МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
"ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ"

А. І. Поворознюк, О.А. Поворознюк

МЕТОДИ ТА ЗАСОБИ МОДЕЛЮВАННЯ СКЛАДНИХ ДИНАМІЧНИХ ОБ'ЄКТІВ

Навчальний посібник

для магістрів спеціальності 123 «Комп'ютерна інженерія»

Затверджено
редакційно - видавничою
радою НТУ «ХПІ»,
протокол № 2 від 27. 06. 2024.

Харків
НТУ «ХПІ»
2024

УДК 004.3(075)
П 42

Рецензенти: *О. Г. Аврунін*, д-р техн. наук, проф., завідувач каф. біомедичної інженерії Харківського національного університету радіоелектроніки,
О. А. Серков, д-р техн. наук, проф., професор кафедри «Системи інформації ім. В.О. Кравця» Національного технічного університету «Харківський політехнічний інститут»

Поворознюк А.І.
П48 Методи та засоби моделювання складних динамічних об'єктів: навчальний посібник. / А. І. Поворознюк, О. А. Поворознюк. – Харків: НТУ «ХПІ», 2024 – 378 с.

ISBN

Посібник дає системне уявлення про теоретичні основи та інформаційні технології побудови моделей складних динамічних об’єктів та вирішальних правил на вказаних моделях, а також апаратно-програмні засоби реалізації спеціалізованих середовищ. Розглянуто питання синтезу та дослідження моделей складних багаторівневих динамічних об’єктів, включаючи моделі біологічних об’єктів та архітектуру і методи синтезу технічних засобів спеціалізованих середовищ на основі цифрових процесорів обробки сигналів та ПЛІС-пристроїв.

Призначено для магістпів спеціальності 123 «Комп’ютерна інженерія» а також може бути корисним для розробників програмного забезпечення медичних діагностичних систем та моделювання складних динамічних біологічних систем.

Іл.108. Табл.41. Бібліогр. 24 назв.

УДК 004.3(075)

ISBN

ЗМІСТ

ВСТУП

Даний посібник є переробленим та доповненим виданням посібника Н. М. Бондіна, А. І. Поворознюк, О. М. Шеін. «Комп'ютеризація спеціалізованих середовищ» 2013 року. Зміна назви пов’язана з зміною назви відповідної дисципліни в ОПП.

Одним з основних напрямків прискорення науково-технічного прогресу є комп'ютеризація найважливіших галузей виробництва, науки, медицини та інших. Поряд із широким спектром універсальних комп'ютерних засобів, які використовуються в основному для розв'язання обчислювальних задач, для створення систем автоматичного управління, моніторингових та діагностичних систем та ін., застосовуються спеціалізовані процесори та інтегральні мікросхеми з логікою, що програмується (ПЛІС), на базі яких розробляються високоефективні спеціалізовані середовища.

Таким чином, для розробників спеціалізованих середовищ украй цінними є знання про теоретичні основи та інформаційні технології побудови моделей складних динамічних об’єктів та вирішальних правил на вказаних моделях, а також апаратно-програмні засоби реалізації спеціалізованих середовищ.

В даний час на кафедрі комп’ютерної інженерії та програмування НТУ «ХПІ» ведуться лекційні і лабораторні заняття з курсу «Методи та засоби моделювання складних динамічних об'єктів».

Структура і зміст посібника відповідає силабусу курсу і складається з чотирьох частин. Усі розділи містять контрольні питання, що можуть використовуватися як при самопідготовці, так і при контролі знань.

У першому розділі “Загальні принципи побудови моделей складних об’єктів та вирішальних правил на цих моделях” розглядаються основні принципи моделювання, класифікація моделей та етапи моделювання. При цьому більш детально розглянуто принципи аналітичного та імітаційного моделювання. Велике місце в першому розділі посідають методи формалізації та попередньої обробки множини вхідних та вихідних даних, методи аналізу структури даних, оцінки інформативності та формування інформативного простору ознак, а також методи синтезу вирішальних

правил. Автори посібника вважають, що тільки після освоєння цієї частини матеріалу в студентів буде сформоване правильне уявлення про необхідність і доцільність застосування методів побудови моделей складних динамічних об'єктів та синтезу технічних засобів спеціалізованих середовищ, представлених у наступних розділах.

У другому розділі «Приклади побудови моделей складних динамічних об'єктів» розглядаються методи синтезу моделей динамічних об'єктів, які описуються системою диференціальних рівнянь. Наведено числові методи розв'язання систем диференціальних рівнянь різних типів. Крім того, розглядаються приклади побудови моделей біосистем, а саме: моделювання розвитку популяцій; моделювання розвитку епідеміологічних процесів; моделювання в обчислювальній біології як завдання пошуку й проблема дешифрування ДНК.

У цьому ж розділі розглянуто особливості побудови моделей окремих органів і систем організму. Проаналізовано концептуальну модель систем організму, рівні взаємодії з оточуючим середовищем, часові й просторові границі роботи систем організму. Розкривається поняття гомеостазу.

Розглядаються приклади моделювання вуглеводного обміну в організмі й розвитку цукрового діабету, моделювання в імунології та моделювання серцево-судинної системи організму.

У третьому розділі "Цифрові процесори обробки сигналів – технічні засоби комп'ютеризації спеціалізованих середовищ" розглянута загальна характеристика цифрових процесорів обробки сигналів (ЦПОС) та область їх застосування, особливості архітектури та організація обчислень в ЦПОС, логічна та фізична організація основних модулів ЦПОС та їх взаємодія.

Розглянуто архітектурні особливості основних сімейств ЦПОС (Motorola, Texas Instruments, Analog Devices), та особливості їх програмування. Детально розглянуто архітектуру, особливості функціонування найбільш типових ЦПОС із фіксованою та плаваючою точкою, а також побудову спеціалізованих систем на їх основі.

Виклад матеріалу в четвертому розділі «Архітектура та методи синтезу ПЛІС пристроїв» заснований на розгляді технології програмування апаратних засобів. Наведено класифікацію спеціалізованих замовних та архітектуру базисних модулів структурованих ВІС. Розглянуто питання конфігурації логічних блоків, методів та засобів програмування архітектури ПЛІС.

Розділ 1, вступ та загальну редакцію виконав проф. Поворознюк А. І., доповнення та редагування розділів 2-4 – доц. Поворознюк О.А.

Автори висловлюють подяку рецензентам О. Г. Аврунінy та О. А. Серкову за цінні поради при підготовці даного видання.

Розділ 1. ЗАГАЛЬНІ ПРИНЦИПИ ПОБУДОВИ МОДЕЛЕЙ СКЛАДНИХ ОБ'ЄКТІВ ТА ВИРІШАЛЬНИХ ПРАВИЛ НА ЦИХ МОДЕЛЯХ

1.1. Сутність моделювання та задачі, що розв'язуються за допомогою моделювання складних об'єктів

1.1.1. Основні поняття моделювання

При дослідженні об'єктів для знаходження розумного рішення, пов'язаного з керуванням або оптимізацією, потрібно скласти математичну модель системи або процесу, що досліджуються.

Моделювання передбачає заміщення одного об'єкта, котрий є оригіналом, іншим, тобто його моделлю, і фіксацію чи вивчення властивостей оригіналу шляхом дослідження властивостей моделі [1]. Заміщення проводиться з метою спрощення, здешевлення, прискорення фіксації чи вивчення властивостей оригіналу.

Під об'єктом-оригіналом можна розуміти будь-яку природну чи штучну, реальну чи уявну систему. Вона має певну множину параметрів A_0 і характеризується певними властивостями. Множина параметрів системи і їх значень відображає її внутрішній зміст, тобто структуру і принципи функціонування чи існування. Кількісною мірою властивостей системи є множина характеристик Y_0. Система виявляє свої властивості під впливом зовнішніх вхідних дій X.

Характеристики системи знаходяться у функціональній залежності від її параметрів. Кожна характеристика системи $y_0 \in Y_0$ визначається в основному чи повністю обмеженою підмножиною параметрів $\{a_{0k}\} \subset A_0$. Інші параметри не впливають, чи практично не впливають, на значення даної характеристики системи. Дослідника, як правило, цікавлять тільки певні конкретні характеристики $\{y_{0k}\} \subset Y_0$ при конкретних зовнішніх діях $\{x_{0k}\} \subset X_0$.

Таким чином, модель є теж системою зі своїми множинами параметрів A_m і характеристик Y_m. Оригінал і модель схожі за одними параметрами і різні за іншими. Заміщення одного об'єкта іншим правомірне, коли характеристика оригіналу і моделі, котрі інтересують дослідника, визначаються однотипними підмножинами параметрів і пов'язані однаковими залежностями з цими параметрами. При однакових зовнішніх діях $\{x_{0n}\}$ за певний час t для оригіналу і моделі характерні залежності

$$y_{0k} = f(\{a_{01}\}, \{x_{0n}\}, t), \tag{1.1}$$

$$y_{mk} = f(\{a_{m1}\}, \{x_{mn}\}, t), \tag{1.2}$$

де y_{mk} – k-та характеристика моделі; x_{mn} – зовнішня дія на модель; t – модельний час, тобто час, на протязі котрого на модель здійснюються зовнішні дії $\{x_{mn}\} \subset X$ і вимірюються характеристики $\{y_{mn}\} \subset Y$.

За допомогою моделювання може розв'язуватись як пряма, так і обернена задача. У першому випадку множина характеристик моделі y_{mk} є відображенням множини характеристик оригіналу y_{ok}. У другому випадку при дослідженні складних природних систем склад елементів і принципи їх взаємодії мало вивчені, тобто відсутня достатня кількість відомостей про множину параметрів $\{a_{oi}\}$, і за допомогою моделювання розв'язується обернена задача. Будують певну модель, визначають її характеристики y_{mk} при еквівалентних зовнішніх діях $\{x_{mn}\}$, і якщо має місце відображення φ: $y_{ok} \rightarrow y_{mk}$ з певною відомою функцією φ, то вважають, що система-оригінал має такі ж параметри.

Теорія моделювання являє собою взаємозв'язану сукупність положень, визначень, методів і засобів створення та вивчення моделей. Ці положення, визначення, методи і засоби, як і самі моделі, є предметом теорії моделювання. Основна задача теорії моделювання полягає в тому, щоб дати в розпорядження дослідників технологію створення таких моделей, котрі б достатньо точно і повно фіксували властивості оригіналів, до яких виявлено інтерес, простіше і швидше піддавалися дослідженню і допускали перенесення його ресурсів на оригінали.

Теорія моделювання є основною складовою загальної теорії систем, де як головний принцип постулюються здійснювані моделі, і де система може бути зображена скінченною множиною моделей, кожна з котрих відображає певну грань її суті [1, 2].

Система – це сукупність взаємозв'язаних елементів, котрі об'єднані в одне ціле для досягнення певної мети. Під метою розуміють сукупність результатів, що визначаються призначенням системи. Наявність мети пов'язує елементи в систему.

Елемент системи – це мінімальний неподільний об'єкт системи. Елемент належить системі тому, що він зв'язаний з іншими елементами системи так, що множина елементів складає систему. Множину елементів, що складає систему, неможливо розбити на дві чи більше непов'язаних підмножин. Видалення елемента з системи обов'язково змінює її властивості в напрямку, відмінному від мети.

Складна система є множиною взаємозв'язаних і взаємодіючих між собою елементів і підсистем різної фізичної природи, що складають нероздільне ціле, котрі забезпечують виконання системою певної складної функції та описуються складною математичною моделлю.

Підсистема є сукупністю елементів, її поняття використовується в тому випадку, коли підсистема є достатньо самостійною частиною складної

системи, але мета її функціонування виходить із загальної мети функціонування системи. Розбиття складних систем на підсистеми називають декомпозицією систем. На сьогоднішній час цей процес неформалізований і носить евристичний характер.

Проста система – система, яка складається з малої кількості елементів, або модель якої можна віднести до розряду простих.

Реальні системи описуються шляхом визначення їх функцій і структур.

Функція системи – це правило одержання результатів, що визначається метою чи призначенням системи. Поведінку системи при цьому описують певною системою понять: відношень між змінними, векторами, множинами та ін. Функція встановлює, що робить система для досягнення поставленої мети і не визначає, як побудована система. Функціонувати – це означає реалізовувати функцію, тобто одержувати результати згідно з призначенням системи. Для опису функцій систем використовуються теорії множин, алгоритмів, випадкових процесів, інформації та ін.

Структура системи – це фіксована сукупність елементів і зв'язків між ними. В загальній теорії систем під структурою розуміють тільки множину зв'язків між елементами. Вона відображає тільки конфігурацію системи безвідносно до елементів, з яких вона складається. На практиці поняття "структура" містить не тільки множину зв'язків, але й множину елементів, між котрими існують зв'язки. Найбільш часто структуру системи відображають у вигляді графа, де елементи зображені вершинами графа, а зв'язки між ними – дугами.

Керування – процес збору, обробки і передачі інформації. За ступенем централізації керування системи розподіляють [2]:

- на централізовані системи керування, в котрих керування зосереджено в єдиному центрі;
- на децентралізовані системи, в котрих функція керування розподілена між головними і периферійними пристроями;
- на змішані системи.

Складні системи мають, як правило, ієрархічну структуру з кількома рівнями керування.

Визначення моделювання. При проектуванні, зокрема автоматизованому, розробник оперує не з самими об'єктами, а з їх моделями. Моделювання в даному випадку виступає і як апарат, і як засіб, за допомогою котрого створюється проект складної системи.

У широкому значенні під моделюванням розуміють процес адекватного відображення найбільш істотних сторін досліджуваного об'єкта чи явища з точністю, яка необхідна для практичних потреб. В загальному випадку моделюванням називають також особливу форму опосередкування, основою котрого є формалізований підхід до дослідження складної системи.

Теоретичною базою моделювання є теорія подібності.

Подібність – це взаємно однозначна відповідність між двома об'єктами, при котрій відомі функції переходу від параметрів одного об'єкта до параметрів іншого, а математичні описи цих об'єктів можуть бути перетворені в тотожні. Теорія подібності дає можливість встановити наявність подібності чи дозволяє розробити спосіб її одержання.

Таким чином, моделювання – це процес представлення об'єкта дослідження адекватною (подібною) йому моделлю і проведення експериментів з моделлю для одержання інформації про об'єкт дослідження. Під час моделювання модель є як засобом, так і об'єктом досліджень, що знаходиться у відношенні подібності до об'єкта, котрий моделюється.

Іншими словами, модель – це фізична чи абстрактна система, котра адекватно відображає собою об'єкт дослідження.

1.1.2. Класифікація моделей

В основу класифікації моделей може бути покладена ступінь абстрагування моделі від оригіналу. В цьому випадку всі моделі можна розділити на дві групи: матеріальні (або фізичні) і абстрактні (або математичні) [1, 2].

Фізичні моделі утворюються із сукупності матеріальних об'єктів. Для їх побудови використовуються різні фізичні властивості об'єктів, причому природа матеріальних елементів, що застосовуються в моделі, не обов'язково така ж, як і в об'єкті, котрий досліджується. Фізичною моделлю взагалі називають систему, котра еквівалентна чи подібна оригіналу, чи процес функціонування котрої такий же, як і оригіналу, і має ту ж фізичну природу. У свою чергу, фізичні моделі можна поділити на натурні, квазінатурні, масштабні та аналогові.

Натурні моделі – це реально досліджувані системи, їх називають макетами і дослідними зразками. Натурні моделі повністю адекватні системі-оригіналу. Це забезпечує високу точність і достовірність результатів моделювання. Зазвичай, випробуванням дослідних зразків завершується процес проектування складних пристроїв та систем.

Квазінатурні моделі – це сукупність натурних та математичних моделей. Цей вид моделей використовується тоді, коли математична модель як складова частини системи не є задовільною (наприклад, модель людини-оператора) чи коли одна складова системи повинна бути досліджена у взаємодії з іншими складовими, але їх на даний момент не існує, чи включення їх в модель ускладнене, або має велику вартість. Прикладами квазінатурних моделей є реальні АСУ, котрі досліджуються сумісно з математичними моделями відповідних виробництв.

Масштабні моделі – це системи тієї ж фізичної природи, що і оригінал,

але відрізняється від нього масштабами. Методологічною основою такого моделювання є теорія подібності, котра передбачає дотримання геометричної подібності оригіналу і моделі та відповідних масштабів для їх параметрів.

Аналогові моделі – це системи, котрі мають фізичну природу, що відрізняється від оригіналу, але подібні до оригіналу процеси функціонування. Обов'язковою умовою при цьому є однозначна відповідність між параметрами об'єкта і його моделі, а також тотожність безрозмірних математичних описів процесів, що протікають в них. Для створення аналогової моделі необхідний математичний опис системи, що вивчається. Як аналогові моделі використовуються механічні, гідравлічні та інші моделі, але найбільшого поширення набули електричні та електронні аналогові моделі. В них величина струму чи напруги є аналогами фізичних величин іншої природи.

Абстрактні моделі – це опис об'єкта дослідження на певній мові. Абстрактність моделі виявляється в тому, що її компонентами є певні поняття, наприклад, словесні описи, креслення, схеми, графіки, таблиці, алгоритми чи програми, математичні описи, а не фізичні елементи. Серед абстрактних моделей розрізняють гносеологічні, інформаційні, сенсуальні (чуттєві), концептуальні та інші.

Гносеологічні моделі направлені на вивчення об'єктивних законів природи, зокрема, наприклад, моделі сонячної системи, біосфери, світового океану, катастрофічних явищ природи та ін.

Інформаційні моделі описують поведінку об'єкта-оригінала, але не копіюють його (приклад – інформаційний опис конкретного мікропроцесора).

Сенсуальні моделі – моделі певних почуттів, емоцій, чи моделі, котрі виявляють дії на почуття людини (наприклад, музика, поезія, живопис).

Концептуальні моделі – це абстрактні моделі, що виявляють причинно-наслідкові зв'язки, котрі притаманні досліджуваному об'єкту. Ці зв'язки істотні в рамках певного дослідження. Основним призначенням концептуальної моделі є виявлення набору причинно-наслідкових зв'язків, врахування котрих необхідне для одержання потрібних результатів. Той самий об'єкт може бути представлений різними концептуальними моделями, котрі будуються залежно від мети дослідження. Наприклад, одна концептуальна модель може відображувати часові аспекти функціювання системи, а інша – вплив відмов на працездатність системи.

Математичні моделі – це абстрактні моделі, представлені на мові математичних відношень. Вони мають форму функціональних залежностей між параметрами, що враховуються відповідними концептуальними моделями. Ці залежності конкретизують причинно-наслідкові зв'язки, котрі виявлені в концептуальній моделі, і характеризують їх кількісно. Математична модель являє собою формалізований опис системи за

допомогою абстрактної мови, зокрема, за допомогою математичних співвідношень, котрі відображують процес функціонування системи. Для складання математичної моделі можна використати будь-які математичні засоби: алгебраїчне, диференціальне, інтегральне числення, теорію множин, теорію алгоритмів, теорію інформації та кодування та ін. Математичні моделі за методом їх дослідження та ознакою подальшого використання поділяють на аналітичні, чисельні, імітаційні, діагностичні та ін.

Аналітичні моделі – такий формалізований опис системи, котрий дозволяє одержати розв'язок рівняння в явному вигляді, використовуючи відомий математичний апарат.

Чисельні моделі – характеризуються залежністю такого виду, котрий допускає тільки часткові чисельні розв'язки для конкретних початкових умов і кількісних параметрів моделі.

Імітаційне моделювання в широкому розумінні являє собою моделювання з використанням комп'ютерів та змістовний опис об'єктів дослідження у формі алгоритмів [1]. Імітаційна модель – це модель, в котрій враховуються такі особливості, як наявність в самій системі елементів неперервної та дискретної дії, нелінійні співвідношення будь-якого характеру, які описують зв'язки між елементами, дію численних факторів складної фізичної природи. Засобами формалізованого опису імітаційних моделей в основному є універсальні і спеціальні алгоритмічні мови.

Для врахування певних особливостей і для визначення характерних рис оригіналу математичні моделі можуть поділятися на детерміновані та імовірнісні.

Детермінована модель – це модель, в котрій у заданий момент часу характеристики стану однозначно визначаються через вказані величини.

Імовірнісна (стохастична) модель – це модель, в котрій за допомогою математичних співвідношень можна визначити лише розподіл характеристик стану системи за заданими імовірнісними характеристиками її параметрів, вхідних сигналів, початкових умов.

Таким чином, модель – це спеціальний об'єкт, котрий в певних відношеннях заміщує оригінал. У принципі не існує моделі, котра була б повним еквівалентом оригіналу. Будь-яка модель відображає лише певні сторони оригіналу. Тому, з метою одержання найбільш повних знань про оригінал доводиться користуватися сукупністю моделей. Складність моделювання як процесу полягає у відповідному виборі такої сукупності моделей, котрі заміщують реальний пристрій чи систему в необхідних співвідношеннях.

1.1.3. Основні етапи моделювання

Для моделювання реальної системи необхідно створити відповідну модель та провести її дослідження. Перед створенням моделі необхідно конкретизувати мету, а після її дослідження провести аналіз результатів моделювання. Процес створення моделі здійснюється в кілька етапів. Він починається з вивчення системи S_0 і зовнішніх дій X і завершується розробкою чи вибором математичної моделі, або програми для обчислювальної системи, якщо моделювання проводиться за її допомогою.

Процес моделювання передбачає такі етапи:

- формулювання мети моделювання;
- вибір засобів моделювання;
- розробка концептуальної моделі;
- підготовка вихідних даних;
- розробка математичної моделі;
- вибір методу моделювання;
- розробка програмної моделі;
- перевірка адекватності та корегування моделі;
- планування машинних експериментів,
- комп'ютерне моделювання;
- аналіз результатів моделювання.

Трудомісткість кожного з етапів для різних систем може бути різною. У процесі моделювання конкретної системи можуть мати місце зміни технології моделювання. Наприклад, в процесі моделювання може бути наперед відомий метод моделювання чи його засоби. Математична модель може виявитись настільки простою, що не буде необхідності в машинних експериментах. Тобто, в процесі моделювання конкретного пристрою чи системи окремі етапи, в залежності від їх важливості, можуть вилучатися.

Розглянемо більш детально перелік задач, що вирішуються на кожному з відмічених етапів.

Формулювання мети моделювання. На цьому етапі повинно бути досягнуте повне розуміння між замовником і розробником моделі. Важливість коректного виконання цього етапу полягає в тому, що наступні етапи проводяться з орієнтацією на дану мету моделювання. На цьому ж етапі конкретизується, в якому вигляді (якісні чи чисельні градації, точність вимірювання та ін.) будуть представлені вихідні дані.

Перевірка адекватності моделі відмічена окремим етапом (див. далі), але адекватність моделі забезпечується якісним виконанням практично всіх етапів. Тому перевірка адекватності моделі повинна проводитись в тому чи іншому вигляді, починаючи з розробки концептуальної моделі і закінчуючи аналізом результатів моделювання.

Під розробкою математичної моделі розуміють створення повністю

формалізованого опису динаміки функціонування пристрою чи системи. Не завжди для цього можна підібрати відомий метод формалізації чи конструктивний математичний апарат. Але це слід намагатися зробити, тобто розробити однозначні залежності вихідних характеристик від параметрів і зовнішніх дій для кожної складової системи, алгоритми взаємодії між складовими, логічні умови зміни станів.

Результати машинного моделювання повинні бути проаналізовані з метою перевірки їх достовірності і вироблення рекомендацій про способи підвищення якості системи, що досліджується.

На всіх етапах моделювання слід звертати особливу увагу на документування рішень, що приймаються, допусків, обмежень і висновків.

