флеш пам’яті

Запам’ятовуючі пристрої, що мають назву *флеш-пам’яті* (*Flash memory*), побудовані подібно до EEPROM, але окремі архітектурні і експлуатаційні властивості обумовили необхідність її розгляду як окремої групи репрограмованих ПЗП. Головною особливістю флеш-пам’яті є те, що стирання записаної в неї інформації забезпечується не окремими словами (байтами), а в цілому для всієї мікросхеми або досить великих її розмірів [1]. Це дає можливість суттєво спростити схемотехніку, підвищити інформаційну ємність і швидкість.

Одночасне стирання всієї інформації РПЗП має і свої недоліки. Головний з них полягає в тому, що заміна навіть одного слова у флеш-пам’яті вимагає стирання і нового перезапису всієї інформації. Оскільки такий недолік не допустимий для великої кількості задач, то, поряд із флеш-пам’яттю з загальним стиранням (*BULK ERASE*) пам’яті, були розробленні мікросхеми на

основі блочної структури, в яких весь об’єм пам’яті ділиться на блоки, що стираються незалежно один від іншого.

У цілому розвиток архітектур флеш-пам’яті йде по двох основних напрямах. Перший з них розвиває мікросхеми пам’яті для зберігання даних і програм, які не дуже інтенсивно змінюються – наприклад, у мобільних телефонах, модемах, BIOS персональних комп’ютерів, системах керування автомобільними двигунами. Такі дані часто називають *параметричними*. Для цього напряму, в зв’язку з порівняно рідким оновленням вмісту, параметри циклів стирання та запису менш суттєві, порівняно з інформаційною ємністю та швидкістю зчитування. В пристроях пам’яті цього напряму використовують як загальне стирання, так і спеціальні архітектури з поблочним стиранням. Пам’ять з параметричними блоками використовується для зберігання телефонних номерів, обліку часу, використання у *SIM*-картках, для зберігання кодів помилок та параметрів оптимальних режимів роботи.

Серед мікросхем з поблочним стиранням слід виділити мікросхеми з спеціалізованими блоками, які призначені для зберігання програм ініціалізації системи, тобто програм, що вводять систему у робочій стан після вмикання живлення. Такі блоки називають *блоками завантаження* (*Boot Blocks*), а мікросхеми з такими блоками мають назву *Boot Block Flash Memory*. У *Boot Block* зберігаються також програми для програмування та стирання флеш-пам’яті.

Мікросхеми, що використовують *Boot Block*, здебільшого мають несиметричну структуру. Приблизний вигляд карти пам’яті, приведений на рис. 16.7 (для мікросхеми з ємністю 4 Мбіт).

3FFFFH 3E000H	**16K Byte Boot Block**
3DFFFH 3D000H	**8K Byte Parameter Block**
3CFFFH 3C000H	**8K Byte Parameter Block**

3BFFFH 30000H	**96K Byte Parameter Block**
2FFFFFH 20000H	**128K Byte Main Block**
1FFFFH 10000H	**128K Byte Main Block**
0FFFFH 00000H	**128K Byte Main Block**

Рис. 16.7. Приблизний вигляд карти пам’яті

Блок (*BLOCK*) – це область пам’яті з фіксованим діапазоном адресних кодів. При стиранні блоку пам’яті стираються всі елементи пам’яті всередині блоку. При цьому інформація в інших блоках залишається незмінною. Оскільки флеш-пам’ять не допускає перезапису окремих слів, а тим більше окремих біт без попереднього стирання всього блоку пам’яті, то для розв’язання вказаних задач використовуються програмні методи емуляції перезапису байта з використанням двох параметричних блоків ємністю по 8 Кбайт, які показані на рис. 16.7.

У *блоках параметрів* (*PARAMETER BLOCK*) зберігаються дані та програми, що змінюються відносно часто, – наприклад, діагностичні програми, коди ідентифікаторів і т. п. В *основних блоках* (*MAIN BLOCK*) зберігаються основні дані. Різниця між блоками полягає також у тому, що інформація, записана у *BOOT BLOCK*, апаратно захищена від випадкового стирання.

Окрім мікросхем з несиметричною картою пам’яті, виготовляються мікросхеми і з симетричним розміщеними *BOOT*-блоками на початку та в кінці адресного простору, що забезпечує симетричне зчитування даних як по зростанню адресного коду, так і по його зменшенню.