Для тієї самої системи можна скласти множину моделей $\{S_m\}$. Моделі будуть відрізнятися ступенем деталізації і врахуванням тих чи інших особливостей і режимів функціонування, відображувати певну грань системи, орієнтуватися на дослідження певної властивості чи групи властивостей системи. Тому перед розробкою моделі необхідно сформулювати цілі дослідження. При створенні чи модернізації будь-якої системи постає задача визначення її ефективності. Якщо розробляється кілька варіантів системи, то з них необхідно вибрати найкращий. Вирішення цих задач і є основною метою моделювання. Взагалі в техніці використовується поняття техніко-економічної ефективності, котре враховує витрати і вимірювані характеристики системи:

$$E = E(Y_0) \tag{1.3}$$

Елементи множини характеристик $y_{0k} \in Y_0$ є частковими показниками якості системи: продуктивність, надійність, вартість, маса, габаритні розміри і т. д. Якщо функція (1.3) відома, тобто виражена аналітично, то показник ефективності E можна обчислити за множиною параметрів системи $\{a_0\}$ при певних зовнішніх діях $\{x_{0n}\}$. У противному разі використовується один з двох підходів: однокритеріальна чи багатокритеріальна оцінка.

При однокритеріальній оцінці обмежуються оцінкою ефективності системи за одним частковим показником якості y_0, а на інші характеристики накладають обмеження на їх допустимі зміни. При цьому можна одержати декілька варіантів систем з однаковим, чи приблизно однаковим, значенням y_0 при суттєво різних інших часткових показниках якості. В цьому випадку не можна з певною впевненістю визначити більш раціональний варіант.

При багатокритеріальній оцінці невідомий вид функції (1.3) штучно представляється у формі узагальненого чи інтегрального критерію. Однією з найбільш поширених форм представлення є адитивний критерій:

$$E = \sum_{i=1}^{n} b_i y_i \ , \tag{1.4}$$

де вагові коефіцієнти b_i узгоджують шкали вимірювань характеристик y_i та задовольняють умову

$$\sum_{i=1}^{n} b_i = 1, \quad \forall b_i > 0 \ . \tag{1.5}$$

Таким чином, визначення мети моделювання полягає, в першу чергу, у виявленні виду критерію ефективності E системи, що досліджується, а це, в свою чергу, передбачає задавання скінченої множини характеристик $\{y_i\}$, їх вагових коефіцієнтів і допустимих меж вимірювань.

Якщо метою моделювання є не просто фіксація властивостей системи, але й оптимізація системи, то перед моделюванням слід виявити ті параметри системи, котрі дослідник може змінювати.

Етап формулювання мети моделювання завершується оцінкою доцільності проведення моделювання. На цьому етапі необхідно зіставити витрати на розробку та модернізацію системи з економічним ефектом від її впровадження.

Вибір засобів моделювання та рівні моделювання. Методика моделювання безпосередньо залежить від рівня моделювання, тобто від ступеня деталізації опису об'єкта. Кожному рівню моделювання ставиться у відповідність певне поняття системи, елемента системи, закону функціонування елементів системи в цілому і зовнішніх дій. Наприклад, залежно від ступеня деталізації опису обчислювальних систем, їх пристроїв та елементів можна виділити три основних рівні моделювання:

1) рівень структурного чи імітаційного моделювання з використанням алгоритмічних моделей системи (моделюючих алгоритмів) із застосуванням спеціалізованих мов моделювання, теорій множин, алгоритмів, формальних граматик, графів, масового обслуговування, статистичного моделювання;

2) рівень логічного моделювання функціональних схем, елементів і вузлів обчислювальних систем, моделі котрих можна представити у вигляді рівнянь безпосередніх зв'язків (логічних рівнянь) і будувати з використанням апарату двозначної чи багатозначної алгебри логіки;

3) рівень кількісного моделювання (аналізу) принципових схем елементів обчислювальних систем, моделі котрих можна представити у виді систем нелінійних алгебраїчних рівнянь чи інтегро-диференціальних рівнянь і котрі можна досліджувати з використанням методів функціонального аналізу, теорії диференціальних рівнянь, математичної статистики.

Сукупність моделей об'єкта на структурному, логічному, кількісному

рівнях моделювання є ієрархічною системою, котра розкриває взаємозв'язок різних сторін опису об'єкта і забезпечує системну зв'язаність його елементів і властивостей на всіх стадіях процесу проектування обчислювальної системи. При переході на більш високий рівень абстрагування здійснюється згортка даних про об'єкт, що моделюється, а при переході до більш детального рівня опису – розгортка цих даних.

На структурному рівні моделюється взаємодія елементів об'єкта більш низького рівня. Топологічною моделлю в цьому випадку слугує орієнтований граф $\boldsymbol{G(V,D)}$, складання котрого базується на змістовому описі складу (множина вершин $\boldsymbol{V}$) і способу дії об'єкта (множина ребер $\boldsymbol{D}$). Вершинами орграфа v_i, є функціонально закінчені блоки об'єкта, а ребрами d_i – інформаційні зв'язки між ними.

Структурні відношення між елементами множини $\boldsymbol{V}$ описуються матрицею суміжності

$$[\boldsymbol{G_{ij}}]_v=[n\text{x}n], \tag{1.6}$$

рядки і стовпці котрої відповідають вершинам орграфа структурної моделі, а її $\boldsymbol{G_{ij}}$-й елемент дорівнює кількості ребер, напрямлених від вершини v_i до вершини v_j.

Відношення між елементами множини $\boldsymbol{V}$ і $\boldsymbol{D}$, тобто між вершинами і ребрами орграфа, описуються у вигляді булевої матриці інцидентності

$$[\boldsymbol{a_{ij}}]_{v,d}=[n\text{x}m], \tag{1.7}$$

рядки якої відповідають вершинам, а стовпці – ребрам орграфа; при цьому її $\boldsymbol{a_{ij}}$-й елемент дорівнює +1, якщо v_i – початкова вершина ребра d_i, і –1, якщо v_i – кінцева вершина d_i.

На логічному рівні моделювання кожній множині, булевій матриці бінарних відношень чи структурному графу відповідають набори логічних відношень між елементами, що в них входять, котрі представлені у вигляді логічних змінних. Множинам $\boldsymbol{V}$ і $\boldsymbol{D}$ також відповідають певні логічні відношення, які відображують причинно-наслідкові зв'язки. Ці зв'язки описують послідовності зміни станів об'єкта з врахуванням станів інших об'єктів і необов'язково суміжних з ним.

При кількісному моделюванні кожному елементу множини булевої матриці чи логічної змінної ставиться у відповідність алгебраїчна чи інша кількісна змінна, а логічні відношення переходять у кількісні відношення, наприклад, рівняння чи нерівності.

На кожному з основних рівнів моделювання можливі описи об'єктів з різним ступенем повноти і узагальнення за причиною існування різних ступенів деталізації структурних, логічних і кількісних властивостей і відношень. Але сама по собі задача побудови необхідної моделі, котра б

достатньо точно відображувала характерні властивості об'єкта чи його елемента (блока) на даному рівні проектування і в той же час була доступною для дослідження, є досить складною.

Розробка концептуальної моделі та технологія моделювання. У концептуальній моделі, зазвичай у словесній формі, наводяться відомості про природу і параметри елементарних явищ досліджуваної системи, про вид і ступінь взаємодії між ними, про місце і значення кожного елементарного явища в загальному процесі функціонування [2].

Спочатку концептуальна модель системи $\boldsymbol{S}_0$ виникає в свідомості дослідника. Модель орієнтується на виявлення певних властивостей системи відповідно до цілей моделювання. Для цього дослідник робить як би подумки зріз системи в "площині" тієї метасистеми $\boldsymbol{M}$, в котрій знаходиться система $\boldsymbol{S}_0$ (рис.1.3). Таку операцію називають $\boldsymbol{M}$-орієнтацією. Потім дослідник виявляє основні ознаки орієнтованої моделі і може додавати ще декотрі ознаки та умови, які полегшать дослідження моделі чи дозволять зобразити її у вигляді певного зрізу модельованої системи.

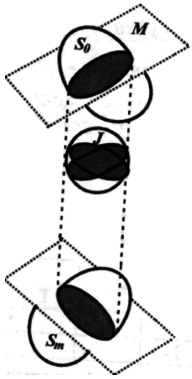


Рис. 1.1. Відображення оригіналу $\boldsymbol{S}_0$ і моделі $\boldsymbol{S}_m$ у свідомості дослідника

Розробка концептуальної моделі потребує достатньо глибоких знань про систему $\boldsymbol{S}_0$. Необхідно обґрунтувати те, що повинне ввійти в модель, а також те, що може бути відкинуте без істотних перекручувань результатів моделювання. Основною проблемою при створенні моделі є знаходження компромісу між простотою моделі та її адекватністю з досліджуваною системою. Розробник моделі, керуючись своїми знаннями системи, оціночними розрахунками, досвідом, приймає рішення про виключення того чи іншого елемента чи явища з моделі без достатньо повної впевненості у тому, що це не внесе істотних похибок в результати моделювання. Напевне, що результат створення концептуальної моделі ніколи не буде повністю формалізованим. Тому інколи кажуть, що моделювання є не тільки наукою, але й мистецтвом.

Наступним кроком при створенні концептуальної моделі є вибір рівня деталізації моделі. Проблема вибору рівня деталізації моделі може бути вирішена шляхом побудови ієрархічної послідовності моделей. На кожному рівні існують характерні особливості системи, змінні, принципи і залежності, за допомогою котрих описується поведінка системи.

Рівні деталізації інколи називають стратами, а процес виділення рівнів – стратифікацією. Вибір страт залежить від цілей моделювання і ступеня попереднього знання елементів.

При побудові стратифікованої концептуальної моделі в неї повинні, в першу чергу, увійти параметри $\{\boldsymbol{S}_{0j}\}$, які забезпечують визначення характеристик, що інтересують дослідника $\boldsymbol{Y}_{0k}$ при конкретних зовнішніх діях $\{\boldsymbol{X}_{0n}\}$ на заданому інтервалі часу t функціонування системи.

Деталізація системи повинна проводитись до того рівня, щоб для кожного елемента були відомі чи могли бути одержані залежності параметрів реакцій елемента, котрі істотні для функціонування системи, а також визначення залежності характеристик системи від параметрів дії, які є вихідними для цього елемента. Якщо за результатами орієнтації, стратифікації і розчленування одержуємо модель великої розмірності, тобто з великою кількістю параметрів, великим числом елементів (сотні і тисячі), то її необхідно спростити. Це можна зробити перетвореннями моделі без зниження ступеня адекватності, у тому числі шляхом декомпозиції системи на підсистеми, інтеграції елементарних операцій і відповідної інтеграції елементів, виключення другорядних технологічних процесів з виключенням елементів, котрі їх забезпечують.

Наступним кроком створення концептуальної моделі є її локалізація, котра здійснюється шляхом представлення зовнішнього середовища у вигляді генераторів зовнішніх дій, які включаються в склад моделі як елементи. При необхідності вони диференціюються на генератори робочого навантаження, котрі подають на вхід системи основні вихідні об'єкти – дані для інформаційних систем, у тому числі для ОС; генератори додаткових об'єктів і енергії; генератори керуючих і активізуючих дій (рис. 1.2.).

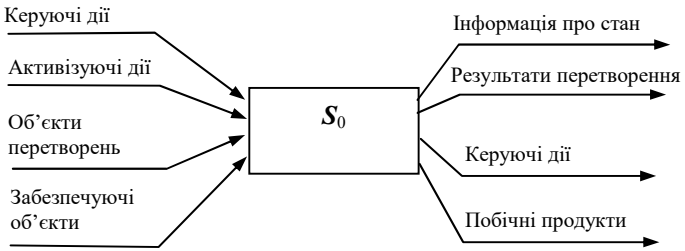


Рис. 1.2. Представлення процесу локалізації моделі

Структуризація та керування. Побудова структури концептуальної моделі завершується показом зв'язків між елементами. Зв'язки розподіляються на речовинні та інформаційні. Речовинні зв'язки відображують можливі шляхи переміщення продукту перетворення від

одного елемента до іншого. Інформаційні зв'язки забезпечують передачу керуючих дій між елементами та інформацію про стан. У концептуальній моделі повинні бути конкретизовані всі правила чи алгоритми керування робочим навантаженням, елементами і процесами.

Виділення процесів. У разі необхідності розгляду динаміки системи модель, що відображає статику системи, доповнюють описом функціонування системи. Функціонування системи полягає у виконанні технологічного процесу перетворення речовини, енергії чи інформації. У складних системах одночасно може протікати кілька технологічних процесів. Як приклад, можна згадати мультипрограмний режим сучасних обчислювальних систем. Технологічний процес являє собою певну послідовність окремих елементарних операцій. Частина операцій може виконуватися паралельно різними елементами системи. Задається технологічний процес маршрутною картою, програмою та ін.

Відображення станів. Для систем зі структурним принципом керування набув поширення інший підхід. Для кожного елемента вибирається певний параметр n, значення котрого змінюється в ході функціонування елемента, і він відображає його стан в поточний момент часу $z(t)$. Множина таких параметрів по всіх $n = \overline{1,N}$, елементах системи $\{z_n\}$ відображає стан системи $Z(t)$. Функціонування системи представляється у вигляді послідовної зміни станів: $Z(t_0)$, $Z(t_1)$,..., $Z(t_m)$. Множину $\{Z\}$ можливих станів системи називають простором станів. Поточний стан системи в момент часу t,($t_0 < t < t_m$) відображається у вигляді координати точки в m-вимірному просторі станів, а вся реалізація процесу функціонування системи за час t_m=T – у вигляді певної траєкторії.

Якщо відомий початковий стан системи $Z(t_0)$, то можна визначити її стан в будь-який момент t, коли відома залежність

$$Z(t) = F(X, Z_0, Z, t) . \qquad (1.8)$$

Тоді характеристики на виходах системи визначатимуться як

$$Y=W(Z,T). \qquad (1.9)$$

Підготовка вихідних даних. При створенні концептуальної моделі виявляються якісні (функціональні) і кількісні параметри системи $\boldsymbol{S}_0$, а також зовнішні дії. Для кількісних параметрів необхідно визначити їх конкретні значення, котрі будуть використані у вигляді вихідних даних при моделюванні. Це відповідальний і трудомісткий етап роботи. Достовірність результатів моделювання однозначно залежить від точності і повноти вихідних даних. Частина параметрів виявляється ще на ранній стадії концептуального моделювання, інші – паралельно з розробкою концептуальної моделі.

Збір вихідних даних ускладнюється через те, що значення параметрів

можуть бути не тільки детермінованими, але й стохастичними. Не всі параметри є стаціонарними. Особливо це стосується параметрів зовнішніх дій. По-друге, мова йде про моделювання неіснуючої системи (система ще тільки проектується, або модернізується), котра повинна функціонувати в нових умовах.

Взагалі більша частина параметрів є випадковими величинами за своєю природою. Але з метою спрощення моделювання вони в більшості випадків представляються детермінованими середніми значеннями. Це можна робити тільки тоді, коли випадкова величина має невеликий розкид. Заміна імовірнісних величин детермінованими повинна здійснюватись обґрунтовано, щоб це не призвело до зміщення середніх значень.

При створенні моделі детерміновані параметри можуть також замінюватись випадковими. Це робиться при інтеграції елементів системи чи зовнішніх дій з метою скорочення розмірності моделі.

Для випадкових параметрів організується збір статистики і наступна обробка. У процесі обробки виявляється можливість представлення параметрів певними теоретичними законами розподілу. Процедура підбору виду закону розподілу виконується наступним чином. За сукупністю чисельних значень параметра будується гістограма відносних частот – емпірична густина розподілу. Гістограма апроксимується плавною кривою. Одержана крива послідовно порівнюється з кривими густини розподілу різних теоретичних законів розподілу. Обирається один із законів за найкращим збіганням виду порівнюваних кривих. За емпіричними значеннями обчислюють параметри цього розподілу. Потім виконують кількісну оцінку ступеня збігання емпіричного і теоретичного розподілів за тим чи іншим критерієм погодження, наприклад, Пірсона, Колмогорова, Смірнова, Фішера чи Стьюдента. Питання підбору виду закону розподілу детально описує математична статистика [1].

Апроксимація функцій. Для кожного елемента системи існує функціональний зв'язок між параметрами вхідних дій і його характеристиками. Вид функціональної залежності для одних елементів є очевидним, а для інших він може бути вияснений з природи функціонування. Але для певних елементів може бути одержана тільки сукупність експериментальних даних про кількісні значення вихідних характеристик при різних значеннях параметрів. У цьому випадку виникає необхідність ввести певну гіпотезу про характер функціональної залежності, тобто апроксимувати її певним математичним рівнянням. Пошук математичних залежностей між двома і більше змінними за зібраними дослідними даними може виконуватись за допомогою методів регресійного, кореляційного чи дисперсійного аналізу [1].

Висування гіпотез. Щодо параметрів, котрі відображають нові елементи

майбутньої системи чи нові умови функціонування, відсутня можливість збору фактичних даних. Для таких параметрів висуваються гіпотези про їх можливі значення. Важливо, щоб гіпотези висували експерти-спеціалісти, котрі достатньо добре уявляють створювану систему чи нові зовнішні дії на систему. Ступінь суб'єктивності зменшується при одержанні відомостей від групи спеціалістів і при застосуванні методик експертних оцінок.

Закінчується етап збору і оцінки вихідних даних їхньою класифікацією на зовнішні і внутрішні, постійні і змінні, неперервні і дискретні, лінійні і нелінійні, стаціонарні і нестаціонарні, детерміновані і недетерміновані (стохастичні). Для змінних кількісних параметрів, котрими можна варіювати в ході моделювання, визначаються межі їх змін, а для дискретних – можливі значення.

Розробка математичної моделі. Концептуальна модель і кількісні вихідні дані є основою для розробки математичної моделі. Створення математичної моделі має дві основні мети:

1) дати формалізований опис структури та процесу функціонування системи для однозначності їх розуміння;

2) намагатися представити процес функціонування у вигляді, що допускає аналітичне дослідження системи.

Розробка єдиної методики створення математичних моделей до цього часу не уявляється можливою. Це обумовлено великою різноманітністю класів систем (статичні, динамічні, із структурним чи програмним керуванням, з постійною чи змінною структурою і т. ін.). За характером вхідних дій і внутрішніх станів системи підрозділяються на неперервні та дискретні, лінійні і нелінійні, стаціонарні і нестаціонарні, детерміновані і стохастичні. Відповідно існує така ж кількість типів моделей, і їх вибір залежить від ступеня стратифікації і деталізації.

Для певних класів систем розроблено формалізовані схеми і математичні методи, котрі дозволяють описати функціонування системи, а в деяких випадках і виконувати аналітичні дослідження. Засобами формального опису процесів функціонування систем з програмним керуванням є певні мови і системи імітаційного моделювання.

Для опису стохастичних систем з дискретними множинами станів, вхідних і вихідний дій, що функціонують в неперервному часі, широко застосовуються стохастичні мережі. Стохастична мережа є сукупністю систем масового обслуговування, в котрій циркулюють заявки, що переходять з однієї системи в іншу.

Велика група мов імітаційного моделювання ґрунтується на формалізованому представленні систем у вигляді стохастичних мереж. За певних умов стохастична мережа може розглядатись як сукупність незалежних систем масового обслуговування. Це дає додаткові можливості

використання досягнень теорії масового обслуговування для проведення аналітичного моделювання.

В основі систем масового обслуговування лежить поняття приладу, котрий може виконувати скінченну множину операцій. Прилад виконує операцію, коли виникає заявка – вимога на виконання операції. Якщо прилад виконує будь-яку операцію, то вважається, що він зайнятий, у противному разі – прилад вільний. Часова послідовність заявок називається потоком заявок. Загальний потік заявок може складатися з кількох потоків. У випадках незалежності потоків, випадкових моментів надходження чи завершення обслуговування заявок в системі можуть виникати черги. Черга – це заявки, котрі чекають обслуговування, коли прилад зайнятий. Прилад може складатися з кількох каналів, кожний з яких здатний обслуговувати будь-яку заявку. Сукупність приладу, потоків заявок і черг до нього називають системою масового обслуговування.

Теорія масового обслуговування добре розроблена і знайшла широке застосування для створення математичних моделей при моделюванні обчислювальних систем.

Системи, стани котрих визначені в дискретні моменти часу t_0, t_1, t_2,…, одержали назву автоматів. У кожний дискретний момент часу (за виключенням t_0) в автомат надходить вхідний сигнал $x(t)$, під дією якого автомат переходить в новий стан відповідно до функції переходів і видає вихідний сигнал, який визначається функцією виходів. Якщо автомат характеризується обмеженими множинами станів z, вхідних сигналів x і вихідних сигналів y, то його називають кінцевим автоматом. Функції переходів і виходів кінцевого автомата задаються таблицями, матрицями чи графіками. Для формалізованого опису функціонування систем використовується також обчислення висловлювань, мережі Петрі та ін.

Таким чином, побудова математичної моделі передбачає аналіз концептуальної моделі і вихідних даних з метою вибору однієї з підходящих формалізованих схем. Якщо це не вдається зробити для всієї системи, то формалізовані схеми можуть бути використані для опису окремих елементів, а вся система описується за допомогою програмного чи структурного підходу.

Вибір методу моделювання. Розроблена математична модель може бути досліджена різними методами – аналітичними, чисельними, якісними, імітаційними.

Аналітичні методи – методи, за допомогою яких можна провести найбільш повне дослідження моделі. В деяких випадках наявність аналітичної моделі робить можливим застосування математичних методів оптимізації. Для використання аналітичних методів математичну модель перетворюють до виду явних аналітичних залежностей між

характеристиками і параметрами системи та зовнішніми діями. Це вдається для простих систем. Застосування математичних методів для більш складних систем пов'язане з більшим, в порівнянні з іншими методами, ступенем спрощення реальності і абстрагуванням. Тому аналітичні методи дослідження використовуються, як правило, для первинної грубої оцінки характеристик всієї системи чи окремих її підсистем, а також на ранніх стадіях проектування систем, коли недостатньо інформації для побудови більш точної моделі.

Чисельні методи. Ряд моделей не піддається розв'язанню відомими аналітичними методами. Для їх дослідження можуть бути використані чисельні методи. Вони прийнятні для більш широкого класу систем, для котрих математична модель представлена у вигляді системи рівнянь, що допускає розв'язання чисельними методами. Застосування чисельних методів особливо ефективне за допомогою швидкодіючих комп'ютерних систем. Результатом дослідження систем чисельними методами є таблиці значень величин, що знаходяться для скінченного набору значень параметрів системи.

Якісні методи. Якщо одержані рівняння не вдається розв'язати аналітичними чи чисельними методами, то вдаються до якісних методів, які дозволяють в ряді випадків оцінити асимптотичні значення величин, що визначаються, стійкість, а також судити про поведінку траєкторії системи в цілому.

Імітаційні методи. При імітаційному моделюванні динамічні процеси системи-оригіналу замінюються процесами, що імітуються в абстрактній моделі, але з дотриманням таких же співвідношень, тривалостей і часових послідовностей окремих операцій (в масштабі модельного часу). Тому метод імітаційного моделювання інколи називають алгоритмічним чи операційним. У процесі імітації, як при експерименті з оригіналом, фіксують певні події і стани чи виміряють вихідні дії, за котрими обчислюють характеристики якості функціонування системи.

Імітаційне моделювання дозволяє розглядати процеси, що проходять в системі, практично на будь-якому рівні деталізації. В імітаційній моделі можна реалізувати будь-який алгоритм керування чи функціонування системи. Моделі, котрі досліджуються аналітичними методами, можуть бути досліджені і імітаційними методами. Все це сприяє застосуванню імітаційних методів моделювання як основних для моделювання складних систем.

Методи імітаційного моделювання класифікуються залежно від класу досліджуваних систем, способу просування модельного часу, виду кількісних змінних параметрів системи і зовнішніх дій. Більш детально імітаційне моделювання розглядається в пп. 1.1.4.

Розділяють методи моделювання дискретних і безперервних систем.

Систему називають дискретною, якщо всі елементи системи мають обмежену множину станів і перехід з одного стану в інший здійснюється миттєво.

Кількісні параметри системи і зовнішніх дій можуть бути детермінованими й випадковими. За цією ознакою розрізняють детерміноване й статистичне моделювання. При статистичному моделюванні для одержання достовірних імовірнісних характеристик процесів функціонування системи необхідне їх багатократне відтворення з різними конкретними значеннями випадкових факторів і статистичною обробкою результатів вимірювань.

При нестаціонарному характері змінних використовуються спеціальні методи моделювання, зокрема метод повторних експериментів.

Ще одним класифікаційним параметром при створенні математичної моделі вважають схему формалізації. При цьому методи розділяють на алгоритмічний (програмний) чи структурний (агрегатний). У першому випадку процеси керують елементами (ресурсами) системи, а в другому – елементи керують процесами, визначають порядок функціонування системи.

Таким чином, вибір того чи іншого методу моделювання повністю визначається математичною моделлю і вихідними даними.

Вибір засобів моделювання та розробка програмної моделі. Для дослідження моделей застосовуються технічні засоби – універсальні (персональні) комп'ютери чи спеціалізовані обчислювальні системи. Для реалізації аналітичного моделювання за допомогою універсальних комп'ютерів не задаються особливі вимоги до технічних засобів. При статистичному моделюванні необхідно досить багато машинного часу, тому бажано використовувати високопродуктивні комп'ютери.

У зв'язку із широким застосуванням імітаційного моделювання все більш актуальними стають розробка і випуск спеціалізованих обчислювальних засобів. До них відносяться стохастичні машини, машини імітаційного моделювання і гібридні обчислювальні комплекси.

Як програмні засоби можуть бути використані процедурно-орієнтовані алгоритмічні мови, програмно-орієнтовані мови чи автоматизовані системи моделювання.

Програмні та технічні засоби моделювання обираються з врахуванням ряду критеріїв. Необхідна умова при цьому – достатність і повнота засобів для реалізації концептуальної і математичної моделі. Серед інших критеріїв можна назвати доступність засобів, наявність у дослідника інформації про ті чи інші засоби. Важливе значення має простота і легкість засвоєння програмних засобів моделювання, швидкість і коректність створення програмної моделі, існування методики застосування засобів для моделювання систем певного класу.

При розробці програм імітаційного моделювання виникають задачі,

загальні для широкого класу моделей. Це: організація псевдопаралельного виконання алгоритмів; динамічний розподіл пам'яті; операції з модельним часом, що відображає астрономічний час функціонування оригіналу; імітація випадкових процесів; введення масиву подій; збирання і обробка результатів моделювання. Щоб полегшити вирішення цих задач, були створені спеціальні мови моделювання. За структурою і правилами програмування мови моделювання подібні до процедурно-орієнтованих алгоритмічних мов високого рівня. Вони мають той чи інший набір операторів, що супроводжуються відповідними операндами. Але оператори мов моделювання визначають виконання більш складних процедур, тому мови моделювання мають більш високий рівень в порівнянні з алгоритмічними мовами, що спрощує складання програм. Мови моделювання є формалізованим базисом створення математичних моделей. На сьогоднішній час відомо більше 500 мов моделювання.

Для спрощення і прискорення процесу створення машинних моделей був реалізований ряд ідей з автоматизації програмування імітаційних моделей. Створено ряд систем, котрі звільняють дослідника від програмування. Програма створюється автоматично за однією з формалізованих схем на основі заданих дослідником параметрів системи, зовнішніх дій і особливостей функціонування. Вихідні дані представляються тією чи іншою канонічною формою, чи в ході діалогу з обчислювальною системою. За результатами машинного експерименту основні вихідні змінні обчислюються і виводяться автоматично, додаткові – за вказівкою дослідника. Такі системи називають універсальними автоматизованими імітаційними моделями чи генераторами імітаційних програм.

Перевірка адекватності і корегування моделі. Суть перевірки адекватності моделі системи полягає в аналізі її відповідності до досліджуваної системи, а також рівнозначності системі. Але модель не може бути повним відображенням системи, інакше губиться сенс її створення. У процесі створення моделі адекватність порушується в результаті орієнтації, стратифікації, деталізації і локалізації. Крім того, адекватність порушується через ідеалізацію зовнішніх умов і режимів функціонування, виключення тих, чи інших параметрів, не врахування деяких випадкових факторів. Відсутність точних відомостей про зовнішні дії, певні нюанси структури системи, прийняті апроксимації, інтерполяції, пропозиції і гіпотези теж ведуть до зменшення відповідності між моделлю і системою. Все це може стати причиною того, що результати моделювання будуть істотно відрізнятися від реальних.

Найпростішою мірою адекватності може бути відхилення певної характеристики y_{ok} оригіналу і y_{mk} моделі:

$$\Delta y = |y_{ok}-y_{mk}|, \qquad (1.10)$$

або відношення відхилення до характеристики оригіналу

$$\Delta y = |y_{ok}\text{-}y_{mk}|/y_{ok}. \quad (1.11)$$

Тоді можна вважати, що модель адекватна системі, якщо імовірність того, що відхилення Δy не перевищує граничної величини Δ, не менше від припустимої імовірності P_{Δ}:

$$P(\Delta y<\Delta) \geq P_{\Delta}. \quad (1.12)$$

Але практичне використання даного критерію не завжди можливе за відсутності інформації про значення характеристики y_{ok} для систем, що проектуються чи модернізуються. На практиці оцінка адекватності зазвичай проводиться шляхом експертного аналізу розумності результатів моделювання, при цьому виконуються такі види перевірок:

- перевірка моделей елементів (у сумнівних випадках слід деталізувати елемент чи провести додатковий аналіз);
- перевірка моделі зовнішніх дій;
- перевірка концептуальної моделі функціонування системи (виявляються помилки постановки задачі);
- перевірка формалізованої і математичної моделі;
- перевірка способів вимірювання і обчислення вихідних характеристик (виявляються помилки розв'язування);
- перевірка програмної моделі (аналізується відповідність операцій і алгоритмів функціонування програмної і математичної моделі, проводяться контрольні розрахунки при типових і граничних значеннях змінних, виявляються інструментальні помилки програмування).

Корегування моделі. Якщо за результатами перевірки адекватності виявляються недопустимі розбіжності моделі і системи, то виникає необхідність в корегуванні моделі. При цьому виділяють такі типи змін: глобальні, локальні і параметричні.

Необхідність в глобальних змінах виникає у випадку виявлення методичних помилок в концептуальній чи математичній моделі. Для усунення таких помилок буде потрібна розробка нової моделі.

Локальні зміни пов'язані з уточненням деяких параметрів чи алгоритмів. Вони виконуються шляхом заміни моделей компонентів системи і зовнішніх дій на еквівалентні, але більш точні моделі. Локальні зміни вимагають часткової зміни математичної моделі.

До параметричних відносяться зміни певних параметрів, котрі називаються калібрувальними. Для цього слід наперед виявити калібрувальні параметри і передбачити прості способи варіювання ними.

Стратегія корегування моделі повинна бути спрямована на першочергове введення глобальних змін, потім – локальних і в кінці –

параметричних.

Завершується етап перевірки адекватності і корегування моделі визначенням і фіксацією області придатності моделі. Під областю придатності розуміють множину умов, при дотриманні котрих точність результатів моделювання знаходиться в допустимих межах.

Планування експериментів з моделлю. Цілі моделювання досягаються шляхом дослідження розробленої моделі. Дослідження полягають в проведенні експериментів, в результаті котрих визначаються вихідні характеристики системи при різних значеннях керованих змінних параметрів моделі. Експерименти проводять за певним планом. Це особливо важливо при уявному і імітаційному моделюванні на універсальних комп'ютерах. Грунтується це на тому, що в такому випадку може бути велика кількість можливих сполучень значень керованих параметрів, а кожний машинний експеримент проводиться при певному сполученні значень параметрів. При обмежених обчислювальних і часових ресурсах проведення всіх експериментів неможливе. Виникає необхідність у виборі певних сполучень параметрів і послідовності проведення експериментів. Цей підхід називається стратегічним плануванням. Сукупність методів зменшення тривалості машинного експерименту при забезпеченні статистичної достовірності результатів моделювання називають тактичним плануванням [1].

Аналіз результатів моделювання. Оскільки вихідні характеристики моделі часто є випадковими величинами чи функціями (зокрема, в результаті імітаційного експерименту), то обробка полягає в обчисленні оцінок математичного очікування, дисперсій і кореляційних моментів. Оцінки, одержані в результаті статистичної обробки вимірювань, повинні бути незміщеними, змістовними і ефективними.

Для виключення необхідності зберігання в комп'ютері всіх вимірювань, обробку результатів вимірювань проводять за рекурентними формулами, коли оцінки обраховують в процесі експерименту методом нарощуваного підсумка в міру появи нових вимірювань. Для стохастичних характеристик будують гістограми відносних частот – емпіричну густину розподілу. Для побудованої гістограми підбирають теоретичний закон розподілу. Це також робиться і при підготовці вихідних даних моделювання.

Для випадкових нестаціонарних характеристик період моделювання T_m розбивається на відрізки з постійним кроком Δt (прогони чи перетини), і запам'ятовуються значення характеристик в кінці кожного прогону. Проводиться серія експериментів із різними послідовностями випадкових параметрів моделі. Потім вимірювання кожного перетину обробляються, як при оцінці випадкових величин.

Визначення залежностей характеристик від параметрів системи. За результатами стохастичного моделювання може бути проведено аналіз

залежностей характеристик від параметрів системи і зовнішній дій. Для цього можна скористатися кореляційним, дисперсійним чи регресійним методами.

За допомогою кореляційного аналізу можна встановити наявність зв'язку між двома чи більше випадковими величинами. Коли коефіцієнт кореляції за абсолютною величиною дорівнює одиниці, то нестохастичний лінійний зв'язок між величинами, що аналізуються, наявний, а коли він дорівнює нулю, то зв'язок відсутній. Проміжні значення коефіцієнта кореляції відповідають наявності лінійного зв'язку з розсіюванням чи нелінійної кореляції.

Дисперсійний аналіз використовують для встановлення відносного впливу різних факторів на значення вихідних характеристик. При цьому загальна дисперсія характеристики розкладається на компоненти, котрі відповідають факторам, що розглядаються. За значеннями окремих компонентів роблять висновок про ступінь впливу того чи іншого фактора на характеристику, що аналізується.

Якщо всі фактори в експерименті є кількісними, то можна знайти аналітичну залежність між характеристиками і факторами. Для цього використовується метод регресійного аналізу. Знайдена залежність називається емпіричною моделлю. Регресійний аналіз полягає в тому, що обирається вид співвідношення між залежними і незалежними змінними, за експериментальними даними обчислюються параметри обраної залежності і оцінюється якість апроксимації експериментальних даних моделлю. Якщо якість незадовільна, то обирається залежність іншого виду, і процедура повторюється.

До аналізу результатів моделювання можна віднести задачу аналізу чутливості моделі до варіацій її параметрів. Під аналізом чутливості розуміють перевірку стійкості характеристик процесу функціонування системи до можливих відхилень значень параметрів.

Аналіз результатів моделювання дозволяє уточнити множину інформативних параметрів моделі, що може привести до первинного виду концептуальної моделі; знайти функціональні залежності характеристик параметрів, що інколи дає можливість створити аналітичні моделі системи чи визначити вагові коефіцієнти критерію ефективності.

Використання результатів моделювання. Результати моделювання використовуються для прийняття рішення про роботоздатність системи, для вибору кращого проектного варіанта чи для оптимізації системи. Рішення про роботоздатність системи приймається з того, виходять чи не виходять характеристики системи за встановлені межі при будь-яких допустимих змінах параметрів. При виборі кращого варіанта з усіх роботоздатних варіантів обирається той, у котрого максимальне значення критерію ефективності. Оптимізація системи є найбільш загальною і складною.

Необхідно знайти таку комбінацію значень змінних параметрів системи чи робочого навантаження з множини допустимих, котре максимізує значення критерію ефективності.

Після створення системи доцільна апостеріорна перевірка результатів моделювання і вимірювання характеристик функціонування. Така перевірка допомагає уточнити модель і підвищити ефективність системи. Наявність моделі діючої системи дає можливість прогнозувати якості функціонування при подальшому розвитку системи чи зміні зовнішніх дій.

1.1.4. Аналітичне моделювання

При аналітичному моделюванні використовуються методи формального представлення аналітичних перетворень певних об'єктів, котрі називаються аналітичними виразами. Вони включають операції диференціювання, інтегрування, зведення подібних, розкриття дужок, підстановки рівностей. Це досить великий клас перетворень. Але практично будь-яке аналітичне перетворення зводиться до підстановки однієї чи кількох рівностей в один чи кілька аналітичних виразів. Зведенню подібних і розкриттю дужок, наприклад, відповідає застосування рівностей, що характеризують дистрибутивність операцій додавання і множення.

Щоб надати цим твердженням точного розуміння, визначається область аналітичних перетворень як деяка формальна теорія, що визначається таким [2]:

1) Нехай $x_1, ..., x_n$ – предметні змінні; $a_1, ..., a_n$ – предметні константи; f_i – функціональні символи; A^j_i – предикатні символи, котрі складають алфавіт теорії.

2) Всяка постійна змінна x_i є термом t_i. Вираз типу $f_i\ (t_1, ..., t_n)$ – терм, якщо $t_1, ..., t_n$ – терми. Ніяких інших термів немає.

3) Вираз $A^n_i(t_1, ..., t_n)$ – елементарна формула. Всяка елементарна формула є формулою. Якщо A і B – формули, то $\neg A, A \supset B \forall x_i A$ (x_i – змінна) теж є формули і інших формул немає. ($\neg$ – ні; $\supset$ – якщо, то; $\forall$ – для всіх).

Визначивши логічні і власні аксіоми, а також правила виведення, одержують певну теорію першого порядку [2]. Представлення аналітичних перетворень в певній формальній теорії дозволяє успішно розв'язувати задачі теоретичного і практичного характеру і дає можливість створювати ефективні системи автоматизованого програмування. Одержані результати є основою при розробці мовного процесора аналітичних перетворень на базі загальноцільових макрозасобів.

Потоки заявок. Під час аналітичного моделювання характеристики системи обчислюються найбільш просто для потоку заявок, котрий називається найпростішим. Найпростіший потік – це потік заявок, який має такі властивості: стаціонарність; відсутність післядії; ординарність.

Стаціонарність означає постійність ймовірності того, що на протязі певного інтервалу часу надійде однакова кількість заявок незалежно від розташування інтервалу на осі часу. Відсутність післядії полягає в тому, що заявки, котрі надійшли, не впливають на майбутній потік заявок, тобто заявки надходять в систему незалежно одна від одної. Ординарність означає, що в кожний момент часу в систему надходить не більше однієї заявки. Будь-який потік, котрий має такі властивості, є найпростішим.

У найпростішого потоку інтервали часу τ між двома послідовними заявками є незалежними випадковими величинами з експоненціальною функцією розподілу

$$F(\tau) = 1 - e^{-\lambda\tau}. \tag{1.13}$$

Такий розподіл має густину

$$f(\tau) = \lambda e^{-\lambda\tau}, \tag{1.14}$$

математичне очікування довжини інтервалу

$$M(\tau) = \int_0^\infty \tau f(\tau) d\tau = \frac{1}{\lambda}, \tag{1.15}$$

дисперсію

$$D(\tau) = \int_0^\infty (\tau - M(\tau))^2 f(\tau) d\tau = \frac{1}{\lambda^2} \tag{1.16}$$

і середньоквадратичне відхилення, що дорівнює математичному очікуванню.

Експоненціальний розподіл характеризується одним кількісним параметром – інтенсивністю λ. Найпростіші потоки заявок мають такі особливості:

1) сума M незалежних, ординарних, стаціонарних потоків з інтенсивностями λ_i (i = 1, ..., M) сходиться до найпростішого потоку з інтенсивністю $\lambda = \sum_{i=1}^{M} \lambda_i$ за умови, що потоки, котрі додаються, виявляють приблизно однаковий малий вплив на сумарний потік;

2) потік заявок, одержаний в результаті випадкового розрідження вихідного стаціонарного ординарного потоку, котрий має інтенсивність λ, коли кожна заявка виключається з потоку з певною ймовірністю p незалежно від того, виключені інші заявки чи ні, утворює найпростіший потік з інтенсивністю $p\lambda$;

3) інтервал часу між довільним моментом часу і моментом надходження чергової заявки має експоненціальний розподіл з тим же математичним очікуванням 1/λ, що і інтервал часу між двома послідовними заявками.

Найпростіший потік набув великого поширення не тільки за аналітичну

простоту пов'язаної з ним теорії, але й за те, що більшість реально спостережуваних потоків статистично не відрізняється від найпростішого.

Пуассонівський потік. Пуассонівським потоком називається ординарний потік заявок з відсутністю післядії, у котрого число заявок, що поступили в систему за проміжок часу τ, розподілено за законом Пуассона:

$$P(k,\tau) = \frac{(\lambda\tau)^k}{k!} e^{-\lambda\tau}, \quad \lambda > 0, \tag{1.27}$$

де $P(k,\tau)$ – ймовірність того, що за час τ в систему надійде точно k заявок; λ – інтенсивність потоку заявок.

Математичне очікування і дисперсія розподілу Пуассона дорівнюють λτ.

Розподіл Пуассона дискретний. Стаціонарний пуассонівський потік є найпростішим. У розподілі Пуассона тривалості інтервалів між двома послідовними заявками – це випадкові величини з експоненціальним розподілом.

1.1.5. Принципи імітаційного моделювання

Метод імітаційного моделювання полягає у створенні логіко-аналітичної (математичної) моделі системи і зовнішніх дій, в імітації функціонування системи, тобто у визначенні часових змін стану системи під впливом зовнішніх дій і в одержанні вибірок значень вихідних змінних, за котрими визначаються їх імовірнісні характеристики. Це визначення справедливе для стохастичних систем. При дослідженні детермінованих систем відпадає необхідність в одержанні вибірок значень вихідних змінних (функцій).

Взагалі імітаційне моделювання – це метод дослідження, котрий ґрунтується на тому, що динамічна система, яка аналізується, замінюється імітатором, і з ним проводяться експерименти для одержання інформації про систему, що вивчається. Роль імітатора виконує спеціальна програма обчислювальної системи.

Моделювання з використанням комп'ютерів, тобто імітаційне моделювання, є в наш час найбільш ефективним засобом дослідження складних систем. Схема такого дослідження включає такі етапи:

- формалізацію системи з метою побудови її математичної моделі;
- розробку і складання моделюючого алгоритму (алгоритмічної моделі) і програми, котра його реалізує;
- відпрацювання моделюючого алгоритму на комп'ютері;
- обробку і аналіз результатів.

Розділення етапів побудови моделі і проведення імітаційних експериментів обумовлено тим, що машинна модель при формуванні плану

експерименту розглядається як чорний ящик. На етапі побудови моделі визначають її параметри. Результати імітаційних експериментів можуть впливати на вид моделюючого алгоритму лише після їх проведення. Наприклад, якщо в процесі експерименту виясниться, що вихідні результати слабо залежать від того чи іншого параметра, то це може бути причиною спрощення моделі, суть котрого полягає в усуненні даного параметра і відповідному зменшенні розмірності моделі.

Залежно від ступеня формалізації системи, що досліджується, і від способу побудови моделюючого алгоритму розрізняють:

- моделювання з використанням чисельних методів;
- імовірнісне чи стохастичне моделювання із застосуванням спеціальних алгоритмічних мов моделювання.

Моделювання з використанням чисельних методів здійснюється в тих випадках, коли систему вдається описати легкоспостережуваними і достатньо строгими математичними співвідношеннями.

Застосування методу стохастичного моделювання робить моделюючий алгоритм за структурою близьким до алгоритму функціонування досліджуваної системи. Зміна умов моделювання не приводить до істотних змін моделюючого алгоритму, котрий може просто доповнюватись новими блоками чи відпрацьовуватись більшу кількість разів на комп'ютері, наприклад, з метою підвищення точності моделювання.

Загальноприйнята схема стохастичного моделювання наведена на рис. 1.3 і містить три блоки:

- блок імітації випадкових процесів, котрі діють на систему;
- блок програми функціонування системи;
- блок статистичної обробки результатів моделювання.

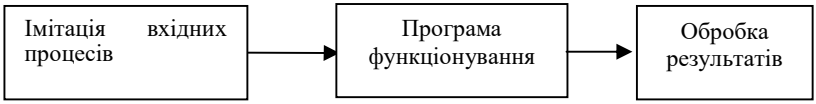


Рис. 1.3. Схема стохастичного моделювання

Статистичні імітаційні моделі є програмним відтворенням структури системи і тих елементарних дій, котрі виконують її окремі елементи.

Основна ідея методу імітаційного моделювання стохастичних систем ґрунтується у багатьох питаннях на методі обчислення випадкових величин, котрий називається методом стохастичних випробувань чи методом Монте-Карло [3]. Він полягає у такому. Нехай необхідно визначити функцію розподілу випадкової величини y. Припустимо, що шукана величина y може бути представлена у вигляді залежності $y = f(\alpha, \beta, ..., \omega)$, де α, β, ..., ω –

випадкові величини з відомими функціями розподілу.

Для розв'язання задачі такого типу використовується наступний алгоритм:

- за кожною з величин α, β, ..., ω здійснюється випадкове випробування, у результаті якого визначається певне конкретне значення випадкової величини α_i, β_i, ..., ω_i;
- використовуючи знайдені величини, визначають одне часткове (певне) значення y_i за наведеною вище залежністю;
- попередні операції повторюються N разів, у результаті чого визначається N значень випадкової величини y_i;
- на основі N значень величини y_i знаходиться її емпірична функція розподілу.

Моделюючий алгоритм, відображаючи елементарні події, котрі відбуваються в системі, слугує для одержання тієї чи іншої інформації про динаміку системи. При моделюванні складних систем, як правило, використовуються імітаційні методи. При цьому домагаються того, щоб моделюючий алгоритм, його структура залежали б не від набору показників роботи системи, а лише від самої математичної моделі. Цього досягають тим, що окремі операції моделюючого алгоритму мають відповідати елементарним явищам, що відбуваються в системі, а послідовність виконання цих операцій повинна відповідати взаємодіям вказаних явищ чи структурі системи. Оскільки моделюючий алгоритм відтворює роботу математичної моделі, то імітаційний підхід вимагає, щоб і математична модель обчислювальної системи структурно і динамічно відповідала реальній системі.

Оператори формування і реалізації випадкових об'єктів (процесів). Вони призначені для імітації різних випадкових факторів (випадкових подій, величин, полів, векторів, функцій), котрі супроводжують процес, що досліджується. Вихідним матеріалом для роботи цих операторів при реалізації на комп'ютері слугують псевдовипадкові числа. Ці числа повинні бути рівномірно розподілені в інтервалі (0,1). Формування чисел можна виділити як окремий оператор, а задачею інших операторів цього ж класу є їх перетворення таким чином, щоб одержати реалізацію заданого закону розподілу з параметрами заданого випадкового об’єкта.

Реалізація модельного часу. Модельний час, котрий відображає час функціонування реальної системи, є одним з основних параметрів при імітаційному моделюванні. Оскільки як моделюючі алгоритми виконуються на комп’ютері, то модельний час є дискретним з кроком квантування Δt.