Другий напрямок розвитку флеш-пам’яті – це мікросхеми з великою ємністю ((64…128) М), які використовуються для зберігання великих обсягів інформації, тобто для заміни носіїв на магнітних дисках. Такі пристрої пам’яті містять у собі більш розвинені засоби перезапису інформації і побудовані на основі симетричних карт. До такого типу відноситься файлова система флеш-

пам’яті, яка в останні роки все більше замінює дискові запам’ятовуючі пристрої, оскільки має значно меншу споживану потужність, значно більшу надійність, значно більші швидкості зчитування інформації, зберігаючи при цьому повну сумісність з існуючими засобами керування пам’яттю. Блочна структура файлових пристроїв флеш-пам’яті є аналогом секторів магнітних дисків. Тому ідеологія обміну даними між секторами жорстких дисків дала можливість розробити програми, що забезпечують обмін між флеш-блоками. Оскільки організація обміну між оперативною пам’яттю та жорсткими дисками побудована на сторінковій основі, то для організації швидкого запису інформації у файлових флеш-пристроях передбаченні сторінкові буфери значної ємності, які дозволяють на високій швидкості сприймати деякі обсяги інформації, які потім перезаписується у блоки основної пам’яті з меншою швидкістю. Для файлових пристроїв пам’яті розроблені програми-драйвери, які дозволяють легко узгодити різноманітні апаратні засоби.

Ще одним напрямком розвитку флеш-пам’яті є мікросхеми, які називаються *Strata Flash* і зберігають в одному елементі пам’яті не один, а два біти. Нова якість обумовлена значним підвищенням точності інтегральних технологій, які дозволили у транзисторі з плаваючим затвором фіксувати не тільки наявність чи відсутність заряду, а й встановлювати його величину. На сучасному рівні розвитку технології чітко можуть фіксуватись чотири рівні заряду, що й забезпечує зберігання в одному елементі пам’яті два біти. Використовуючи багаторівневі компаратори та джерела еталонного струму можна визначити один з чотирьох рівнів струму транзистора, який описується двохрозрядним двійковим кодом у межах 00…11.

Оскільки зберігання двох бітів інформації отримані практично у тих же запам’ятовуючих пристроях, то це дало можливість удвічі збільшити інформаційну ємність флеш-пам’яті на одному кристалі.

Сучасні мікросхеми флеш-пам’яті виготовляються за технологією, яка називається *Smart Voltage*, що дозволяє використовувати єдину напругу

програмування і живлення модулів, забезпечуючи при цьому високу швидкодію мікросхем при зчитуванні інформації, високу надійність запису і найшвидший спосіб програмування. Ряд мікросхем, як приклад, при напрузі живлення 5 В забезпечують зчитування даних за 60 нс, а запис – за 13 мкс.

Оперативні запам'ятовуючі пристрої

Назва *"оперативні запам'ятовуючі пристрої"* (ОЗП) фактично відображає функціональне призначення такого роду пристроїв пам'яті, які призначенні для швидкого запису, зчитування та тимчасового зберігання обсягів інформації, які використовуються оперативно при виконанні програм [1]. В англомовній літературі початкова назва такої пам'яті була *Read/Write Memory* (*RWM*), тобто, знову ж таки, пам'ять для читання і запису, а, відповідно, і зберігання, в будь-який час. Більш пізня і сучасна її назва – *Random Access Memory* (*RAM*) (пам'ять з довільним доступом) –означає, що такі пристрої пам'яті мають можливість зчитувати або записувати біт інформації незалежно від його розміщення в пам'яті. З цієї точки зору *ROM* (*Read Only Memory*) (або ПЗП – постійні запам'ятовуючі пристрої) також відносяться до *RAM*. Але, незважаючи на таке непорозуміння, абревіатура *RAM* широко використовується як англомовна назва ОЗП.

У цілому ОЗП розділяють на дві групи – *статичні* та *динамічні*. Різниця між ОЗП кожної з груп досить суттєва, і вона проявляється у різноманітних характеристиках – від областей застосування до вартості. *Статичні ОЗП* в якості елементів пам'яті використовують *D*-тригери, інформація в яких при наявності напруги живлення зберігатиметься від запису до перезапису. *Динамічні ОЗП* в якості елементів пам'яті використовують конденсатори ємністю порядку 0,5 пФ. Тривалість зберігання інформації у статичних ОЗП необмежена. У динамічних вона обмежується часом саморозряду конденсаторів, що вимагає спеціальних засобів відновлення (регенерації) даних і додаткових витрат часу на цей процес. Звідси і витікає суттєва різниця як у схемотехніці обох типів пам'яті, так і в принципах роботи з нею.