Важливим є питання просування модельного часу та вибору кроку квантування Δt. Для цього використовують кілька принципів, на основі яких розробляють різні способи апроксимації характеристик стану складних

обчислювальних систем і побудови відповідних моделюючих алгоритмів [1].

Принцип Δt (*принцип приросту часового інтервалу*). Використовується при достатньо малому Δ*t*, для одержання кусочно-задовільної апроксимації характеристик стану системи на кожному кроці дискретизації Δ*t* . Цей принцип є найбільш універсальним і практично придатним для будь-яких систем, але він неекономічний щодо машинного часу. Згідно з принципом Δ*t*, модельний час просувається на деяку величину Δ*t*. Визначаються зміни станів елементів і вихідних дій системи, котрі пройшли за цей час. Після цього модельний час знову просувається на величину Δ*t*, і процедура повторюється до кінця періоду моделювання T_m. Крок Δ*t* в більшості випадків є постійним, але в загальному випадку він може бути і змінним.

Принцип подій. Застосовується при імітаційному моделюванні реальних систем з дискретними вхідними діями та дискретним часом відповідно. При цьому вважається, що стан системи не змінюється між двома сусідніми подіями, а в момент надходження події (особливий стан системи, в якому система знаходиться на протязі дуже короткого інтервалу часу, який при моделюванні вважається нульовим), характеристики системи змінюються стрибком, згідно з алгоритмом реакції системи на вказану подію.

При цьому в поточний момент модельного часу *t* спочатку аналізуються ті майбутні події – одержання дискретної вхідної дії (заявки), завершення обслуговування і т. ін., для яких були визначені моменти їх настання $t<t_i < T_m$. Обирається найбільш рання подія, і модельний час просувається до моменту настання цієї події. Аналізується реакція системи на поточну подію – зміна станів пов'язаних з цією подією модулів системи та генерація нових типів наступних події, пов'язаних з новими станами. Процедура повторюється до завершення періоду моделювання T_m., або до настання особливого стану системи (нероботоздатність, колапс, та ін.).

Принцип послідовного проведення заявок. Є модифікацією принципу подій при моделюванні систем масового обслуговування і полягає в послідовному відтворенні історії кожної із заявок (пріоритету, часу надходження, місця формування) в порядку їх надходження в систему. Алгоритм звертається до інформації про інші заявки тільки в тому випадку, коли це необхідно для вирішення питання щодо долі даної заявки. Алгоритми, побудовані за цим принципом, мають складну логічну структуру, але є найбільш економічними по відношенню до машинного часу.

Нерідко при побудові моделюючих алгоритмів використовують кілька принципів одночасно. Наприклад, загальну структуру моделі будують за принципом особливих станів, а моделюючий алгоритм між особливими станами – за принципом послідовного проведення заявок.

Фіксація і обробка результатів моделювання. При моделюванні складних обчислювальних систем здійснюється багатократна реалізація на

комп’ютері моделюючого алгоритму для одержання статично стійких оцінок шуканих величин. При цьому накопичується значний обсяг статистичної інформації. З метою економії пам'яті комп'ютерної системи необхідно здійснювати фіксацію результатів моделювання та їх статистичну обробку безпосередньо в процесі моделювання з одержанням певних біжучих оцінок для шуканих характеристик.

До якості оцінок, одержаних в результаті статистичної обробки результатів моделювання, ставляться такі вимоги: оцінки повинні бути незміщеними, змістовними (істотними) і ефективними.

Коли серед результатів моделювання присутні випадкові величини, то в якості оцінок для шуканих характеристик розраховують середні значення, дисперсії, кореляційні моменти.

Імітаційне моделювання дає можливість враховувати надійнісні характеристики обчислювальних систем. Зокрема, якщо час напрацювання на відмову і відновлення всіх пристроїв, котрі входять в систему відомий, то визначаються моменти виникнення відмов і моменти відновлення пристроїв протягом періоду моделювання. Якщо в моменти виникнення відмови пристрій зайнятий обслуговуванням заявки, то може прийматися різне рішення залежно від типу пристрою і режиму його роботи: заявка знімається і більше не обслуговується (вибуває з системи) чи заявка ставиться в чергу, а після відновлення пристрою дообслуговується або надходить на повторне обслуговування.

Етапи побудови імітаційної моделі. Розглянемо більш детально розробку моделюючого алгоритму, який реалізує принцип подій. В цьому випадку робота виконується з застосуванням двох основних таблиць – таблиці станів і таблиці подій, які при програмній реалізації можуть бути реалізовані в вигляді структур, об’єктів та ін.

Формування таблиці станів. Система, що моделюється, з заданою точністю деталізації розбивається на блоки (модулі). Всі модулі групуються за типами і нумеруються. В більшості випадків таблиця станів має стільки рядків, скільки модулів в системі (якщо немає динамічно-змінного числа модулів). Число стовпців таблиці станів залежить від типу імітаційної моделі, складності системи, що моделюється та ін., але обов'язковими полями таблиці станів є:

1) унікальний номер модуля;

2) тип модуля, або його ім'я;

3) поточний стан модуля;

4) поля, що показують зв'язок даного модуля з іншими.

Ефективна побудова полів 4-го типу є деякою мірою мистецтвом. Крім того в деяких складних багаторівневих системах дані поля доповнюються різного виду логічними та алгоритмічними перевірками. Головна вимога до

таблиці станів така: вона повинна повністю задавати взаємозв'язок між модулями системи, що моделюється.

Для кожного типу модулів, залежно від призначення імітаційної моделі, задаються можливі типи станів, які кодуються (символами, цифрами, або якось по-іншому). Визначаються початкові стани всіх модулів і заносяться в поле 3 "поточний стан модуля".

Таблиця станів сформована. У подальшому при імітаційному моделюванні поточні стани модулів динамічно змінюються, відображаючи таким чином зміни в реальній системі.

Формування таблиці подій. Подія – ключове поняття імітаційного моделювання – є причиною зміни станів. Якщо всі можливі стани зобразити у вигляді вершин орієнтованого графа, то всі можливі події – це дуги вказаного графа. Таким чином визначається множина можливих подій на всіх можливих станах всіх типів модулів, які аналогічним чином кодуються.

Таблиця подій (майбутніх подій, які повинні виникати в системі) є динамічною таблицею, яку при програмуванні доцільно організувати у вигляді динамічних структур. Кожен запис у таблиці подій (незалежно від типу імітаційної моделі) характеризується трьома атрибутами і складається з трьох полів відповідно:

1) номер модуля, на якому виникає подія;

2) тип (код) події;

3) модельний час виникнення події.

В нульовий момент модельного часу формується початкова таблиця майбутніх подій, при цьому в циклі перебираються всі модулі (проглядається таблиця станів) і залежно від типу модуля та його поточного стану генеруються всі дозволені типи подій (по одній події на кожний тип для кожного модуля). Таким чином поля 1 та 2 задаються явно. А звідки взяти в програмі поле типу 3 – час виникнення події (в реальній системі події виникають самі по собі)?

Для цього використовують статистичні властивості подій (хоча є детерміновані імітаційні моделі, в яких час майбутньої події однозначно визначається залежно від поточного стану модуля).

Події на різних типах модулів підкоряються різним законам розподілу (допускається, що ці закони, а також їх характеристики відомі розробникам імітаційної моделі), тому час наступної події розраховується, виходячи з відомого закону з використанням генератора псевдовипадкових чисел.

Так, для імітаційної моделі надійності, де всі події підкоряються експотенціальному закону, час події розраховується за формулою

$$T = -\frac{1}{\lambda} ln(1 - N), \qquad (1.28)$$

де λ – інтенсивність події даного типу; N – псевдовипадкове число, рівномірно розподілене в інтервалі [0,1].

Псевдовипадкове число N одержується програмою RANDOM. Після заповнення таблиці подій, вона ранжується за зростанням по полю 3 – час виникнення події. Тепер починається сам етап імітаційного моделювання.

Алгоритм імітаційної моделі. При імітаційному моделюванні з таблиці подій вибирається найближча за часом подія (при цьому модельний час дискретно змінюється і прирівнюється до часу настання події), і розраховується реакція моделі на дану подію (яка в кінці приводить до зміни поточних станів модулів). Реакцією системи може бути:

1) зміна стану того модуля, на якому виникла подія (номер модуля є в таблиці подій);

2) зміна станів інших модулів, зв'язаних з даним модулем (обчислюється при аналізі полів таблиці станів);

3) зміна стану всієї системи (обчислюється при перевірці різних критеріїв);

4) генерація нових подій (з врахуванням поточного модельного часу), зумовлених новими станами модулів.

Таким чином, при імітаційному моделюванні першою дією розрахунку реакції є аналіз типу події і реалізація процедур розрахунку реакції для кожного типу події.

Слід відмітити, що після генерації нових подій таблиця подій по-новому ранжирується (можна за простішим алгоритмом).

Імітаційне моделювання виконується до лімітованого часу T_m, або до досягнення конкретного стану всієї системи. Під час імітаційного моделювання формується протокол подій (по кожному модулю, по групі, або по всій системі), на основі якого розраховуються статистичні характеристики системи.

Оскільки вхідними даними для імітаційної моделі є випадкові числа – формула (1.28), то імітаційна модель дає випадкові точечні оцінки, але при достатньо великому часі моделювання (для ергодичних процесів), або при багатократному прорахунку моделі вона дає стійкі статистичні оцінки.

Контрольні питання

1. Задачі, які розв'язуються за допомогою моделювання складних об'єктів.
2. Основні поняття моделювання. Формалізація.
3. Що таке система? Основні визначення та формалізація.
4. Теорія подібності. Основні визначення.
5. Класифікація моделей і області їхнього застосування.
6. Що таке фізична модель і які типи фізичних моделей вам відомі?

7. Що таке абстрактна модель і які типи абстрактних моделей вам відомі?

8. Яке основне призначення концептуальної моделі?

9. Що таке математична модель і які типи математичних моделей вам відомі?

10. Принципи імітаційного моделювання.

11. Етапи розробки моделей.

12. Які задачі розв'язуються на етапі формулювання мети моделювання?

13. Які задачі розв'язуються на етапі вибору засобів моделювання?

14. Які задачі розв'язуються на етапі розробки концептуальної моделі?

15. Які задачі розв'язуються на етапі підготовки вихідних даних?

16. Які задачі розв'язуються на етапі розробки математичної моделі?

17. Які задачі розв'язуються на етапі вибору методу моделювання?

18. Які задачі розв'язуються на етапі розробки програмної моделі?

19. Які задачі розв'язуються на етапі перевірки адекватності та корегування моделі?

20. Які задачі розв'язуються на етапі планування машинних експериментів?

21. Які задачі розв'язуються на етапі комп’ютерного моделювання?

22. Які задачі розв'язуються на етапі аналізу результатів моделювання?

23. Сутність аналітичного моделювання.

24. Основні етапи імітаційного моделювання.

25. Схема стохастичного моделювання при побудові імітаційної моделі.

26. Основні положення методу стохастичних випробувань (методу Монте-Карло).

27. Реалізація модельного часу та принципи побудови моделюючих алгоритмів.

28. Структура таблиці станів при імітаційному моделюванні.

29. Формування таблиці подій при імітаційному моделюванні.

30. Алгоритм імітаційної моделі. Взаємодія таблиць станів та подій.

31. Що таке реакція системи на подію і як вона розраховується?

1.2. Аналіз та попередня обробка множини вхідних та вихідних даних при розробці моделей складних об’єктів

Показники якості моделей складних об'єктів залежать у першу чергу від одержання точних даних і інформації про стан об'єктів, що досліджуються, а також про взаємодію об'єктів з зовнішнім технічним і природним середовищем.

1.2.1. Типи вхідних даних – бінарні, рангові, чисельні та методи їхньої обробки

В задачах класифікації та діагностики вхідні дані одержали назву вхідних ознак. За метрологічною оцінкою вхідні ознаки поділяються на такі типи:

1) Кількісні або числові ознаки. Це заміряні у визначеній шкалі і виражаються числами з визначеною точністю виміру (результати інструментальних досліджень і ознаки, отримані в результаті обробки сигналів та зображень).

2) Якісні, рангові або бальні. Використовуються для вираження експертних оцінок, термінів і понять, не мають цифрових значень (наприклад, тяжкість патології) і заміряються у шкалі порядку.

3) Бінарні або дихотомічні. Набувають тільки двох значень ("0" або "1", "ТАК" або "НІ") і використовуються для фіксації у формалізованих документах наявності або відсутності якоїсь ознаки.

4) Класифікаційні або номінальні (наприклад, стать, професія, група крові). Це ознаки, заміряні в шкалі найменувань.

Для кожного з розглянутих типів ознак застосовуються свої методи дослідження, хоча багато алгоритмів, які розроблені для одного типу, адаптуються до інших типів [4]. При цьому змістовна частина інформації містить такі її складові:

- систематичну інформацію, котра викликана досліджуваними впливами, що і є зазвичай предметом дослідження;
- систематичну інформацію, пов'язану з умовами дослідження (методами дослідження), тобто постійними похибками (помилками);
- випадкову (залишкову) інформацію, викликану нерегулярними змінами у процесі дослідження.

Таким чином, при побудові моделей складних об'єктів (особливо в медичній діагностиці) використовується різнорідна, отримана різним дослідниками (і в різний час), недостатньо формалізована і така, що несе елементи суб'єктивної оцінки експерта, інформація. Тому обов'язковими етапами первинної обробки інформації є етапи формалізації опису і формування переліку вхідних ознак для даної задачі дослідження.

При формалізації опису кожному значенню ознаки ставиться у відповідність визначене кодове число (оцифровка шкал). При оцифровці шкал виконується зведення всіх типів ознак до однієї кількісної шкали.

Кодування не повинне змінювати семантичну силу і зміст вихідних даних, тобто предметний зміст інформації.

Кількісні ознаки кодуються у вихідному виді (значення) або в квантованому виді (належність кількісної ознаки до діагностично-значимих інтервалів). При неправильному визначенні кількості і границь інтервалів

можна утратити важливу для розпізнавання властивість даної ознаки.

У найпростішому випадку дихотомічної шкали, тобто коли ознака може набувати значень «так» або «ні», немає великої різниці, які числа будуть приписані позитивній або негативній відповіді. Найпоширеніші варіанти такі: відповіді «так» приписують число 1, відповіді «ні» – число –1, або 0.

Рангові ознаки підрозділяються на градації відповідно до зміни їхньої виразності або в зростаючому, або в спадному ряді значень. При цьому застосовується як 4-градаційна шкала, що відповідає прийнятому експертами ступеню виразності (відсутність ознаки або норма, слабкий ступінь, середній ступінь і сильний ступінь прояву ознак), так і 7-градаційна шкала, що дозволяє виявити дискретність прояву якісних ознак і яка відповідає психофізичним можливостям людини по переробці інформації (закон "сім плюс мінус два").

У випадку порядкових шкал, як правило, порядок проходження градацій ознаки відображає ступінь посилення або ослаблення тієї або іншої якості. Числові мітки ознаки в цьому випадку привласнюються таким чином, щоб відстані між двома оцінками інтуїтивно відповідали різниці між відповідними градаціями (наприклад, якщо ознака має шкалу «добре», то логічно приписати градаціям мітки –1; 0; 1, а от у випадку шкали «малий–середній–великий–дуже великий» більш доречним може виявитися використання логарифмічних міток, тобто 0.1; 1; 10; 100).

Номінальним ознакам кодові значення можуть бути привласнені довільно, відповідно до прийнятого порядку перерахування показників.

Наступним кроком у статистичній обробці даних, як правило, є знаходження точки середнього значення всіх ознак – геометричного центра багатовимірної хмари точок даних. Зручно зрушити всі точки даних на однаковий вектор таким чином, щоб центр хмари виявився на початку координат. Таке перетворення називається центруванням даних.

Далі виконується нормування даних – тобто ділення усіх значень ознак на визначене число таким чином, щоб значення ознак попадали в порівнянні за величиною інтервали. В якості такого числа вибирається один із характерних масштабів.

У багатовимірній хмарі даних існує кілька масштабів. Перший – це середньоквадратичне відхилення

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(X_i - \overline{X})^2},$$

де $\overline{X} = \frac{1}{N}\sum_{i=1}^{N} X_i$ (X_i – вектор даних).

У випадку, якщо вибірка може вважатися отриманою із нормального розподілу, то в колі з центром в $\bar{x}$ та радіусом σ знаходиться близько двох третин від числа точок даних. Існує масштаб, який характеризує максимальне розсіювання в множині даних

$$R = \max_{i=1,N} \left\| X_i - \bar{X} \right\| .$$

Нормування всіх ознак на R призводить до того, що вся множина даних поміщається в коло одиничного радіусу.

Якщо масштаб вибрано σ або R, то відповідні формули обробки (нормування на «одиничну дисперсію» і «на одиничне коло») мають вигляд:

$$\tilde{X}_i = \frac{X_i - \bar{X}}{\sigma} , \; \tilde{X}_i = \frac{X_i - \bar{X}}{R} ,$$

де $\tilde{X}_i$ – новий вектор ознак; X_i – старий вектор ознак.

Крім того, якщо діапазони значень для різних ознак сильно відрізняються один від одного, то краще для кожної з ознак застосувати власний масштаб. Тобто, для кожної з ознак можна ввести своє середньоквадратичне відхилення та розсіювання:

$$\sigma_j = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_{ij} - \bar{x}_j)^2} , \; R_j = \max_{i=1,N} \left\| x_{ij} - \bar{x}_j \right\| ,$$

де $\bar{x}_j = \frac{1}{N} \sum_{i=1}^{N} x_{ij}$ (x_{ij} – значення i -ї ознаки на j -му векторі), .

Як результат отримаємо формули для нормування на «одиничну дисперсію для кожної ознаки» і «на одиничний куб»:

$$\tilde{x}_{ij} = \frac{x_{ij} - \bar{x}_j}{\sigma_j} , \; \tilde{x}_{ij} = \frac{x_{ij} - \bar{x}_j}{R_j} .$$

Наступним важливим етапом обробки даних є аналіз різких відхилень у вихідних даних і відновлення пропущеної інформації. Статистичні процедури виділення відзначених "сумнівних" даних засновані на припущенні про однорідність даних, у той час як “дані, що вискакують”, розглядаються як атипічно далеко віддалені від центра розподілу. Сумнівні спостереження або цілком виключаються з подальшого розгляду, або їхній внесок зменшується за допомогою вагової функції, що убуває в міру зростання ступеня аномальності спостережень. Якщо є пропуски інформації (типова ситуація для медичних БД), то застосовуються різні підходи – "обнуління", умовне

кодування, усереднення, відновлення [1].

Слід зазначити, що при синтезі вирішальних правил доводиться використовувати ознаки, вірогідність яких викликає сумнів у розробників моделей та відповідних систем підтримки прийняття рішень (СППР) на вказаних моделях. У цьому випадку вводиться система вагових коефіцієнтів, що відображає кількісну оцінку їхньої значимості.

Важливим етапом попереднього аналізу даних є оцінка особливостей розподілу, тому що значне число статистичних методів припускає нормальний характер розподілу ймовірностей. Однак саме біологічні і медичні дані часто характеризуються значним відхиленням від нормального закону. При відхиленні закону розподілу від нормального використовуються непараметричні критерії (метод квантильних шкал, "бутстреп" і ін.).

Оцінка інформативності і формування інформативного простору ознак. Серед комплексу проблем, розв'язуваних при розробці СППР у роботі [5] виділено дві актуальні задачі оптимізації: добір інформативних ознак і оптимізація вирішальних правил. При цьому відзначається таке: "Якщо оптимізація вирішальних правил у якомусь ступені знаходить задовільне рішення, що обґрунтовується наявністю добре розробленої теорії перевірки статистичних гіпотез, то проблема власне аналізу різних методик обстеження освітлена слабо". Розроблювальна система вихідних діагностичних ознак повинна задовольняти таким вимогам:

1) Повнота опису. Система вихідних ознак має охоплювати усі виділені аспекти вимірюваного поняття.

2) Ощадливість опису. Найбільш розповсюдженою помилкою багатьох дослідників є "спроба аналізу украй великого числа ознак, що, на думку дослідників, повинне сприяти підвищенню інформативності наведенної вибірки". Однак для розв’язання будь-якої класифікаційної задачі необхідно використовувати корисну для даної задачі інформацію, що несе не "шум" і неіррелевантну інформацію (не відноситься до мети дослідження), тому при розробці системи ознак варто уникати зайвого обсягу вихідної інформації.

3) Структурованість системи ознак. Ознаки повинні групуватися, відносно рівномірно описуючи всі сторони вимірюваного явища.

4) Кількісна визначеність діагностичних ознак. Ця визначеність забезпечується формалізацією опису ознак, що розглянута вище.

Наведені вимоги не є вичерпними. У випадку одержання даних за допомогою тестів-опитувальників велика увага має приділятися прийомам зниження можливості фальсифікації відповідей і зменшення систематичної помилки тестування. У деяких випадках добір корисної інформації зі змістовних розумінь виконується дослідником самостійно (можливо не зовсім удало), у противному разі боротьба з надмірністю здійснюється формальними методами шляхом оцінки інформативності простору

діагностичних ознак.

1.2.2. Методи аналізу структури даних

За результатами вимірювання характеристик об'єктів в заданій предметній області і формалізації опису формується двовимірна таблиця експериментальних даних (ТЕД), структура якої показана в табл. 1.1.

Таблиця 1.1 – Структура таблиці експериментальних даних

Об'єкти навч. вибірки	Вихідні ознаки					
	x_1	x_2	...	x_j	...	x_p
A_1	x_{11}	x_{12}	...	x_{1j}	...	x_{1p}
A_2	x_{21}	x_{22}	...	x_{2j}	...	x_{2p}
...	...	...	...	...	...	...
A_i	x_{i1}	x_{i2}	...	x_{ij}	...	x_{ip}
...	...	...	...	...	...	...
A_N	x_{N1}	x_{N2}	...	x_{Nj}	...	x_{Np}

У цій таблиці прийняті такі позначення:

N – загальна кількість об'єктів навчальної вибірки;

p – загальна кількість ознак;

x_j – j-а ознака;

x_{ij} – значення j-ї ознаки, виміряне в i-го об'єкта;

$X = (x_1, ..., x_p)'$ – вектор ознак;

$A = \{A_1, ..., A_N\}$ – множина об'єктів.

ТЕД служить для синтезу структури діагностичної моделі й для оцінки її параметрів. У зазначеній моделі повинен у визначеній формі виражатися зв'язок вимірюваного вектора ознак X з тестуємою властивістю Y. При цьому необхідно забезпечити економічність за формою і змістовність за змістом перетворення $Y = f(X)$ при дотриманні заданої точності моделі для відображення моделлю загальних закономірностей структури експериментальних даних з метою підвищення стійкості і надійності кількісної оцінки діагностуємих показників. Для забезпечення зазначених вимог до діагностичної моделі виконується оптимізація простору ознак X.

Структура експериментальних даних відображається за допомогою двох основних категорій взаємодії між елементами ТЕД – категорій подібності і розходження. Подібність і розходження ознак визначається мірами зв'язку, а об'єктів ТЕД – мірами близькості (відстаней). Матриця зв'язку задає відношення "ознака – ознака" і являє собою двовимірну симетричну квадратну матрицю розміром $p \times p$.

$$\boldsymbol{R} = \begin{vmatrix} r_{11} & r_{12} & \cdots & r_{1p} \\ r_{21} & r_{22} & \cdots & r_{2p} \\ \cdots & \cdots & \cdots & \cdots \\ r_{p1} & r_{p2} & \cdots & r_{pp} \end{vmatrix}, \qquad (1.29)$$

де r_{ij} – міра зв'язку між ознаками x_i і x_j.

У роботі [5] виділені дві представницькі групи зв'язку між ознаками, засновані на принципі коваріації і на принципі спряженості ознак.

Виходячи з першого принципу, висновок про наявність зв'язку між змінними робиться в тому випадку, коли збільшення значення однієї змінної супроводжується стійким збільшенням або зменшенням значень іншої. У математичному виразі задача зводиться до обчислення коваріації, тобто до супутньої зміни чисельних значень ознак.

На принципі коваріації заснований коефіцієнт парної кореляції Пірсона (r_{kj}^{p}), що є мірою лінійного зв'язку двох змінних: x_k і x_j. Він обчислюється за формулою:

$$r_{kj}^{p} = s_{kj} \Big/ \sqrt{s_{kk}\, s_{jj}}\,, \qquad (1.30)$$

де $s_{kj} = \frac{1}{N-1}\sum_{i=1}^{N}(x_{ik} - m_k)(x_{ij} - m_j)$, $m_j = \frac{1}{N}\sum_{i=1}^{N} x_{ij}$ є елементами коваріаційної матриці генеральної сукупності, з якої витягнуті об'єкти x_i і x_j .

Критерієм значимості коефіцієнта кореляції служить критерій Стьюдента t для нормального розподілу значень випадкових величин.

Перевірка гіпотези про відсутність статистичного зв'язку H_0 між x_k та x_j виконується шляхом перевірки нерівностей

H_0: $r = 0$ при $\lvert t_0 \rvert < t_{1-\alpha/2\,(n\text{-}2)}$, H_1 : $r \neq 0$ при $\lvert t_0 \rvert > t_{1-\alpha/2\,(n\text{-}2)}$.	(1.31)

В нерівностях (1.31) розрахунковий критерій Стьюдента t_0 обчислюється за формулою

$$t_0 = \frac{\tilde{r}\sqrt{n-2}}{\sqrt{1-\tilde{r}^2}}$$

та порівнюється з табличним значенням критерію Стьюдента $t_{1-\alpha/2\,(n\text{-}2)}$ з рівнем значимості α (зазвичай 0,05 або 0,01) при n–2 ступенях свободи (n – обсяг навчальної вибірки).

Для інших типів ознак застосовуються інші міри зв'язку, які по суті, є

алгебраїчним перетворенням коефіцієнта кореляції Пірсона r_{kj}^{p}, і враховують тип ознак, що зіставляються. Для аналізу рангових ознак застосовується коефіцієнт рангової кореляції Спірмена, що визначається за формулою:

$$r_{kj}^{s} = 1 - 6 \frac{\sum_{i=1}^{N}(r_{ji} - r_{ki})}{N(N^2 - 1)},$$

де r_{ji} і r_{ki} – ранги ознак x_j і x_k відповідно.

Крім того, для рангових змінних використовується міра зв'язку, заснована на підрахунку числа розбіжностей у ранжируванні об'єктів ("тау" Кендалла):

$$\tau = \frac{P - Q}{N(N-1)/2},$$

де P – число збігів, а Q – число розбіжностей порядків з $N(N–1)/2$ пар ознак.

Незважаючи на розходження в підходах, між коефіцієнтами рангової кореляції Спірмена і Кендалла існує тісний логічний зв'язок, і в більшості випадків вони дають однакові результати.

Друга велика група мір зв'язку, заснована на принципі взаємної спряженості, спрямована на з'ясування наступного факту: чи з'являються деякі значення однієї ознаки одночасно з визначеними значеннями іншої частіше, ніж при випадковій вибірці?

Загальним для першої і другої груп є коефіцієнт спряженості φ, що призначений для виміру зв'язку двох дихотомічних ознак: $x_j = \{0,1\}$ і $x_i = \{0,1\}$. Для обчислення коефіцієнта φ будується чотириелементна таблиця спряженості, структура якої показана в табл. 1.2.

Таблиця 1.2 – Таблиця спряженості дихотомічних ознак

$x_i \setminus x_j$	0	1	S
0	a	b	$a + b$
1	c	d	$c + d$
S	$a + c$	$b + d$	

Елементами таблиці спряженості (a, b, c, d) є кількості об'єктів з відповідними комбінаціями значень дихотомічних ознак x_i і x_j (a – 0, 0; b – 0, 1; c – 1, 0; d – 1, 1). При цьому $a + b + c + d = N$. Тут же підраховуються відповідні суми (по рядках і по стовпцях).

Коефіцієнт φ являє собою алгебраїчне спрощення коефіцієнта кореляції Пірсона r_{kj}^{p}, з урахуванням специфіки дихотомічних ознак і обчислюється за

формулою:

$$\varphi = \frac{ad - bc}{\sqrt{(a+c)(b+d)(a+b)(c+d)}} \qquad (1.32)$$

Коефіцієнти спряженості будуються не тільки для дихотомічних ознак по чотириелементній таблиці, але й у більш складних випадках. У загальному випадку будується $(m \times m)$-елементна таблиця спряженості при розбивці діапазону зміни ознак на m частин і при порівнянні їхніх значень з $m - 1$ порогами. У такий спосіб коефіцієнти спряженості можна застосовувати для аналізу зв'язків між чисельними, ранговими і дихотомічними ознаками і їх комбінаціями.

При виборі тієї або іншої міри зв'язку для розв'язання конкретної задачі в роботі [4] відзначається, що застосування до тих самих даних різних мір зв'язку нерідко приводить до результатів, що відрізняються. Це обумовлено тим, що математики, які конструювали коефіцієнти зв'язків, як правило, досліджували їхні властивості в граничних ситуаціях – близько 0 або 1. Поводження ж різних мір зв'язку усередині інтервалу [0, 1] порівняно мало вивчене. Тому на практиці вибір якої-небудь міри зв'язку визначається особистими симпатіями дослідника.

Розглянуті вище міри зв'язків між ознаками вказують лише на наявність/відсутність лінійного зв'язку між двома окремими ознаками. При цьому слід зазначити, що коефіцієнт кореляції (і його алгебраїчні спрощення для інших типів ознак) має чіткий сенс як характеристика ступеня тісноти зв'язку тільки у випадку спільного нормального розподілу досліджуваних ознак. Наявність кореляційного зв'язку між ознаками не є однозначною вказівкою на їхню взаємозумовленість (можливі випадки "помилкової" кореляції). З іншого боку, некорельованість не може служити однозначною вказівкою на відсутність зв'язку між ознаками, вона лише вказує на відсутність лінійної залежності між ними.

Більшість методів синтезу діагностичних моделей припускають роботу з незалежною системою ознак (чого практично не буває при прийнятих методиках вимірювань), або в крайньому випадку дають прийнятні результати при слабозв'язаній системі ознак. Тому аналіз зв'язків між ознаками є необхідним етапом обробки експериментальних даних з метою ухвалення рішення про перетворення простору ознак.

Матриця близькостей (віддаленостей) задає відношення "об'єкт – об'єкт" і являє собою квадратну симетричну матрицю $N \times N$ з невід'ємними елементами (1.33). Елементи d_{ij} є значеннями деякої міри близькості (віддаленості або відстані) між об'єктами x_i і x_j у заданому просторі ознак (вихідному або перетвореному). Найчастіше при аналізі даних використовуються міри віддаленості

$$\boldsymbol{D} = \begin{vmatrix} d_{11} & d_{12} & \cdots & d_{1N} \\ d_{21} & d_{22} & \cdots & d_{2N} \\ \cdots & \cdots & \cdots & \cdots \\ d_{N1} & d_{N2} & \cdots & d_{NN} \end{vmatrix}. \qquad (1.33)$$

До цих мір пред'являються наступні вимоги [1,4]: максимальна подібність об'єкта із самим собою $d_{ij} = \min d_{ij}$; вимога симетрії $d_{ij} = d_{ji}$; виконання нерівності трикутника $d_{ij} \le d_{ik} + d_{kj}$.

В задачі класифікації об'єктів матриця близькості, або відстаней $\boldsymbol{D}$, між об'єктами будується у просторі ознак, де для всіх об'єктів навчальної вибірки визначена належність до одного із заданої множини класів. Матриця близькості показує, наскільки в даному просторі ознак один клас віддаляється від іншого. Для синтезу діагностичних вирішальних правил (класифікації об'єктів) необхідне виконання гіпотези компактності: об'єкти, що належать одному класові, повинні розташовуватися в просторі ознак компактними групами.

Найбільш розповсюдженими мірами (відстанями) між об'єктами A_i і A_j є такі [5]: звичайна d_{ij}^{E} і зважена $d_{ij}^{\%E}$ евклідова відстань; відстань Махаланобіса d_{ij}^{M}; узагальнена відстань Мінковського d_{ij}^{MI}; відстань Хеммінга d_{ij}^{H}. Вказані міри визначаються таким чином:

$$d_{ij}^{E} = \sqrt{\sum_{k=1}^{p} (x_{ik} - x_{jk})^2}, \quad d_{ij}^{\%E} = \sqrt{\sum_{k=1}^{p} w_k (x_{ik} - x_{jk})^2},$$

$$d_{ij}^{M} = \sqrt{(\bar{x}_i - \bar{x}_j)^T S^{-1} (\bar{x}_i - \bar{x}_j)}, \; d_{ij}^{MI} = \sqrt[q]{\sum_{k=1}^{p} (x_{ik} - x_{jk})^q}, \; d_{ij}^{H} = \sum_{k=1}^{p} \left| x_{ik} - x_{jk} \right|,$$

де x_{ik}, x_{jk} – координати (ознаки) i і j об'єктів у просторі ознак X відповідно; w_k – ваги ознак; p – розмірність простору ознак X; $\bar{x}_i$, $\bar{x}_j$ – вектори ознак i та j об'єктів у просторі ознак X відповідно; S – коваріаційна матриця генеральної сукупності, з якої витягнуті об'єкти x_i і x_j.

Евклідова відстань застосовується для обчислення відстані між об'єктами, описаними кількісними, якісними і дихотомічними ознаками. Її використання доцільне, коли ознаки однорідні за семантичним навантаженням й однаково важливі для розв'язуваної задачі.

Зважена евклідова відстань використовується, коли необхідно кількісно

виразити важливість яких-небудь ознак або коли треба вирівняти масштаби неоднорідних ознак.

Відстань Махаланобіса застосовується при сильній залежності і неоднорідності досліджуваних ознак, тому що вона інваріантна до лінійних перетворень простору ознак (до зміни масштабу і повороту осей).

Узагальнена відстань Мінковского є універсальною мірою близькості. При $q = 1$ вона переходить у Хеммінгову відстань для дихотомічних ознак або в "міську метрику" для ординальных ознак; при $q = 2$ – у евклідову; при $q \to \infty$ – у так називану супремум-норму $d_{ij}^{(q)} = \max_k \left| x_{ik} - x_{jk} \right|$. Але в даний час "…не відомі приклади її використання при довільних $q \neq 1, 2, \infty$ …" [4].

Відстань Хеммінга найчастіше використовується для визначення розходжень між об'єктами, що задаються дихотомічними ознаками, й інтерпретується як число розбіжностей значень ознак x_i і x_j у розглянутих об'єктів A_i і A_j.

1.2.3. Методи зниження розмірності простору ознак

Пошук найбільш інформативної системи діагностичних ознак виконується безліччю формальних методів, об'єднаних загальним поняттям "аналіз даних". При цьому відбувається агрегування (стиск) вихідного простору ознак з метою зведення його до компактного і доступного для огляду вигляду при подальшому дослідженні. Існує ряд підходів до розв'язання даної задачі, серед яких можна умовно виділити дві групи методів:

– зниження розмірності простору ознак шляхом заміни значної кількості вихідних ознак невеликим числом інтегральних (узагальнених) ознак, що зберігають достатню інформацію про досліджувані об'єкти. Дана група методів заснована на дослідженні автоінформативності вихідного простору ознак. Інтегральні ознаки є лінійною комбінацією вихідних ознак. До зазначеної групи методів відносяться метод головних компонент, факторний аналіз, багатовимірне шкалювання та ін.;

– зниження розмірності простору ознак шляхом оцінки відносної значимості окремих ознак при розв'язанні класифікаційної задачі і виділення підпростору значимих ознак. У даній групі методів використовується "зовнішній" критерій інформативності – вплив окремої ознаки (або групи ознак) на властивість, що діагностується. Обчислення "зовнішнього" критерію може бути виконане методами дисперсійного аналізу, кореляційного аналізу, на основі теоретико-інформаційного підходу. При використанні будь-якого "зовнішнього" критерію можна застосовувати різні алгоритми зниження розмірності (повний перебір; *k* кращих; методи послідовного зменшення і збільшення простору ознак та ін.).

Перша група методів застосовується за наявності значного зв'язку (корельованності) між окремими ознаками вихідного простору, а друга група методів припускає роботу з незалежною або слабозв'язаною системою ознак.

Важливою задачею "аналізу даних" є кластеризація ознак, тобто формування з вихідної множини ознак m підмножин (кластерів). Кластеризація ознак знижує розмірність задачі і дозволяє застосовувати методи зниження розмірності усередині кожного кластера. Для кластеризації ознак застосовують методи кластерного аналізу, кореляційних плеяд та ін.

Розглянемо більш докладно відзначені вище методи й особливості їхнього застосування до експериментальних даних.

Кластерний аналіз [1] заснований на обчисленні відстаней між об'єктами в просторі ознак і формуванні m кластерів (значення m задане або визначається алгоритмами), усередині яких об'єкти мають максимальну подібність (мінімальна відстань), у той час як між кластерами спостерігається різнорідність (максимальна відстань). Відомі такі методи кластерного аналізу: одиночних і повних зв'язків, попарне середнє, центроїдний, Варда, k-середнього [1, 6]. Відмінність названих методів полягає в способі обчислення відстані між кластерами, а також – у виді цільової функції.

Нижче перераховані найбільш відомі методи кластерного аналізу.

Метод повних зв'язків (найбільш віддалених сусідів). Суть даного методу полягає в тім, що два об'єкти, які належать тому самому кластеру, мають коефіцієнт подібності, який менший від деякого граничного значення. У термінах евклідової відстані $D_{ij}^{(E)}$ це означає, що відстань між двома точками кластера не повинна перевищувати деяке граничне значення h. Таким чином, значення h визначає максимально припустимий діаметр підмножини, що утворить кластер. При цьому процедуру кластеризації можна розглядати як одержання графа, у якому ребра з'єднують усі вершини в групу, тобто кожна група утворить повний підграф. При цьому відстань між групами K_s і K_t визначається найбільш віддаленими вершинами в цих групах:

$$R_{st} = \max_{i \in K_s,\, j \in K_t} D_{ij} .$$

Метод одиночних зв'язків (метод найближчого сусіда). Кожний об'єкт розглядається як одноточковий кластер. Об'єкти групуються за таким правилом: два кластери поєднуються, якщо відстань між самими ближніми точками одного кластера K_s і точками іншого K_t мінімальна:

$$R_{st} = \min_{i \in K_s,\, j \in K_t} D_{ij} .$$

Таким чином, найближчі сусіди визначають найближчі підмножини. Процедура складається з (n-1) кроків і, виходячи з позиції теорії графів, коли

об'єкти розглядаються як вершини графа, полягає в знаходженні мінімального покриваючого дерева.

Метод Ворда. У цьому методі як цільову функцію застосовують внутрішньогрупову суму квадратів відхилень, яка є не чим іншим, як сумою квадратів відстаней між кожною точкою і центром кластера, який містить цю точку. На кожному кроці поєднуються такі два кластери, які приводять до мінімального збільшення цільової функції, тобто внутрішньогрупової суми квадратів. Цей метод спрямований на об'єднання близько розташованих кластерів.

Центроїдний метод (*k*-середніх). Відстань між двома кластерами K_s і K_t визначається як евклідова відстань між центрами (зваженими середніми за кожним показником) цих кластерів:

$$R_{st} = D^{(E)}(\overline{x_{K_s}}, \overline{x_{K_t}}),$$

де $\overline{x_{K_s}}$ – середнє арифметичне векторних спостережень x_i, при $i \in K_s$.

Кластеризація йде поетапно. На кожному з (n–1) кроків поєднують два кластери (K_s і K_t), що мають мінімальне значення R_{st}. При цьому на нульовому кроці за центри шуканих k кластерів приймають випадково обрані k спостережень — точки $x_1, x_2, \ldots, x_k$. Якщо при обчисленнях використовуються ваги для врахування різниці між розмірами кластерів (тобто кількістю точок у них), то цей метод називають ще методом зважених груп.

Метод попарного середнього. У цьому методі відстань між двома різними кластерами K_s і K_t, обсягом p_s і p_t об'єктів відповідно, обчислюється як середня відстань між усіма парами об'єктів у них:

$$R_{st} = \frac{1}{p_s p_t} \sum_{\substack{i \in K_s \\ j \in K_t}} D_{ij}.$$

У випадку нерівних розмірів кластерів враховується розмір відповідних кластерів як ваговий коефіцієнт, і тоді даний метод називають зваженим попарним середнім.

Усі розглянуті вище методи є методами, спрямованими на послідовне об'єднання одиночних груп.

Ієрархічне групування. Суть даного методу полягає у послідовному об'єднанні (алгомеративні процедури) або поділі (дивізимні процедури) кластерів. При послідовному об'єднанні на першому кроці кожен об'єкт вважається окремим кластером, і на кожному наступному кроці два найближчих кластери поєднуються в один доти, поки всі об'єкти не об'єднаються в один кластер. При послідовній розбивці, навпаки, всі об'єкти на першому кроці належать одному кластерові, а на кожному наступному

кроці з кластера відокремлюються об'єкти, найбільш віддалені від центра кластера. В обох процедурах як міра близькості між об'єктами використовується відстань у просторі ознак. При цьому оптимальну розбивку вибирає сам дослідник шляхом аналізу так званої дендограми, побудованої за результатами групування на всіх кроках алгоритму з урахуванням шкали подібності.

Зазначені методи кластерного аналізу призначені для кластеризації об'єктів, використовують координати об'єктів у просторі ознак і їхнє безпосереднє застосування до задачі кластеризації ознак неможливе (між ознаками можна визначити відстань, але немає координат), однак при використанні деяких прийомів (обчислення фіктивних координат ознак; транспонування ТЕД, у результаті чого об'єкти стають фіктивними координатами, а ознаки – фіктивними об'єктами відповідно) методи кластерного аналізу використовуються для кластеризації ознак.

Метод кореляційних плеяд. У даному методі [1] структуру зв'язків між діагностичними ознаками зображають у вигляді графа, де кожна вершина відповідає одній з ознак, а ребра вказують на кореляційний зв'язок між ознаками. Метод призначений для синтезу таких груп ознак – "плеяд", для яких зв'язок усередині плеяд досить великий, а між ними – малий. Задаючись деякими граничними значеннями коефіцієнта кореляції, виключають з вихідного графа всі ребра з коефіцієнтом кореляції, який по модулю менший від граничного. Поетапно збільшуючи граничне значення коефіцієнта кореляції, повторюють цю процедуру до розбивки графа на декілька підграфів. Вершини кожного підграфа й утворять плеяду. Для отриманих у такий спосіб плеяд, коефіцієнти кореляції всередині плеяд будуть більші від останнього граничного значення, а між ними – менші. Однак даний метод аналізує тільки окремі зв'язки між вершинами, а не їхню сукупність, що не завжди приводить до оптимальної розбивки, тому він використовується в основному для наочності зображення структури зв'язків між ознаками.

Метод головних компонентів здійснює перехід до нової системи координат y_1, ..., y_p у вихідному просторі ознак x_1, ..., x_p яка є системою ортонормованих лінійних комбінацій [1].

$$\begin{cases} y_j(x) = w_{1j}(x_1 - m_1) + \cdots + w_{pj}(x_p - m_p); \\ \sum_{i=1}^{p} w_{ij}^2 = 1 \to (j = \overline{1,p}); \\ \sum_{i=1}^{p} w_{ij} w_{ik} = 0 \to (j,k = \overline{1,p}, j \neq k); \end{cases} \tag{1.34}$$

де m_i – математичне очікування ознаки x_i.

Метод головних компонентів заснований на виділенні лінійних комбінацій випадкових величин, що мають максимально можливу дисперсію, за допомогою матриці парних кореляцій [1]. Лінійні комбінації вибираються таким чином, що серед всіх можливих лінійних нормованих комбінацій вихідних ознак перший головний компонент $y_1(x)$ має найбільшу дисперсію. Геометрично це виглядає як орієнтація нової координатної осі y_1 уздовж напрямку найбільшої витягнутості еліпсоїда розсіювання об'єктів досліджуваної вибірки в просторі ознак $x_1, ..., x_p$. Другий головний компонент має найбільшу дисперсію серед всіх лінійних перетворень, що залишилися, не корельованих з першим головним компонентом. Він інтерпретується як напрямок найбільшої витягнутості еліпсоїда розсіювання, перпендикулярний першому головному компоненту. Наступні головні компоненти визначаються за аналогічною схемою. У зв'язку з цим з'являється можливість виразити інформацію, що утримується у великому наборі вихідних ознак, за допомогою меншого числа незалежних головних компонентів. Метод застосовується для системи нормованих числових ознак.

Обчислення коефіцієнтів головних компонентів w_{ij} засновано на тому факті, що вектори $w_i = (w_{11}, ..., w_{pl})', ..., w_p = (w_{1p}, ..., w_{pp})'$ є власними (характеристичними) векторами кореляційної матриці S. У свою чергу, відповідні власні числа цієї матриці дорівнюють дисперсіям проекцій множини об'єктів на осі головних компонентів.

Алгоритми, що забезпечують виконання методу головних компонентів, входять практично в усі пакети статистичних програм.

Факторний аналіз [1] заснований на припущенні, що вихідні ознаки є проявами невеликого числа об'єктивно існуючих, але таких, що не піддаються безпосередньому вимірові факторів, що детермінують розходження між об'єктами. В описаному вище методі головних компонентів під критерієм автоінформативності простору ознак розуміють те, що цінну для діагностики інформацію можна відобразити в лінійній моделі, яка відповідає новій координатній осі в даному просторі з максимальною дисперсією розподілу проекцій досліджуваних об'єктів. Такий підхід є продуктивним, коли явна більшість завдань «чорнового» варіанта тесту узгоджено «працює» на прояв властивості, що тестується й придушує вплив ірелевантних факторів на розподіл об'єктів. Також позитивний результат буде отриманий при порівняно невеликому обсязі групи зв'язаних інформативних ознак, але при неузгодженій взаємодії сторонніх факторів, під впливом яких не порушується однорідність еліпсоїда розсіювання, а лише зменшується витягнутість розподілу об'єктів уздовж напрямку тенденції, що діагностується. На відміну від методу головних компонентів, факторний аналіз заснований не на дисперсійному критерії автоінформативності системи ознак, а він орієнтований на пояснення наявних між ознаками

кореляцій. Тому факторний аналіз застосовується в більш складних випадках спільного прояву на структурі експериментальних даних тестуємої й іррелевантної властивостей об'єктів, порівнянних за ступенем внутрішньої погодженості, а також для виділення групи діагностичних показників із загальної вихідної множини ознак. Область застосування методу – система числових ознак. Основна модель факторного аналізу записується такою системою рівностей

$$x_i = \sum_{j=1}^{m} l_{ij} f_j + \varepsilon_i \,, \qquad (1.35)$$

де $i = 1, ..., p$ – кількість ознак; $m<p$ – число факторів; l_{ij} – навантаження i-ї ознаки на j-й фактор.