Статичні ОЗП. Сучасні оперативні запам’ятовуючі пристрої статичного типу відносяться до пристроїв пам’яті високої вартості, і тому у мікропроцесорних системах використовуються досить обмежено. Основною перевагою динамічних ОЗП виступає висока швидкодія, що визначає їх використання у швидкодіючих системах обробки інформації. Як КЕШ-пам’ять вона широко використовується для суттєвого підвищення швидкості взаємодії процесора з основною (динамічною) пам’яттю у персональних комп’ютерах, а також є основною пам’яттю у потужних мультипроцесорних системах.

На сучасному рівні розвитку технології напівпровідникової техніки статичні ОЗП виготовляються на основі КМОН-технологій. Основою статичних ОЗП є елемент пам’яті, побудований на синхронних статичних *D*-тригерах.

Однобітні елементи пам’яті поєднуються в матрицю пам’яті, в якій керуючі сигнали кожного з елементів об’єднуються. Матриці пам’яті будуються на основі схемотехніки, подібної до репрограмованих запам’ятовуючих пристроїв. На рис. 16.8 як приклад статичного ОЗП приводиться блок-схема мікросхеми SRM2114 фірми SU WA SEIKOSHA (Японія) з організацією 1024 слова по 4 біти (4 Кбіт).

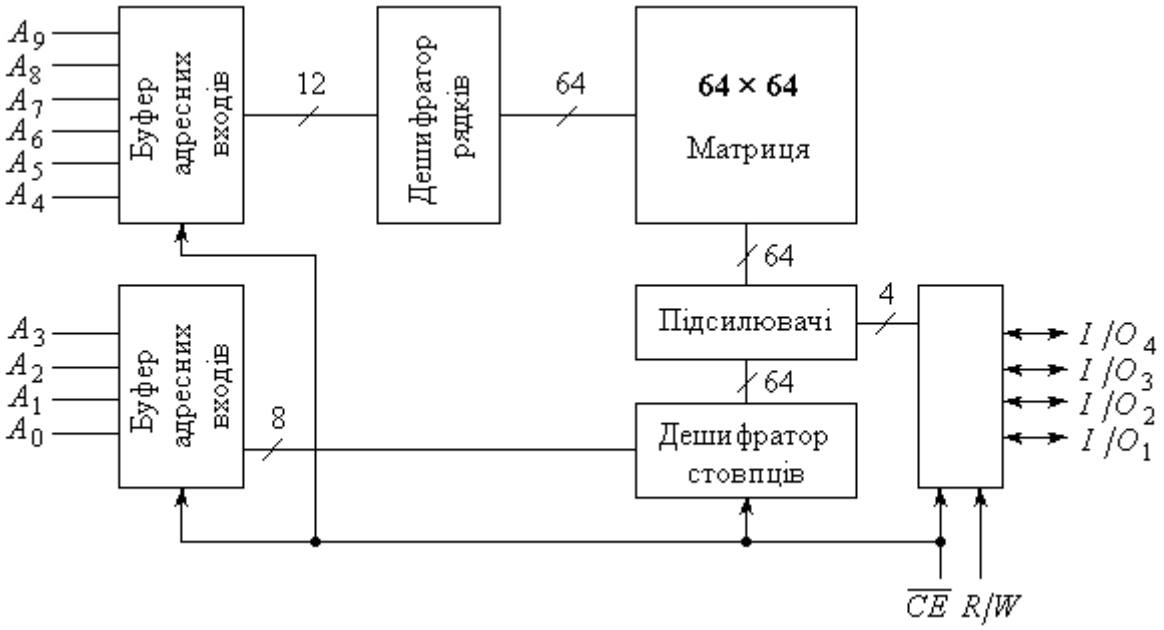


Рис. 16.8. Приклад статичного ОЗП

Порівняння приведеної блок-схеми ОЗП мікросхеми пам’яті з блок-схемою РПЗП показує їх значну схожість, але, незважаючи на це, ОЗП має ряд своїх особливостей при їх експлуатації, які будуть розглянуті нижче.

Умовне зображення ОЗП мікросхеми приводиться на рис. 16.9.