Таким чином припускається, що значення кожної ознаки x_i може бути виражене зваженою сумою латентних змінних (простих факторів) f_j, кількість яких менша від числа вихідних ознак, і залишковим членом ε_i (специфічним фактором) з дисперсією σ^2, що діє тільки на x_i.

Коефіцієнти l_{ij} називаються навантаженням i-ї змінної на j-й фактор або навантаженням j-го фактора на i-ту змінну. У найпростішої моделі факторного аналізу вважається, що фактори f_j взаємно незалежні і їхні дисперсії дорівнюють одиниці, а випадкові величини ε_i теж незалежні одне від одної та від будь-якого фактора f_j. Максимально можлива кількість факторів m при заданому числі ознак p визначається нерівністю

$$(p+m)<(p-m)^2,$$

яка повинна виконуватися, щоб задача не вироджувалася в тривіальну.

Дана нерівність одержана на підставі підрахунку ступенів свободи, що є у задачі. Суму квадратів навантажень у формулі основної моделі факторного аналізу називають спільністю відповідної ознаки x_i і чим більше це значення, тим краще описується ознака x_i виділеними факторами f_j. Спільність є частиною дисперсії ознаки, яку пояснюють фактори. У свою чергу, величина ε_i^2 показує, яка частина дисперсії вихідної ознаки залишається непоясненою при використовуваному наборі факторів, що використовується, і дану величину називають специфічністю ознаки. Таким чином,

$$\textit{Дисперсія\_ознаки} = \textit{спільність}\left(\sum_{j-1}^{m} l_{ij}^2\right) + \textit{специфічність}\left(\varepsilon_i^2\right).$$

Основне співвідношення факторного аналізу (1.35) показує, що коефіцієнт кореляції будь-яких двох ознак x_i і x_j можна виразити сумою добутку навантажень некорельованих факторів

$$r_{ij} = r(x_i, x_j) = l_{i1} l_{j1} + l_{i2} l_{j2} + \cdots + l_{im} l_{jm}.$$

Задачу факторного аналізу не можна розв'язати однозначно. Рівності основної моделі факторного аналізу (1.35) не піддаються безпосередній перевірці, тому що p вихідних ознак задається через (p+m) інших змінних – простих і специфічних факторів. Тому представлення кореляційної матриці факторами (факторизацію) можна зробити нескінченно великим числом способів. Якщо вдалося зробити факторизацію кореляційної матриці за допомогою деякої матриці факторних навантажень F, то будь-яке лінійне ортогональне перетворення F (ортогональне обертання) приведе до такого ж результату.

Існуючі програми обчислення навантажень починають працювати з однофакторної моделі (m = 1). Потім перевіряється, наскільки кореляційна матриця, відновлена за однофакторною моделлю відповідно до виразу (1.35), відрізняється від кореляційної матриці вихідних даних. Якщо однофакторна модель визнається незадовільною, то випробовується модель із m = 2 і т. д. Це триває доти, поки при деякому m не буде досягнута адекватність або поки число факторів у моделі не перевищить максимально припустиме. В останньому випадку говорять, що адекватної моделі факторного аналізу не існує. Якщо факторна модель існує, то виконується обертання отриманої системи загальних факторів, тому що значення факторних навантажень і навантажень на фактори є лише одним з можливих розв'язків основної моделі. Обертання факторів може виконуватися різними способами. Найбільш часто воно здійснюється таким чином, щоб якомога більше число факторних навантажень стало нулями й щоб кожний фактор по можливості описував групу сильно корельованих ознак. Також можна обертати фактори доти, поки не будуть одержані результати, які піддаються змістовній інтерпретації. Можна, наприклад, забажати, щоб один фактор був навантажений переважно ознаками одного типу, а інший – ознаками іншого типу. Або, скажемо, можна забажати, щоб зникли якісь важко інтерпретуємі навантаження з негативними знаками. Нерідко дослідники йдуть далі й розглядають прямокутну систему факторів як окремий випадок косокутної, тобто заради змісту жертвують умовою некорельованості факторів.

На завершення всієї процедури факторного аналізу за допомогою математичних перетворень виражають фактори f_j через вихідні ознаки, тобто одержують у явному вигляді параметри лінійної діагностичної моделі.

Відома велика кількість методів факторного аналізу (ротацій, максимальної правдоподібності й ін.). Нерідко в тому самому пакеті програм аналізу даних реалізовано відразу кілька версій таких методів, і тому виникає правомірне питання про те, який з них кращий.

Метод контрастних груп [1]. Вихідною інформацією при використанні методу контрастних груп, крім таблиці експериментальних даних з результатами обстеження «чорновим» варіантом діагностичного тесту, є

також «чорнова» версія лінійного правила обчислення показника, що тестується. Ця «чорнова» версія може бути складена експериментатором, виходячи з його теоретичних уявлень про те, які ознаки й з якими вагами повинні бути включені в лінійну діагностичну модель. Крім того, «чорнова» версія може бути почерпнута з літературних джерел, коли в експериментатора виникає потреба адаптувати опублікований діагностичний тест до нових умов. Метод контрастних груп застосовується також у складі процедури підвищення внутрішньої погодженості завдань раніше відпрацьованого тесту.

В основі методу контрастних груп лежить гіпотеза про те, що значна частина «чорнової» версії діагностичної моделі підібрана або вгадана правильно. Тобто в праву частину рівняння $y_ч = y_ч(x)$ увійшло досить багато ознак, що узгоджено відображають тестуєму властивість, що тестується. У той же час в «чорновій» версії $y_ч(x)$ певна частка ознак є непотрібною або навіть шкідливим баластом, від якого потрібно позбутися. Як і у всіх інших методах, що спираються на категорію внутрішньої погодженості, це означає, що в просторі ознак, включених у вихідну діагностичну модель, розподіл об'єктів уписується в еліпсоїд розсіювання, витягнутий уздовж напрямку діагностуємої тенденції. У свою чергу, вплив інформаційного баласту виражається в зменшенні такої витягнутості еліпсоїда розсіювання, тому що «шумові» ознаки збільшують розкид досліджуваних об'єктів по всіх інших напрямках. При цьому «зашумлення» основної тенденції буде тим сильніше, чим ближче до центра розподілу розташуються діагностуємі об'єкти, і тим слабкіше, чим ближче до полюсів головної осі еліпсоїда розсіювання перебуватимуть розглянуті об'єкти. Це пов'язане з тим, що попадання об'єктів у крайні області пояснюється, головним чином, кумулятивним ефектом погодженої взаємодії інформативних ознак. Описані уявлення про структуру експериментальних даних лежать в основі наступної процедури.

Спочатку призначаються вихідні шкальні ключі (ваги) w°_j для пунктів тесту (дихотомічних ознак) x_j. Для кожного i-го випробування підраховується сумарний тестовий бал

$$y_ч(x) = \sum_{j=1}^{p} w^\circ_j x_j \,. \qquad (1.36)$$

Зазвичай абсолютні значення ваг w_j визначають приблизно й часто беруть такими, що дорівнюють одиниці. Тому напрямок (1.36) буде трохи відрізнятися від напрямку головної діагоналі еліпсоїда розсіювання $y(x)$ (рис. 1.4).

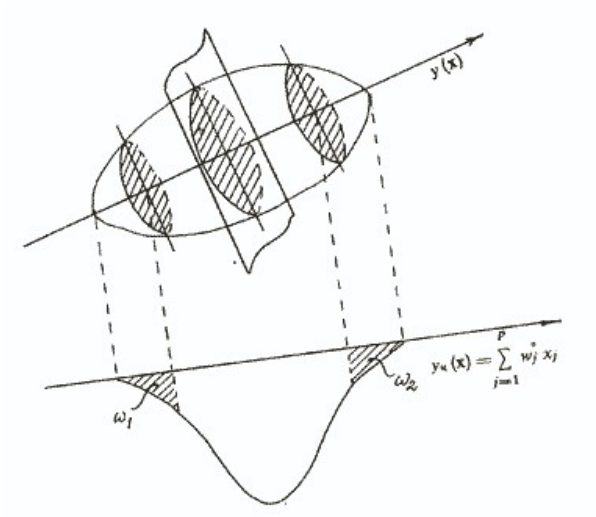


Рис. 1.4. Ілюстрація методу контрастних груп

Але якщо орієнтовно $y_ч(x)$ правильно відображає діагностуєму властивість, то на краях розподілу сумарного бала, побудованого по всіх об'єктах досліджуваної вибірки, можна виділити контрастні групи ω_1 і ω_2, у які ввійдуть об'єкти з мінімальними похибками, внесеними «шумовими» ознаками. Ці групи не повинні бути занадто малі. Для нормального розподілу, як правило, беруть контрастні групи обсягом 27 % від загального обсягу вибірки, для більше плоского – 33 %. У принципі вважається прийнятною будь-яка цифра від 25 до 33 %.

Наступний крок полягає у визначенні ступеня зв'язку кожного пункту з дихотомічної змінної – номером контрастної групи. Мірою цього зв'язку може служити так званий коефіцієнт розрізнення, що являє собою різницю відсотків тієї або іншої відповіді на аналізований пункт у полярних групах випробуваних. Найбільше часто використовується коефіцієнт зв'язку Пірсона φ, який потім порівнюється із граничним значенням

$$\left|\varphi_{гр}\right| = \sqrt{\chi^2{}_{гр} / N} ,$$

де $\chi^2{}_{гр}$ – стандартний квантиль розподілу χ_2 з одним ступенем свободи.
Зазвичай орієнтуються на рівні значимості 5 % і 1 % , для яких значення χ^2 дорівнює 3,84 і 6,63 відповідно. Якщо для *i*-го пункту $|\varphi_i|<|\varphi_{гр}|$, то ваговому коефіцієнту w_i привласнюється значення нуля, тобто ознака x_i виключається з лінійної діагностичної моделі $y_ч(x)$. У такий спосіб перевіряються всі пункти «чорнового» варіанта тесту. Потім для пунктів, що залишилися, вся

процедура знову повністю повторюється й т. д.

На практиці не зустрічаються випадки, коли остаточно відібрані за допомогою наведеної процедури інформативні ознаки абсолютно б збіглися з початково заданими. Збіжність цієї процедури залежить від вихідного співвідношення «гарних» і «поганих» завдань тесту. Очевидно, для діагностичних моделей, заснованих на принципі внутрішньої погодженості ознак, у кожному конкретному завданні існує певний поріг співвідношення інформативних і «шумових» ознак, починаючи з якого можливе виникнення ефекту самоорганізації або самовдосконалення діагностичної моделі за допомогою описаного вище алгоритму.

Багатовимірне шкалювання [1] – сукупність методів, які дозволяють за заданою інформацією про міри розходження (близькості) між об'єктами приписувати кожному з цих об'єктів вектор його кількісних характеристичних показників. При цьому розмірність шуканого координатного простору задається заздалегідь, а "занурення" у нього аналізованих об'єктів виконується таким чином, щоб структура взаємних розходжень між ними, яка вимірюється за допомогою приписуваних їм допоміжних координат, у середньому найменш відрізнялася б від заданої в сенсі того або іншого функціоналу якості. Визначення координат об'єктів у просторі і розмірності простору засноване на перетворенні матриці відстаней у матрицю скалярних добутків центрованих векторів. Таким чином, на виході алгоритму виходять числові значення координат, що приписуються кожному об'єктові в деякій новій системі координат (у "допоміжних шкалах", зв'язаних з латентними змінними), причому розмірність нового простору ознак істотно менша від розмірності вихідного. При функціональному шкалюванні [1] будується єдиний (інтегральний) показник. Методи багатовимірного шкалювання застосовуються в ряді випадків, коли більшість методів факторизації є непридатною.

Дисперсійний аналіз [1, 3] був розроблений у 20-х роках XX сторіччя англійським математиком і генетиком Рональдом Фішером. На дисперсійному аналізі заснований широкий клас критеріїв значущості. Дисперсійний аналіз дозволяє визначити вплив різних факторів (умов) x_i на досліджувану ознаку (явище) y, що досягається шляхом розкладання сукупної дисперсії D_y на окремі компоненти, викликані впливом різних джерел мінливості. При цьому перевіряється тільки наявність або відсутність статистично значущого зв'язку x_i з y, тому метод застосовується для попередньої оцінки і добору значущих факторів, і для відсівання всіх інших.

Вхідні змінні x_i представляються номінальними, ординальними або дихотомічними шкалами, вихідна ознака y зазвичай є числовою. При аналізі числових факторів x_i, їх необхідно звести до ординальної шкали, виконуючи квантування динамічного діапазону Δx на p рівнів.

Для незалежної (або слабозалежної) системи факторів застосовується однофакторний дисперсійний аналіз, а для врахування сумісної дії факторів – двофакторний дисперсійний аналіз.

Однофакторний дисперсійний аналіз. Ідея методу заснована на оцінці вибіркових середніх m_y і дисперсій D_y по вибірці. Якщо вибірки отримані при різних значеннях рівнів факторів (дві групи при бінарних факторах, p груп при номінальних або ординарних ознаках, що мають p градацій), то виникає таке питання: чи розходження в оцінках m_y і D_y викликані впливом факторів, або вони випадкові? Використовуються методи оцінки статистичної значущості розходжень (їх називають критеріями значущості, або просто критеріями). Методів цих існує безліч, але усі вони побудовані на одному принципі. Спочатку формулюється нульова гіпотеза H_0, тобто припускається, що фактори x_i, які досліджуються, не роблять ніякого впливу на досліджувану величину y, й отримані розходження оцінок випадкові і обумовлені обмеженістю вибірки. Потім ми визначаємо, яка імовірність одержати розходження, що спостерігаються (або більш сильні) за умови справедливості нульової гіпотези. Якщо ця імовірність менша від заданого порогового значення, то нульова гіпотеза відкидається і робиться висновок, що результати експерименту статистично значущі (приймається гіпотеза H_1). Це ще не означає, що ми довели дію саме досліджуваних факторів (це питання насамперед планування експерименту), але, у всякому разі, малоймовірно, що результат обумовлений випадковістю.

Інтуїтивно зрозуміло, що вибірки «не розрізняються», якщо розкид вибіркових середніх значно менший від розкиду значень у кожній з вибірок, і навпаки – вибірки «розрізняються», якщо розкид вибіркових середніх перевищує розкид у кожній з вибірок. Залишилося тільки формалізувати це судження та оформити його кількісно.

Дослідник не може спостерігати генеральну сукупність, з якої взяті вибірки. Усе, що він може аналізувати – це його експериментальні групи. Дисперсію сукупності D_y можна оцінити двома способами.

По-перше, дисперсія, обчислена для кожної групи $D_{yi}\ i=\overline{1,p}$, – це оцінка дисперсії сукупності D_y. Тому дисперсію сукупності D_y можна оцінити на підставі групових дисперсій D_{yi}. Така оцінка не буде залежати від розходжень групових середніх.

По-друге, розкид вибіркових середніх теж дозволяє оцінити дисперсію сукупності D_y. Зрозуміло, що така оцінка дисперсії залежить від розходжень вибіркових середніх.

Якщо експериментальні групи – це p випадкові вибірки з тієї самої нормально розподіленої сукупності (це означає, що фактор не впливає на y), то обидві оцінки дисперсії сукупності D_y дали б приблизно однакові результати. Тому, якщо ці оцінки виявляються близькими, то ми не можемо

відкинути нульову гіпотезу. У противному разі ми відкидаємо нульову гіпотезу H_0, тобто вважаємо малоймовірним те, що ми одержали би такі розходження між групами, якби вони були просто випадковими вибірками з однієї нормально розподіленої сукупності.

Перейдемо до обчислень. Як оцінити дисперсію сукупності D_y за p вибірковими дисперсіями D_{yi}? Якщо вірна гіпотеза H_0 про те, що x не впливає на величину y, то кожна з них дає однаково гарну оцінку. Тому за оцінку дисперсії сукупності D_y візьмемо середнє вибіркових дисперсій D_{yi}. Ця оцінка називається внутрішньогруповою дисперсією, що позначається як D_R

$$D_R = \frac{\sum_i D_{yi}}{p}. \tag{1.37}$$

Оцінимо тепер дисперсію сукупності за вибірковими середніми m_y. Оскільки ми припустили, що всі p вибірки витягнуті з однієї сукупності, стандартне відхилення p вибіркових середніх служить оцінкою помилки середнього. Стандартна помилка середнього σ_{m_y} зв'язана зі стандартним відхиленням сукупності σ_y і з обсягом вибірки n наступним співвідношенням:

$$\sigma_{m_y} = \frac{\sigma_y}{\sqrt{n}}. \tag{1.38}$$

Таким чином, дисперсію сукупності σ_y^2 можна розрахувати в такий спосіб:

$$D_A = \sigma_y^2 = n\sigma_{m_y} \tag{1.39}$$

Ця оцінка називається міжгруповою дисперсією, що позначається як D_A. Якщо вірна нульова гіпотеза, то тоді як внутрішньогрупова, так і міжгрупова дисперсії служать оцінками тієї ж самої дисперсії і повинні бути приблизно рівні. Виходячи з цього, обчислюється критерій значущості Фішера F:

$$F = D_A/D_R. \tag{1.40}$$

І чисельник, і знаменник цього відношення – це оцінки тієї ж самої величини – дисперсії сукупності D_y, тому значення F повинне були близьке до 1. Отже, якщо F значно перевищує 1, нульову гіпотезу H_0 варто відкинути, у противному разі H_0 варто прийняти. Залишилося зрозуміти, починаючи з якої саме величини F варто відкидати H_0.

Якщо витягати випадкові вибірки обсягом n з нормально розподіленої сукупності N ($N >> n$), то значення F буде мінятися з кожним експериментом,

тому F є випадковою величиною, яка має розподіл Фішера $f(F)$. При цьому вид кривої розподілу залежить від параметрів n та p.

Значення критерію, починаючи з якого ми відкидаємо нульову гіпотезу, називається критичним значенням F_k. Імовірність помилково відкинути вірну нульову гіпотезу, тобто знайти розходження там, де їх немає, позначається як P та визначається як площа під кривою $f(F)$ на інтервалі $F>F_k$. Імовірність помилки P називається рівнем значущості α (зазвичай α = 0,05 або 0,01).

Таким чином, критичне значення F_k однозначно визначається рівнем значущості α і ще двома параметрами, що називаються внутрішньогруповим $\nu_{вну}$ і міжгруповим числом ступенів свободи $\nu_{між}$, значення яких визначається на основі формул (1.37, 1.39):

$$\nu_{між} = p - 1, \ \nu_{вну} = p\,(n - 1).$$

Обчислити критичне значення F_k досить складно, тому користуються таблицями критичних значень F_k для різних α, $\nu_{між}$ і $\nu_{вну}$. Після обчислення F та F_k приймається таке рішення:

H_0: якщо $F < F_k$;
H_1: якщо $F > F_k$.

При плануванні експериментів для проведення дисперсійного аналізу кількість експериментів n на кожному рівні p вибирається однаковою, хоча в роботі [1] наведено вирази, аналогічні формулам (1.38, 1.40) для різних обсягів груп, які застосовуються при пасивному експерименті.

При виконанні розрахунку за допомогою програмних пакетів (наприклад, StatGraphics) результати розрахунків видаються у вигляді табл. 1.3. Наведені у цій таблиці середні значення на кожному i-му рівні фактора y_{i0} та по всій вибірці y_{00} розраховуються за формулами:

$$y_{i0} = \frac{1}{n}\sum_{j=1}^{n} y_{ij}; \qquad y_{00} = \frac{1}{q \cdot n}\sum_{j=1}^{q}\sum_{i=1}^{n} y_{ij}\ .$$

Якщо взяти рівень значущості $\alpha = 0{,}05$, то при $\alpha > 0{,}05$ приймається гіпотеза H_0, а при $\alpha < 0{,}05$ – H_1.

Якщо справедлива гіпотеза H_1, то для визначення відхилень у рівнях фактора x_i звичайно розраховуються граничні (95 % довірчий інтервал) відхилення вихідної величини y при заданому рівні фактора p.

Таблиця 1.3 – Результати однофакторного дисперсійного аналізу

Компоненти дисперсії	Сума квадратів	Число ступенів свободи	Середньо-квадратичне відхилення	Відхилення Фішера	Рівень значу-щості
Між рівнями фактора	$S_A = \sum_{i=1}^{p} n(y_{i0} - y_{00})^2$	$p-1$	$d_A = \frac{S_A}{p-1}$	$\frac{d_A}{d_R}$	α
Усередині факторів	$S_R = \sum_i \sum_j (y_{ij} - y_{i0})^2$	$p(n-1)$	$d_R = \frac{S_R}{p(n-1)}$		
Повна дисперсія	$S_G = \sum_{i=1}^{p} \sum_{j=1}^{n} (y_{ij} - y_{00})^2$	$pn-1$			

Двофакторний дисперсійний аналіз. Задача двосторонньої класифікації виникає при проведенні спостережень в експерименті, у якому одночасно діють два фактори (x_i та x_j), які мають p та q рівнів варіацій відповідно. Тому результати вимірів заносяться в двовимірну таблицю Y розміром $p*q$, причому кожна її комірка містить результати n вимірів вихідної величини y_{ijk}, $i = \overline{1,p}$, $j = \overline{1,q}$, $k = \overline{1,n}$.

Середні значення по рядках, по стовпцях і повні середні розраховуються за формулами

$$y_{ij0} = \frac{1}{n}\sum_{k=1}^{n} y_{ijk}; \qquad y_{i00} = \frac{1}{q \cdot n}\sum_{j=1}^{q}\sum_{k=1}^{n} y_{ijk};$$

$$y_{0j0} = \frac{1}{p \cdot n}\sum_{i=1}^{p}\sum_{k=1}^{n} y_{ijk}; \qquad y_{000} = \frac{1}{p \cdot q \cdot n}\sum_{i=1}^{p}\sum_{j=1}^{q}\sum_{k=1}^{n} y_{ijk}.$$

Дисперсії розраховуються за формулами

$$S_G = \sum_{i=1}^{p}\sum_{j=1}^{q}\sum_{k=1}^{r}(y_{ijk} - y_{000})^2; \qquad S_A = q \cdot r\sum_{i=1}^{p}(y_{i00} - y_{000})^2;$$

$$S_B = p \cdot r\sum_{r}^{q}(y_{0j0} - y_{000})^2; \qquad S_{AB} = r\sum_{i=1}^{p}\sum_{j=1}^{a}(y_{ij0} - y_{000})^2;$$

$$S_R = r\sum_{i=1}^{p}\sum_{j=1}^{a}\sum_{k=1}^{r}(y_{ijk} - y_{ij0})^2;$$

Результати розрахунків двофакторного дисперсійного аналізу видаються у вигляді табл. 1.4.

Таблиця 1.4 – Результати двофакторного дисперсійного аналізу

Компоненти дисперсії	Сума квадратів	Число ступенів свободи	Середньо-квадратичне відхилення	Відношення Фішера	Рівень значущості
Між факторами А	S_A	p-1	$\frac{S_A}{p-1}$	$\frac{S_A/(p-1)}{S_R/pq(n-1)}$	α_A
Між факторами В	S_B	q-1	$\frac{S_B}{q-1}$	$\frac{S_B/(q-1)}{S_R/pq(n-1)}$	α_B
Між взаємодією АВ	S_{AB}	$(p$-1$)(q$-1$)$	$\frac{S_{AB}}{(p-1)(q-1)}$	$\frac{S_{AB}/(p-1)(q-1)}{S_R/pq(n-1)}$	α_{AB}
Помилка (усередині фактора)	S_R	$pq(n$-1$)$	$\frac{S_R}{pq(n-1)}$		
Повна	S_G				

Таким чином: $S_G = S_A + S_B + S_{AB} + S_R$.

де S_G визначає повну дисперсію; S_A – дисперсію за фактором A; S_B – дисперсію за фактором B; S_{AB} – дисперсію за взаємодією A та B; S_R – дисперсію вимірів (помилку).

Перевіряється гіпотеза H_0.

Якщо $\left.\begin{array}{l}\alpha_A > 0{,}05\\ \alpha_B > 0{,}05\\ \alpha_{AB} > 0{,}05\end{array}\right\}$, то має місце гіпотеза H_0.

Якщо справедлива гіпотеза H_0, то вплив факторів A, B та їх взаємодії AB статистично не значущі. Якщо справедлива H_1, то можна визначити, які з факторів впливають на результати виміру вихідної величини y.

Регресійний аналіз. [1, 7]. З позиції регресійного аналізу розглядається зв'язок між вектором вхідних змінних X = (x_i, ..., x_p) і вихідною (залежною) змінною y:

$$y = f(X) + \varepsilon. \qquad (1.41)$$

Для оцінки ефективності регресійної діагностичної моделі вводиться вектор залишків $\varepsilon=(\varepsilon_1, ..., \varepsilon_n)'$, що відображує вплив на y сукупності неврахованих випадкових факторів або міру досяжної апроксимації значень y

функціями типу $y = f(x)$. Зазвичай $f(x)$ шукається в вигляді полінома з цілочисельними значеннями степенів x. За кількістю вхідних змінних (факторів) розрізняють одно- та багатофакторні (множинні) регресії, за степенем полінома – лінійні та нелінійні. У найбільш загальному випадку (нелінійна множинна регресія) рівняння регресії задається у вигляді полінома Колмогорова – Габора

$$\hat{y} = P(x_1,\ldots,x_n) = a_0 + \sum_i a_i x + \sum_i \sum_j a_{ij} x_i x_j + \sum_i \sum_j \sum_k a_{ijk} x_i x_j x_k \ldots \quad (1.42)$$

Задачею регресійного аналізу є визначення параметрів моделі – коефіцієнтів полінома (1.42) на основі аналізу синхронно виміряних значень вектора вхідних даних $X = (x_i, ..., x_p)$ та вихідних значень y. Розрізняють два підходи залежно від походження матриці даних. У першому вважається, що вектор ознак X є детермінованим, а випадковою величиною є тільки залежна змінна y. Ця модель використовується найбільш часто й називається моделлю з фіксованою матрицею даних. У другому підході вважається, що вектор ознак X і y – випадкові величини, що мають спільний розподіл. У такій ситуації оцінка рівняння регресії є оцінкою умовного математичного очікування випадкової величини y залежно від випадкових величин $x_i, ..., x_p$. Дана модель називається моделлю з випадковою матрицею даних. Кожний з наведених підходів має свої особливості. У той же час показано, що моделі з фіксованою матрицею даних і з випадковою матрицею даних відрізняються тільки статистичними властивостями оцінок параметрів рівняння регресії, тоді як обчислювальні аспекти цих моделей збігаються. Слід зазначити, що число вимірювань (точок) N повинне бути не менше від числа коефіцієнтів n у виразі (1.42), а з врахуванням того, що вхідні дані є випадковими величинами, число вимірювань N вибирається за умови $N>>n$.

У рівнянні функції регресії (1.41) зазвичай вважається, що величини ε_i (i=1, N) незалежні й випадково розподілені з нульовим середнім і дисперсією σ^2_ε, а оцінка параметрів a_i виконується за допомогою методу найменших квадратів (МНК).

Всі зазначені раніше види регресійних залежностей можна звести до лінійної багатофакторної регресії шляхом заміни всіх складових у виразі (1.42) новими вхідними змінними $z_i,\ i = \overline{1,n}$:

$$z_1 = x_1,\ \ldots,\ z_p = x_p,\ z_{p+1} = x_1 x_2,\ \ldots,\ z_n = x_p x_p \ldots x_p.$$

Якщо ввести фіктивну змінну $z_0 \equiv 1$ та підставити вираз (1.42) в (1.41), то одержимо

$$y_i = \sum_{j=1}^{n} a_j z_{ji} + \varepsilon_i \,. \quad (1.43)$$

Оскільки величина y обчислюється завжди з деякою помилкою ε, то можна обчислити суму квадратів відхилень між виміряним та розрахунковим значенням y_i:

$$S = \sum_{i=1}^{N} (y_j - \sum_{j=1}^{n} a_j z_{ji})^2 . \qquad (1.44)$$

При використанні МНК сума квадратів відхилення повинна прямувати до мінімальної. Так як функція S залежить від коефіцієнтів регресії a_j, їх потрібно вибрати таким чином, щоб $S \to \min$, тобто

$$\frac{\partial S}{\partial a_j} = 2\sum_{i=1}^{N} (y_j - \sum_{j=1}^{n} a_j z_{ji}) z_j = 0;\ j = \overline{1,n} . \qquad (1.45)$$

Це приводить до нормальної системи лінійних рівнянь відносно невідомих коефіцієнтів регресії:

$$\boldsymbol{S^*A = C_{yz}}, \qquad (1.46)$$

де $\boldsymbol{S}$ – матриця коваріації ознак z_0, ..., z_n, елементи якої обчислюються за формулою $s_{ij} = \sum_{k=1}^{N} z_{ik} z_{jk}$; $\boldsymbol{C_{yz}}$ – вектор оцінок коваріації між вихідною величиною y та ознаками z_0, ..., z_n, елементи якого обчислюються за формулою $c_i = \sum_{k=1}^{N} y_k z_{jk}$; $\boldsymbol{A}$ – вектор коефіцієнтів, які потрібно обчислити.

Розв'язання системи рівнянь (1.46) виконується будь-яким відомим методом, доступним для дослідника (метод Гаусса, квадратного кореня та ін.). Можна визначити довірчі інтервали та гіпотези щодо значущості коефіцієнтів a_i за виразом (1.31), тому що значення t_a, як і t_r в кореляційному аналізі розподілені за законом Стьюдента з $(N-2)$ ступенями свободи.

Перевірка гіпотези про адекватність регресійної моделі виконується з використанням критерію Фішера. Якщо перевірити гіпотезу H_0 про те, що розглянута модель адекватна об'єктові, то для перевірки цієї гіпотези необхідно зіставити досягнуту точність моделі з величиною, що характеризує точність спостережень. Якщо точність моделі перевершує точність спостережень, то гіпотеза H_0 відхиляється.

Порівняння дисперсій виконується за формулою

$$W_0 = \frac{\dfrac{S_d}{N-n-1}}{\dfrac{S_y}{N-1}}, \qquad (1.47)$$

де S_d – сума квадратів відхилення експериментальних точок і точок моделі, яка розраховується за виразом (1.44); S_y – сума квадратів відхилення вихідної величини y від свого середнього m_y, яка розраховується за формулою

$$S_y = \sum_{i=1}^{N} (y_i - m_y)^2 \ .$$

Якщо $W_0 < W_{0,95}(N - n - 1, N - 1)$, то приймається гіпотеза H_0. Інакше вона відкидається. Крім того, використовуються показники якості регресійної діагностичної моделі, а саме:

- залишкова сума квадратів, яка розраховується за виразом (1.44);
- незміщена оцінка дисперсії помилки $s_\varepsilon^2 = S_d / (N - n)$;
- оцінка дисперсії прогнозованої змінної $\sigma_y^2 = \frac{1}{N} S_y$;
- коефіцієнт детермінації $R^2 = \frac{N\sigma_y^2 - S_d}{N\sigma_y^2}$, який змінюється від 0 (відсутній зв'язок X з y) до 1 (повністю детермінований зв'язок);
- оцінка дисперсії коефіцієнтів регресії $D_{ai} \approx \frac{s_\varepsilon^2}{N} s_{ii}$, де s_{ii} – відповідний елемент коваріаційної матриці ознак.

Складна регресійна залежність. Якщо регресійна залежність має складний характер, то підібрати відповідний поліном практично не вдається. Тому використовується кусково-параметрична апроксимація, при цьому вісь X розбивається на кілька ділянок, у яких вибирається лінійна або квадратична апроксимація, яка щонайкраще описує цю ділянку. Часто для цих цілей використовується сплайнова апроксимація.

Методи самоорганізації регресійних моделей. Як було зазначено раніше, визначення параметрів моделі (коефіцієнтів полінома) за МНК зводиться до розв'язання нормальної системи лінійних алгебраїчних рівнянь щодо невідомих коефіцієнтів полінома. При значному числі вихідних ознак і при збільшенні ступеня полінома число коефіцієнтів у виразі (1.42) зростає лавиноподібно, що висуває підвищені вимоги до обсягу навчальної вибірки (для реалізації МНК число точок має бути істотно більше від сумарного числа коефіцієнтів). Крім того, у реальних даних, як правило, зустрічаються групи сильнозв'язаних ознак. У цих умовах виникає явище мультиколінеарності, що приводить до поганої обумовленості і, у граничному випадку, до виродження коваріаційної матриці. При цьому розв'язання нормальної системи лінійних рівнянь є нестійким, або розв'язок одержати не можна. Тому на практиці зазвичай обмежуються лінійними

регресійними моделями, хоча вони неточні і використовуються для "грубої" оцінки з метою добору множини впливаючих факторів моделі *X*.

Для синтезу регресійної моделі за невеликим числом експериментальних даних використовуються методи самоорганізації [8], які одночасно визначають структуру полінома і його коефіцієнти. У методах самоорганізації використовується ітераційна процедура послідовного ускладнення полінома з вибором найкращих рішень на кожному кроці ітерації. Особливістю методів самоорганізації є застосування принципів неостаточності рішення і зовнішнього критерію.

Принцип неостаточності рішення запозичений від еволюційних і генетичних алгоритмів і полягає в тому, що на кожному кроці залишається не один кінцевий результат, а група найкращих рішень, тобто відтинаються безперспективні рішення, і тільки на останньому кроці з усіх найкращих рішень вибирається єдине найкраще (оптимальне) серед усіх рівних.

Принцип зовнішнього критерію полягає в тому, що оцінка якості прогнозуючої моделі (точності наближення експериментальних даних до рівнянь регресії) виконується за допомогою критерію, що є зовнішнім стосовно критерію, за допомогою якого визначаються коефіцієнти. Одним з варіантів зовнішнього критерію може бути розбиття всіх експериментальних точок на дві частини, де перша з них (навчальна вибірка) служить для визначення коефіцієнтів по МНК, а друга (перевірочна (зовнішня) вибірка) – для оцінки точності рівняння регресії. Застосування зовнішнього критерію дозволяє одержувати модель оптимальної складності.

Суть алгоритмів самоорганізації полягає в тому, що повний опис об'єкта виду (1.42) заміняється множиною часткових описів. Як часткові описи беруться поліноми ступеня, що не вищий від другого, причому від числа змінних не більше двох (лінійні, з коваріаційними частковими описами, із квадратичними частковими описами). Ускладнення моделі (збільшення числа змінних і ступеня полінома) виконується при переході до наступного кроку ітерації, причому результати найкращих моделей попереднього кроку (відносно зовнішнього критерію) є вихідними даними наступного кроку ітерації. Безліч алгоритмів самоорганізації відрізняються структурою часткових описів, зовнішнім критерієм (точність моделі на точках перевірочної послідовності, баланс коефіцієнтів), способом одержання результуючої моделі й ін.

1.2.4. Методи оцінки інформативності та формування інформативного простору ознак

Теоретико-інформаційний підхід. Оцінка інформативності різнорідних ознак. Теоретико-інформаційний підхід [1, 9] є найбільш строгим і

формалізованим методом оцінки інформативності окремих ознак (або комплексу ознак) щодо заданої системи станів $\{D\}_n$. У даному методі для оцінки інформативності використовуються такі фундаментальні поняття теорії інформації, як ентропія та кількість внесеної інформації [9].

Якщо система станів утворює повну групу несумісних подій (кожен об'єкт в навчальній вибірці належить тільки одному стану D_i, немає об'єктів з декількома станами), то сумарна ймовірність всіх станів дорівнює 1:

$$\sum_{i=1}^{n} P(D_i) = 1 .$$

Тоді невизначеність системи можливих станів оцінюється за допомогою ентропії

$$H(D) = -\sum_{i=1}^{n} P(D_i) \cdot \log_2 P(D_i) . \qquad (1.48)$$

При цьому для n можливих рівноймовірних складових її значення буде максимальним:

$$H(D) = \log_2 n,$$

а у випадку диференціальної діагностики ($n = 2$) $H(D)$ не перевищить 1 біта.

Оскільки ентропія відображає міру невизначеності системи, її величина буде змінюватися при надходженні в систему нової інформації. Такою інформацією для станів є дані, отримані в результаті вимірювання ознак об'єкта, що діагностується. Зменшення ентропії відбувається на величину, що дорівнює кількості внесеної інформації. Крайнє значення, якого може набувати ентропія, дорівнює нулеві і має місце для достовірної події. У цьому випадку нуль показує відсутність невизначеності в системі.

Відповідно кількість інформації, що надійшла в систему, визначається як різниця між величиною ентропії до і після вимірювання:

$$I_D(x_j) = H(D) - H(D / x_j) , \qquad (1.49)$$

де $I_D(x_j)$ – кількість інформації, внесеної в систему після проведення вимірювання ознаки x_j; $H(D)$ – початкова ентропія системи діагнозів; $H(D/x_j)$ – ентропія системи після проведення вимірювання ознаки x_j.

Таким чином, величина $I_D(x_j)$ характеризує діагностичну цінність ознаки x_j стосовно системи діагнозів D і ґрунтується на кількості інформації, яка надійшла.

Діагностична цінність простої дихотомічної ознаки, яка набуває одного з двох можливих значень, визначається за формулою:

$$I_{D_i}(x_j) = \log_2 \frac{P(x_j / D_i)}{P(x_j)}, \qquad (1.50)$$

де $I_{Di}(x_j)$ – діагностична вага ознаки x_j для стану D_i; $P(x_j/D_i)$ – апріорна імовірність наявності ознаки при стані D_i; $P(x_j)$ – апріорна імовірність наявності ознаки у всій системі можливих станів D.

Величина $P(x_j/D_i)$ розраховується як відношення кількості об'єктів, у яких є присутньою ознака x_j при стані D_i, до загального числа об'єктів з розглянутим станом. На підставі формули (1.50) можна зробити висновок, що при однаковому значенні ймовірностей наявності ознаки для конкретного стану і для всієї системи станів діагностична вага ознаки дорівнює нулю й ознака не несе ніякої інформативності.

Діагностична вага відсутності простої ознаки визначається за допомогою виразу, що виходить з формули (1.50) шляхом внесення обернених величин ймовірностей:

$$I_{D_i}(\overline{x_j}) = \log_2 \frac{1 - P(x_j / D_i)}{1 - P(x_j)}, \qquad (1.51)$$

Варто враховувати, що діагностична вага ознаки може бути як позитивною, так і негативною величиною, тобто вона може як зменшувати, так і збільшувати імовірність того або іншого стану.

Повна діагностична вага простої ознаки для стану D_i враховує як наявність, так і відсутність ознаки і може бути розрахована за формулою:

$$I_{D_i}(x_j) = P(x_j / D_i) \cdot \log_2 \frac{P(x_j / D_i)}{P(x_j)} + [1 - P(x_j / D_i)] \cdot \log_2 \frac{1 - P(x_j / D_i)}{1 - P(x_j)}. \qquad (1.52)$$

Діагностична цінність простої ознаки для системи станів визначається як

$$I_D(x_j) = \sum_{i=1}^{n} P(D_i) \cdot I_{D_i}(x_j). \qquad (1.53)$$

Оцінка інформативності складних ознак. До складних ознак належать рангові, значення яких можна виразити скінченним числом інтервалів, і числові, котрі теж можна виразити скінченним числом інтервалів, тому що будь-які виміри виконуються з скінченною точністю. Якщо розглядати кожен інтервал (діагностичний розряд) складної ознаки як просту ознаку, то діагностична цінність s-го інтервалу складної ознаки за формулою (1.50) запишеться у вигляді:

$$I_{D_i}(x_{js}) = \log_2 \frac{P(x_{js} / D_i)}{P(x_{js})}, \qquad (1.54)$$

де $P(x_{js} / D_i)$ – апріорна ймовірність s-го діагностичного інтервалу складної ознаки для стану D_i.

Величина

$$I_{D_i}(x_j) = \sum_{s=1}^{m} P(x_{js} / D_i) \cdot \log_2 \frac{P(x_{js} / D_i)}{P(x_{js})} \qquad (1.55)$$

визначає діагностичну цінність складної ознаки для діагнозу D_i.

Для визначення повної діагностичної цінності складної ознаки відносно системи станів застосовується формула

$$I_D(x_j) = \sum_{i=1}^{n} \sum_{s=1}^{m} P(D_i) \cdot P(x_{js} / D_i) \cdot \log_2 \frac{P(x_{js} / D_i)}{P(x_{js})}. \qquad (1.56)$$

При визначення інформативності за формулами (1.54–1.56), особливо при великому m, крім збільшення складності обчислень, ставляться підвищені вимоги до обсягу і репрезентативності навчальної вибірки, що не завжди може бути досягнуто в реальних базах даних.

Для оцінки інформативності числових ознак за виразами (1.55, 1.56) необхідно виконати розбиття динамічного діапазону ознаки на діагностично-значущі інтервали. Дана задача вирішується на інтуїтивному рівні, причому для зручності обчислень, розбиття виконується на m рівномірних інтервалів [9]. У роботі [5] пропонується метод розбиття динамічного діапазону числової ознаки на задане число m нерівномірних діагностично-значущих інтервалів на основі аналізу інтегральної помилки аппроксимації теоретичного закону розподілу гістограмою та обмеженістю навчальної вибірки.

Алгоритми визначення групи інформативних ознак. Для вирішення задачі визначення групи інформативних ознак використовуються такі підходи й алгоритми [1]:

- перебір усіх можливих комбінацій ознак;
- метод "k" кращих ознак;
- методи послідовного зменшення (DEL або ПЗМП) і збільшення (ADD або ПЗБП) простору ознак;
- узагальнений алгоритм (ADD - DEL або "плюс l мінус r");
- методи, засновані на критерії максиміну;
- еволюційні алгоритми, зокрема алгоритми випадкового пошуку з адаптацією;
- метод гілок і границь та інші.

Повний перебір усіх можливих комбінацій ознак. Задача пошуку групи інформативних ознак при побудові алгоритмів розпізнавання формулюється в такий спосіб. Нехай задані вихідна множина ознак $X = \{x_i\}, i = \overline{1, p}$ і

деякий критерій якості розпізнавання *J*.

Позначимо через $X_n \in X$ групу з *n* ознак, що має найкраще значення критерію якості рішення задачі розпізнавання $J(X_n)$ у порівнянні з будь-якою іншою групою $X_n^l \in X$. Тобто $J(X_n) = \max_l J(X_n^l)$.

Самий надійний метод пошуку X_n полягає в повному переборі всіх можливих груп ознак. Але кількість таких груп складає C_p^n, і для високих размірностей повний перебір є нереальним. Тому всі розглянуті нижче алгоритми визначення X_n пов'язані зі спробами уникнути повного перебору.

Метод "*k*" кращих ознак. У даному алгоритмі використовується припущення про статистичну незалежність ознак. Вихідні ознаки ранжируються за обраним критерієм якості:

$$J(x_{i_1}) \geq J(x_{i_2}) \geq \ldots \geq J(x_{i_j}) \geq \ldots \geq J(x_{i_p}), \qquad (1.57)$$

і з побудованого ряду відбирається "*k*" перших, найбільш цінних ознак.

Чим суворіше дотримується умова незалежності ознак, що відбираються, тим кращий виходить кінцевий результат. Але умова незалежності рідко виконується в реальних БД, тому для добору і наближеного визначення ваг коррельованих ознак використовують більш складні методи.

Методи послідовного збільшення і зменшення простору ознак (ПЗБП і ПЗМП). Зазначені евристичні алгоритми враховують коррельованість ознак, виконують спрямований перебір і одержують рішення, близьке до оптимального.

У ПЗБП спочатку визначається одна ознака, що має максимальне значення критерію *J*. Потім на кожному кроці пошуку нова група утворюється шляхом додавання однієї ознаки, яка, будучи включеною у розширену групу, дає максимальне значення критерію *J*. Процедура повторюється, поки не буде побудована група з *n* ознак.

Незважаючи на більш витончені операції з експериментальною інформацією у порівнянні з методом "*k*" кращих ознак, метод ПЗБП не гарантує одержання оптимального результату, який може бути досягнутий за допомогою повного перебору всіх можливих комбінацій вихідних ознак.

ПЗМП заснований на послідовному зменшенні групи на одну ознаку. Спочатку з поточної групи X_k по черзі вилучаються одиночні ознаки. Групи X_{k-1}, що утворилися, перевіряються за критерієм *J*. Вилученню підлягає ознака, при відсутності якої зменшена група X_{k-1} має максимальне значення критерію *J*. Процедура повторюється доти, поки не буде побудована група з *n* ознак. За допомогою зазначеного алгоритму можуть бути отримані більш ефективні результати, ніж для ПЗБП, у випадку порівняно невеликого обсягу

групи вихідних ознак. Для високих розмірностей простору вихідних ознак виникають серйозні проблеми оцінки показника якості, тому що вплив окремо узятої ознаки на сумарний ефект стає порівняним з похибкою його виміру.

Алгоритм "плюс *l* мінус *r*". Узагальненням ПЗБП і ПЗМП служить метод "плюс *l* мінус *r*", що по черзі працює то на додавання ознак, то на вилучення цих ознак. У цьому методі на *k*-му ступені пошуку група спочатку розширюється на *l* ознак, з використанням методу ПЗБП, а потім із одержаної групи вилучається *r* ознак за допомогою методу ПЗМП. Якщо $l < r$, то маємо метод зменшення групи, а якщо $l > r$, – то метод збільшення групи. Хоча алгоритм "плюс *l* мінус *r*" дозволяє вилучати або додавати ознаки в поточну групу, врахування їхнього взаємного відношення не виконується. Цей недолік деякою мірою може бути компенсований шляхом додавання і виключення з групи одночасно декількох ознак ("узагальнений алгоритм плюс *l* мінус *r*").

Методи, засновані на стратегії максиміну. Особливістю цих методів є використання дуже обмеженого обсягу інформації, наприклад, тільки про індивідуальну $J(x_i)$ і парну ефективність ознак $J(x_i, x_j)$. У даному випадку нова ознака x_j включається в групу, якщо її приєднання до однієї з уже наявних ознак забезпечує максимальне додаткове збільшення критеріїв якості, тобто

$$\Delta J(x_i, x_j) = J(x_i, x_j) - J(x_i) = \max . \qquad (1.58)$$

У той же час включення x_j у групу виправдане, якщо x_j не коррельована з іншими ознаками.

Алгоритм випадкового пошуку з адаптацією. Суть алгоритму полягає в такому. З множини вихідних ознак X випадковим чином вибираються n ознак і генерується серія l груп ознак $X_n^{(l)}$. Потім визначаються величини критерію J для всіх отриманих $X_n^{(l)}$. Група з максимальним значенням критерію заохочується збільшенням імовірності вибору її ознак у наступних шагах алгоритму, а група з найменшою величиною критерію карається відповідним чином. Ця процедура повторюється доти, поки ймовірність вибору інших груп не наблизиться до нуля. Група з максимальною ймовірністю вибору приймається за найбільш цінну групу з n ознак.