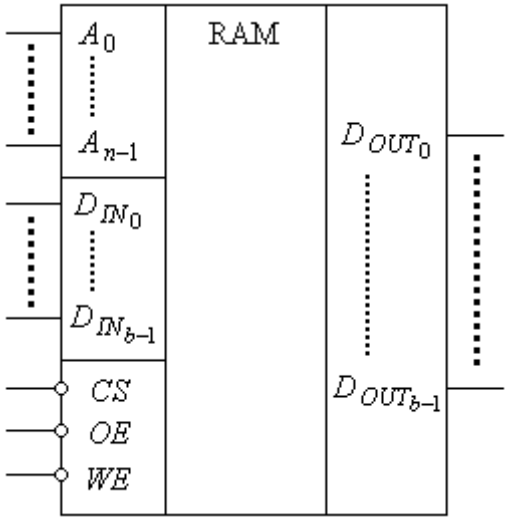


Рис. 16.9. Умовне зображення ОЗП мікросхеми

Для забезпечення операції зчитування необхідно забезпечити низькі рівні сигналів на входах $\overline{CS}$ та $\overline{OE}$ і за заданою адресою вибрати необхідні дані, які з матриці пам’яті будуть передані на виводи $D_{OUT\,0} \ldots D_{OUT\,(b-1)}$.

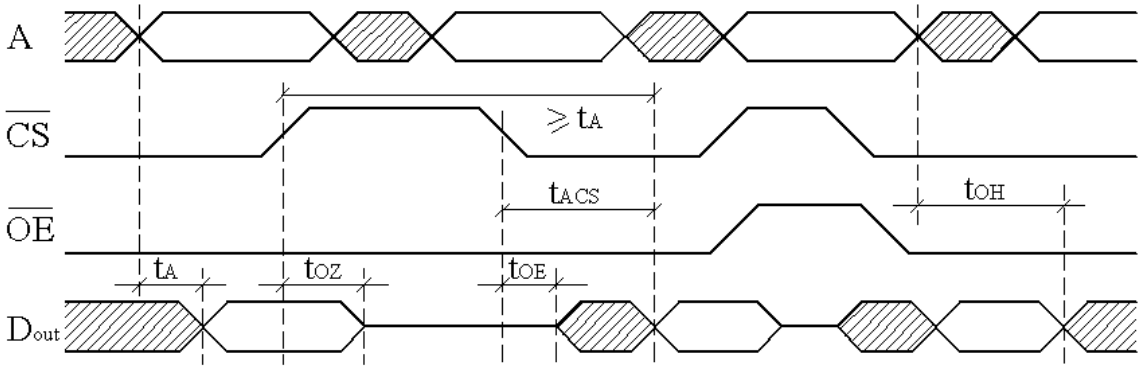
Для забезпечення операції запису вибирається необхідна адреса $A_0 \ldots A_{n-1}$, необхідна мікросхема $\overline{CS} = 0$, необхідні для зберігання дані подаються на входи $D_{IN\,0} \ldots D_{IN\,(b-1)}$ і за сигналом $\overline{WE} = 0$ заносяться у тригери матриці пам’яті.

Але для забезпечення надійного запису, так само як і зчитування, при роботі з ОЗП необхідно чітко дотримуватися послідовності подачі сигналів та часових співвідношень між ними. У загальному плані робота ОЗП описується таблицею станів табл. 16.4, яка характеризує статичні значення сигналів для кожного з режимів.

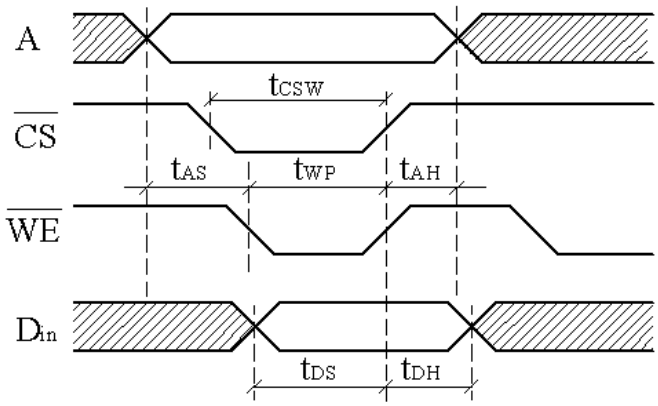
Таблиця 16.4. Таблиця станів

$\overline{CS}$	$\overline{OE}$	R/W	A	D_{IO}	Режим
1	x	x	x	Z	Зберігання
0	x	0	A	DI	Запис
0	0	1	A	DO	Зчитування

Часові співвідношення між сигналами задаються часовими діаграмами та відповідними часовими співвідношеннями, подібними до відповідних параметрів ПЗП (рис. 16.10).



а



б

Рис. 16.10. Часові діаграми ОЗП для режиму зчитування (а) та для режиму запису (б)

Часові параметри мають наступну інтерпретацію:

- t_A (*Access time from address*) – час доступу до даних після подачі установлених адресних сигналів (час перехідних процесів зображено

заштрихованими ділянками). Це мінімальний інтервал часу між установленими значеннями адресних сигналів до моменту початку зчитування даних за умови, що сигнали $\overline{CS}$ і $\overline{OE}$ підготовлені раніше. Цей інтервал часу мається на увазі, коли говорять про час доступу до даних конкретного типу статичної пам'яті;

- t_{ACS} (*Access time from chip select*) – час доступу до даних після подачі і установлення $\overline{CS}$. Це мінімальний інтервал часу між установленням сигналу $\overline{CS}$ і моментом початку зчитування даних при установлених попередньо адресних сигналах і сигналі $\overline{OE}$;
- t_{OE} (*Output enable time*) – час переходу вихідних підсилювачів із *Z*-стану в активний. Визначається як інтервал часу з моменту переходу одного з керуючих сигналів $\overline{CS}$ або $\overline{OE}$ в активний стан (низький рівень) до моменту появи даних (неустановлений режим) на вихідних шинах при установлених значеннях адресного сигналу;
- t_{OZ} (*Output disable time*) – час переходу вихідних підсилювачів з активного у *Z*-стан. Він визначається як інтервал часу між переходом одного з керуючих сигналів $\overline{CS}$ або $\overline{OE}$ у пасивний стан (високий рівень) та моментом, з якого виходи мікросхеми перейдуть у *Z*-стан при установлених значеннях адресного сигналу;
- t_{OH} (*Output hold time*) – час підтримки виходу. Цей параметр визначає, як довго можна знімати дані з виходів мікросхеми після зміни адресного коду;
- t_{AS} (*Address setup time before write*) – час установлення адресного коду перед записом. Цей інтервал часу важливий з точки зору надійного і непошкодженого запису даних за установленим адресним кодом. Визначається з моменту установлення коду адреси до моменту активізації останнього з керуючих сигналів $\overline{CS}$ або $\overline{WR}$. Рекомендована послідовність встановлення сигналів приведена на часовій діаграмі;
- t_{AH} (*Address hold time after write*) – час підтримки адреси після запису. Цей параметр аналогічний t_{AS} і визначає інтервал часу, протягом якого адресний

код повинен залишатися без зміни після переводу керуючих сигналів $\overline{CS}$, $\overline{OE}$ у пасивний стан (високий рівень);

- t_{CSW} (*Chip-select setup before end of write*) – час підтримки активного рівня керуючого входу $\overline{CS}$ при виконані операції запису;
- t_{WP} (*Write-pulse width*) – тривалість активного рівня керуючого входу $\overline{WE}$ для забезпечення надійного запису;
- t_{DS} (*Data setup time before end of write*) – інтервал часу, протягом якого всі сигнали повинні бути незмінними для забезпечення надійного запису;
- t_{DH} (*Data hold time after end of write*) – аналогічно t_{DS}, тобто дані на вході повинні залишатись незмінними по закінченню циклу запису.

Виробники статичних мікросхем оперативної пам’яті визначають два типи циклів запису: керований сигналом $\overline{WE}$ та керований сигналом $\overline{CS}$. Різниця між ними полягає лише у тому, який з сигналів активізується. При цьому несуттєво, який з цих сигналів першим переходить у пасивний стан, оскільки після цього рівень сигналу на іншому вході не має значення.

Стандартні мікросхеми статичної пам’яті мають ємність і організацію, подібні до ПЗП і РПЗП. Широко використовувані мікросхеми мають ємність до 4 Мбіт і мінімальний час доступу 2…3 нс.

Найширше використання статичні ОЗП знаходять у малих мікропроцесорних системах, вмонтованих системах керування, телефонії. Швидкодіючі статичні ОЗП використовуються як КЕШ-пам’ять у високопродуктивних обчислювальних машинах і системах.

Перелік посилань

1. Рябенький В.М., Жуйков В.Я., Гулий В.Д. Цифрова схемотехніка: Навч. посібник. – Львів: Новий світ – 2000, 2009, – 736 с.