Метод гілок і границь. В основі даного методу лежить припущення про монотонність функції критерію, що виражається такою умовою:

$$J(X_{\text{р}}) > J(X_{\text{р-1}}) > \ldots > J(X_{\text{n}}). \qquad (1.59)$$

Метод полягає в процедурі зменшення групи, але з можливістю повернення, і дозволяє вилучити з розгляду деякі групи ознак без розрахунків оцінки їхньої ефективності. На кожному *k*-му кроці аналізу дерева рішень утворюється група ознак *k*-го рівня, що містить ознаки, обрані з групи (*k*+1)-

го рівня. Потім одна з гілок дерева рішень обстежується до останнього n-го рівня. Максимальна величина критерію груп X_n останнього рівня береться за поріг. Якщо при пошуку на k-му рівні недослідженої частини дерева величина критерію $J(X_k)$ виявилася меншою від порога, то подальший пошук серед груп, що утворяться з X_k, не виконується, тому що внаслідок монотонності критерію всі ці групи будуть мати величину критерію, що нижча від порога. Як правило, для отримання оптимальної групи ознак велика кількість гілок дерева обстежується не до останнього рівня, що забезпечує значне скорочення обчислювальних витрат.

У цілому можна відзначити, що багато які зі згаданих методів визначення складу ознак містять евристичну складову. У кожному конкретному випадку важко заздалегідь угадати, який з цих методів приведе до результату, більш близькому до оптимального. Тому на практиці спроби наблизитися до бажаного оптимуму завжди пов'язані з комбінованим застосуванням різних алгоритмів пошуку групи інформативних ознак.

Контрольні питання

1. Математичні методи перетворення простору ознак. Їх призначення й область застосування.
2. Типи вихідних даних і методи їхньої обробки.
3. Як виконується центрування та нормування даних? Поняття масштабу.
4. Які вимоги повинна задовольняти система вихідних діагностичних ознак.
5. Як і для чого формується таблиця експериментальних даних (ТЕД).
6. Міри зв'язку між ознаками.
7. Коефіцієнт кореляції Пірсона, його призначення та метод обчислення.
8. Як визначається значущість коефіцієнта кореляції?
9. Коефіцієнти рангової кореляції Спірмена та Кендалла, Їх призначення та методи обчислення.
10. Коефіцієнт спряженості, його призначення та метод обчислення.
11. Міри близькості (віддаленості) між об'єктами. Вимоги до них.
12. Як обчислюються і де застосовуються евклідова та зважена евклідова відстані, відстані Махаланобіса, Мінковского, Хеммінга?
13. Які вам відомі методи зниження розмірності простору ознак?
14. Які задачі вирішуються за допомогою кластерного аналізу та області його застосування?
15. Які вам відомі методи кластерного аналізу та особливості їх реалізації?
16. Як реалізується метод кореляційних плеяд?
17. Сутність та реалізація методу головних компонент.

18. Факторний аналіз. Його сутність та реалізація.
19. Сутність та реалізація методу контрастних груп.
20. Що таке і як реалізується багатовимірне шкалювання?
21. Перевірка статистичних гіпотез. Основи дисперсійного аналізу.
22. Реалізація однофакторного дисперсійного аналізу.
23. Реалізація двофакторного дисперсійного аналізу.
24. Основи регресійного аналізу. Класифікація регресійних моделей.
25. Синтез регресійних моделей. Метод найменших квадратів.
26. Показники якості регресійних моделей.
27. Методи самоорганізації регресійних моделей.
28. Міра інформативності дихотомічних ознак.
29. Міра інформативності складних ознак.
30. Методи зменшення розміру простору ознак. Повний перебір, *к*-найкращих, ПЗМГ (DEL), ПЗБГ (ADD), (*к*-плюс *l*-мінус).
31. Методи зменшення розміру простору ознак, засновані на стратегії максиміну. Алгоритм випадкового пошуку з адаптацією. Метод гілок і границь.

1.3. Методи синтезу та області застосування вирішальних правил

При формуванні вирішальних правил класифікації моделлю об’єкта діагностики (ОД) є "чорна шухляда", і шукається залежність між формалізованими станами Y і вектором вхідних ознак X, тобто $Y = f(X)$. Існує безліч методів синтезу вирішальних правил класифікації об'єктів, серед яких виділяють такі класи [1, 5]:

- детерміновані методи;
- імовірнісні методи;
- метод послідовного аналізу (метод Вальда);
- методи, засновані на теорії розпізнавання образів;
- логіко-лінгвістичні методи;
- методи, засновані на нечіткій логіці;
- методи на основі штучних нейронних мереж та ін.

1.3.1. Типи вирішальних правил

Детерміновані методи. Застосовуються у випадках наявності детермінованих зв'язків між ознаками і формалізованими станами об’єктів, як правило на етапі попередньої класифікації. Наприклад, в медичній діагностиці існує група діагностичних ознак (патогномонічні синдроми), які

однозначно визначальні при деяких захворюваннях, і, навпаки, існує група синдромів, що ніколи не зустрічаються при деяких захворюваннях. Тому використання детермінованих зв'язків виконується для вирішення двох задач:

- однозначна постановка діагнозу за наявності в пацієнта патогномонічного синдрому;
- виключення з подальшого розгляду визначеної групи захворювань за відсутності в пацієнта патогномонічних синдромів зазначеної групи.

Алгоритмічно детерміновані методи реалізуються у вигляді обчислення хеммінгової відстані (на код детермінованих ознак об'єкта діагностики послідовно накладаються коди синдромів діагностуємих станів) або у виді розгалуженого дерева можливих рішень з перевіркою визначених умов у вузлах розгалуження – система детермінованих правил.

Імовірнісні (статистичні) методи. Ці методи засновані на використанні апарата математичної статистики [1, 10]. Вони найчастіше застосовуються у випадках, коли відомі ймовірнісні характеристики класів або коли вони можуть бути визначені за наявною навчальною вибіркою, що звужує область їхнього застосування. Найчастіше імовірнісні методи розрізняються за критерієм розпізнавання. Нижче перераховані основні види критеріїв.

Критерій Байєса. Байєсівський підхід полягає в обчисленні умовних апостеріорних імовірностей. При цьому рішення приймається на підставі порівняння значень цих імовірностей. Якщо об'єкт ω характеризується N ознаками x_i, що набувають значення $x_1 = x_1^0$, $x_2 = x_2^0$, …, $x_N = x_N^0$, то апостеріорна ймовірність віднесення об'єкта до класу Ω_m, $m = \overline{1, M}$ при здійсненні події $a_N = (x_1^0, x_2^0, \ldots, x_N^0)$ обчислюється таким чином:

$$P(\Omega_m / a_N) = \frac{P(\Omega_m) f_m(x_1^0, x_2^0, \ldots, x_N^0)}{\sum_{m=1}^{M} P(\Omega_m) f_m(x_1^0, x_2^0, \ldots, x_N^0)}, \qquad (1.60)$$

де $P(\Omega_m)$ – апріорна ймовірність появи об'єктів класу Ω_m;

$f_m(x_1^0, x_2^0, ..., x_N^0)$ – умовна щільність розподілу ймовірностей значень ознак x_i об'єктів класу Ω_m.

Більш детально байєсівський підхід розглядається в 1.3.3.

Мінімаксний критерій. Мінімаксний критерій мінімізує максимально можливе значення середнього ризику. Алгоритм класифікації формулюється в такий спосіб: якщо виміряне значення ознаки x в об'єкті ω дорівнює x^0, то $\omega \in \Omega_1$, якщо $x = x^0 < x_0$, і $\omega \in \Omega_2$, якщо $x = x^0 > x_0$. Граничне значення x_0

визначається зі співвідношення

$$c_1 Q_1(x_0) = c_2 Q_2(x_0), \tag{1.61}$$

де c_1, c_2 – втрати, зв'язані з помилками 1-го і 2-го роду відповідно;

Q_1, Q_2 – умовні імовірності помилок 1-го і 2-го роду відповідно.

Таким чином, «мінімаксна стратегія є байєсівська стратегія для найгірших значень апріорних імовірностей, що дає хоча й обережне, але гарантоване значення середнього ризику».

Критерій Неймана-Пірсона. Суть критерію Неймана-Пірсона полягає в тім, щоб домогтися мінімуму умовної ймовірності помилки 2-го роду Q_2 при заданому значенні умовної імовірності помилки 1-го роду Q_1. Тобто у випадку класифікації об'єктів на два класи (Ω_1 і Ω_2) при $Q_1 \le A$, де A – деяка постійна величина, потрібно визначити рішення x_0, що задовольняє рівнянню

$$\int_{x_0}^{\infty} f_1(x)dx = A, \tag{1.62}$$

де $f_1(x)$ – умовна щільність розподілу ймовірностей значень ознаки x об'єктів класу Ω_1.

Тоді якщо в об'єкта ω виміряне значення ознаки $x = x^0$, то $\omega \in \Omega_1$, якщо $x = x^0 < x_0$, і $\omega \in \Omega_2$, якщо $x = x^0 > x_0$.

Метод послідовного аналізу (метод Вальда) [1]. Використовується для диференціальної діагностики станів D_1 і D_2 і являє собою послідовну процедуру обстежень за допомогою системи простих незалежних ознак (бінарний, якісний або діагностичний інтервал кількісної ознаки), за умовою якої досягається заданий рівень вірогідності стану. Спочатку проводиться обстеження за ознакою x_1, за результатами якого визначається відношення правдоподібності

$$\Theta = \frac{P(x_1 / D_2)}{P(x_1 / D_1)}, \tag{1.63}$$

яке порівнюється з порогами A та B – відповідно верхньою і нижньою границею "області невизначеності", необхідної для ухвалення рішення.

Якщо $\Theta > A$, то робиться висновок на користь стану D_2, у противному разі, якщо $\Theta < B$, – на користь стану D_1. Якщо не досягнутий жоден поріг, тобто $B < \Theta < A$, то для ухвалення рішення проводиться додаткове обстеження за ознакою x_2 і т. д.

Для реалізації методу необхідно ранжувати ознаки за критерієм їхньої діагностичної цінності, а крім того, необхідно використовувати систему незалежних ознак.

1.3.2. Методи, засновані на теорії розпізнавання образів

У даній групі методів результати вимірювання характеристик об'єктів представляються точками в просторі діагностичних ознак. При цьому різні класи повинні утворювати компактні множини в просторі ознак. Діагностика нового об'єкта зводиться до обчислення міри близькості до кожного класу. Розрізняють такі групи методів:

1) Методи, засновані на принципі поділу (*R-моделі*). Основою зазначених методів є формування поділяючої поверхні або групи поверхонь, що щонайкраще розділяють елементи різних класів. До *R-моделей* належать:

- дискримінантний аналіз;
- метод порівняння з прототипом (еталоном);
- метод К-найближчих сусідів;
- метод січних площин;
- метод узагальненого портрета.

2) Методи, побудовані на основі "потенційних функцій" (*П-моделі*) Ці методи базуються на запозиченій з фізики ідеї електричного потенціалу, що визначений у будь-якій точці простору і змінюється в міру віддалення від заряду.

3) Методи обчислення оцінок (голосування) (*Г-моделі*) [11]. В основу цих методів покладено принцип часткової прецедентності, тобто прийняття рішень за аналогією.

4) Логіко-лінгвістичні методи (*Л-моделі*) – засновані на обчисленні висловлень, зокрема, на апараті алгебри логіки.

5) Методи, засновані на нечіткій логіці, на основі штучних нейронних мереж та інші.

В наступних підпунктах більш детально розглядаються відмічені методи.

Дискримінантний аналіз [1]. Якщо критеріальний показник y вимірюється у номінальній шкалі або якщо зв'язок цього показника з вихідними ознаками X є нелінійним й носить невідомий характер, то для визначення параметрів діагностичної моделі використовуються методи дискримінантного аналізу. У цьому випадку результати обстеження розбиваються на групи (класи), а ефективність діагностичної моделі розглядається під кутом зору її здатності розділяти (дискримінувати) класи, що діагностуються.

Показником якості дискримінантного аналізу є ймовірність помилкової класифікації досліджуваних об'єктів P_ε, яка мінімізується. У свою чергу, для розкриття взаємозв'язку P_ε зі структурою експериментальних даних у дискримінантному аналізі широко використовуються геометричні знання про поділ діагностуємих класів у просторі ознак.

При цьому вважається, що сукупність об'єктів, які належать до одного класу ω_i, утворює «хмару» у p-вимірному просторі R_p, що задається вихідними ознаками. Для успішної класифікації необхідно:

а) щоб хмара з ω_i в основному була сконцентрована в деякій області D_i простору R_p;

б) щоб в область D_i потрапила незначна частина «хмар» об'єктів, що відповідають іншим класам.

Тоді побудова вирішального правила розглядається як задача пошуку K непересічних областей D_i ($i = \overline{1,K}$), що задовольняють умови а і б. Дискримінантні функції (ДФ) дають визначення цих областей шляхом завдання їхніх границь у багатовимірному просторі R_p. Якщо об'єкт x попадає в область D_i, то приймається рішення про належність об'єкта до ω_i. Тоді критерієм правильного визначення областей буде

$$Q = \sum_{i=1}^{K-1} \sum_{j>i}^{K} P(\omega_i) P(\omega_i / \omega_j), \qquad (1.64)$$

де $P(\omega_i)$ – апріорна ймовірність появи об'єкта з класу ω_i в області D_i;

$P(\omega_i/\omega_j)$ – ймовірність того, що об'єкт із класу ω_j помилково попадає в область D_i, що відповідає класу ω_i.

Критерій Q називається критерієм середньої ймовірності помилкової класифікації. Мінімум Q досягається при використанні, зокрема, розглянутого вище байєсівського підходу, що, однак, може бути практично реалізований тільки при справедливості дуже сильного припущення про незалежність вихідних ознак, і в цьому випадку дає оптимальну лінійну діагностичну модель. Велика кількість інших підходів також використовує лінійні дискримінантні функції, але при цьому на структуру даних накладаються менш тверді обмеження.

Для випадку двох класів (ω_1 і ω_2) методи побудови лінійної дискримінантної функції (ЛДФ) спираються на два припущення. Перше полягає в тому, що області D_1 і D_2, у яких концентруються об'єкти з діагностуємих класів ω_1 і ω_2, можуть бути розділені (p-1)-вимірною гіперплощиною – дискримінантною функцією (ДФ):

$$y(X) = \sum_{i=0}^{p} a_i x_i \,, \qquad (1.65)$$

де p – кількість дискримінантних змінних; x_i – значення незалежних змінних; a_i – коефіцієнти (константи), що оцінюються за допомогою ДА.

Коефіцієнти a_i у цьому випадку інтерпретуються як параметри, що характеризують нахил гіперплощини до координатних осей, а a_0 називається порогом і відповідає відстані від гіперплощини до початку координат. Переважне розташування об'єктів одного класу, наприклад ω_1, по одну сторону гіперплощини виражається в тому, що для них, здебільшого, буде виконуватися умова $y(X)<0$, а для об'єктів іншого класу ω_2 – обернена умова $y(X)>0$. Друге припущення стосується критерію якості поділу областей D_1 і D_2 гіперплощиною (1.65). Найчастіше вважають, що поділ буде тим кращий, чим далі відстоятимуть одне від одного середні значення випадкових величин:

$$m_1 = E\{y(X)\}, \quad y \in \omega_1 \text{ і } m_2 = E\{y(X)\}, \; y \in \omega_2,$$

де $E\{\ \}$ – оператор усереднення.

У найпростішому випадку вважають, що класи ω_1 і ω_2 мають однакові коваріаційні матриці $S1=S2=S$.

Лінійний дискримінантний аналіз не тільки виявляє лінійні комбінації змінних для найкращого поділу заданих груп об'єктів, але і може використовуватися для зниження розмірності вхідних даних.

Як було зазначено раніше, результатом ДА є побудова дискримінантної функції виду (1.65).

Задачею дискриминантного аналізу є визначення таких коефіцієнтів a_i, щоб за значеннями ДФ можна було з мінімальною помилкою провести поділ усієї сукупності на класи. Якщо значення x_i нормовані, то чим більше значення коефіцієнтів a_i, тим більший внесок відповідної змінної в дискримінацію сукупності. Якщо класів більш двох, то будується декілька ДФ. Процедура ДА може проводитися двома методами:

– методом одночасного врахування змінних;

– покроковим методом.

Покроковий метод є одним із способів виключення зайвих змінних і полягає у використанні процедури послідовного добору найбільш корисних дискримінантних змінних, хоча отримана множина дискримінантних змінних може і не бути найкращою їх комбінацією.

У результаті застосування методу одночасного врахування змінних за наявності великої кількості змінних частина з них може виявитися зайвими, тобто такими, що не роблять ніякого внеску в аналіз, а лише ускладнюють його, деякі з них можуть нести однакову інформацію, будучи при цьому

хорошими дискримінаторами.

Щоб вибрати найбільш оптимальний за критерієм якості (Λ-статистика Уілкса) класифікації вид ДФ, дискримінантний аналіз необхідно проводити кожним з перерахованих вище методів.

Як міра розходження між дискримінантними змінними застосовується Λ-статистика Уілкса. Її значення обчислюється за формулою

$$\Lambda = \prod_{i=k+1}^{k_0} \frac{1}{1+\lambda_i}, \tag{1.66}$$

де k_0 – кількість станів (класів); k – кількість вже обчислених ДФ; λ_i – власне число i-ї ДФ, кожне з яких визначається при розв'язанні системи рівнянь

$$\begin{cases} \sum_{i=1}^{r} b_{1i}\nu_i = \lambda \sum_{i=1}^{r} w_{1i}\nu_i \\ \quad \ldots\ldots\ldots\ldots\ldots \\ \sum_{i=1}^{r} b_{ni}\nu_i = \lambda \sum_{i=1}^{r} w_{ni}\nu_i \end{cases}, \tag{1.67}$$

де r – кількість дискримінантних змінних; b_{ij} ($i,j = \overline{1,r}$) – елементи матриці $\boldsymbol{B}$ міжгрупової суми квадратів відхилень і попарних добутків; w_{ij} ($i,j = \overline{1,r}$) – елементи матриці $\boldsymbol{W}$ розкиду усередині класу; ν_i ($i = \overline{1,r}$) – послідовність коефіцієнтів, яка використовується для побудови ДФ, причому $\sum_{i=1}^{r} \nu_i = 1$.

Побудова ДФ зводиться до розв'язання системи рівнянь відносно λ і послідовності ν_i (кількість отриманих розв'язків відповідає кількості побудованих ДФ). Коефіцієнти кожної ДФ визначаються за формулою

$$a_i = \nu_i \sqrt{m - k_0}, \tag{1.68}$$

де m – загальне число спостережень по всіх класах.

Величина Λ є оберненою мірою, тобто її значення, близьке до нуля, говорить про високе розрізнення класів – центри класів добре розділені і сильно відрізняються один від одного стосовно ступеня розкиду усередині класів. Збільшення величини Λ до її максимального значення, що дорівнює 1, приводить до поступового погіршення розрізнення, тому що центри класів при цьому збігаються.

На основі Λ-статистики Уілкса одержують тест значущості, апроксимуючи розподіл деякої функції від неї розподілом χ^2. Для того щоб визначити рівень значущості цього розподілу, користуються спеціальними таблицями.

Метод порівняння з прототипом (еталоном). Даний метод найчастіше використовується, коли класи Ω_m ($m=\overline{1,M}$) утворюють компактні множини об'єктів, що мають сферичну форму в просторі ознак. У цьому випадку кожний із класів Ω_m описується прототипом або еталоном $\omega^{m\text{э}}$, у якості якого вибирається геометричний центр угруповання класу.

Невідомий об'єкт ω належить до класу Ω_t, відстань до прототипу якого $R(\omega,\omega^{t\text{э}})$ буде мінімальною:

$$R(\omega,\omega^{t\text{э}})=\min_{m=\overline{1,M}} R(\omega,\omega^{m\text{э}}), \qquad (1.69)$$

де $R(\omega,\omega^{m\text{э}})$ – відстань між об'єктом ω та еталоном $\omega^{m\text{э}}$ класу Ω_m;
M – кількість класів.

Метод Фікса–Ходжеса (метод "найближчих сусідів") [5]. Даний метод використовується в тому випадку, коли структура класів досить складна, далека від сферичної, або взагалі невідома, але підтверджується гіпотеза про безперервність багатовимірної щільності розподілу в кожній локальній області простору ознак. Суть даного методу полягає у визначенні деякого заданого числа k найближчих до невідомого об'єкта (в обраній метриці) об'єктів навчальної вибірки ("найближчих сусідів"). Невідомий об'єкт ω належить до того класу, число представників якого переважає серед обраних k "найближчих сусідів".

Реалізація методу. Визначають відстані $R(\omega,\omega_i)$ ($\text{i}=\overline{1,N}$), де N – об'єм навчальної вибірки, між невідомим об'єктом ω та усіма об'єктами навчальної вибірки, які являють собою сукупність всіх класів Ω_m ($m=\overline{1,M}$). Після цього знайдені відстані ранжуються за зростанням, вибираються перші k елементів – "найближчі сусіди", серед яких визначається клас Ω_t, число представників якого переважає серед обраних k "найближчих сусідів". До класу Ω_t і належить невідомий об'єкт ω.

Метод січних площин. Даний алгоритм полягає в апроксимації поділяючої поверхні «шматками» гіперплощин. Для формування поділяючої гіперповерхні необхідно: провести січні гіперплощини; виключити зайві гіперплощини; виключити зайві шматки гіперплощин.

Метод узагальненого портрета [1]. Даний метод дозволяє побудувати оптимальну нормально орієнтовану гіперплощину, що розділяє множини

векторів Ω_1 і Ω_2. Таке розбиття має місце, якщо для $k<1$ існує вектор ψ, для якого виконуються нерівності

$$\begin{aligned}&(\omega,\psi)\geq 1,\ \forall\omega\in\Omega_1,\\&(\omega,\psi)\leq k,\ \forall\omega\in\Omega_2.\end{aligned}\qquad(1.70)$$

Кожному значенню ψ, що задовольняє вираз (1.70) ставиться у відповідність гіперплощина $(\omega,\psi)=\dfrac{1+k}{2}$. Мінімальний по модулі вектор ψ, що задовольняє нерівностям (1.70), називається узагальненим портретом множини Ω_1 відносно Ω_2.

Метод потенційних функцій [1]. В якості потенційної функції, що характеризує належність об'єкта відповідному класові, використовується усюди позитивна і монотонно спадна функція відстані, аналогічна за формою електричному потенціалові φ. Прикладами таких функцій можуть бути

$$\varphi(R)=\left|\frac{\sin(\alpha R^2)}{\alpha R^2}\right|,\quad \varphi(R)=e^{-\alpha R^2}\ \text{ або }\ \varphi(R)=\frac{1}{1+\alpha R^2},\qquad(1.71)$$

де R – визначена будь-яким чином відстань між точкою-джерелом і точкою-приймачем, у якій обчислюється потенціал; $\alpha>0$ – ваговий коефіцієнт, який характеризує швидкість убування потенціалу φ.

Точками-джерелами потенціалу виступають об’єкти класу Ω_m, а точкою-приймачем – об’єкт ω, який підлягає класифікації. Об'єкт ω належить до класу Ω_t, сумарний потенціал якого буде максимальним.

Методи обчислення оцінок (голосування) [1]. Як було відмічено раніше, в основу цих методів покладено принцип часткової прецедентності, тобто прийняття рішень за аналогією. Між частинами описів розпізнаваного й еталонного об'єктів проводиться аналіз "близькості", наявність якої служить частковим прецедентом і оцінюється за деяким заданим правилом (за допомогою числової оцінки). Загальна оцінка розпізнаваного класу, отримана за набором оцінок близькості, і є значенням функції належності об'єкта класові. При цьому в алгоритмах розпізнавання, заснованих на обчисленні оцінок, ступінь подібності об'єктів обчислюється шляхом зіставлення всіх можливих (або визначених) сполучень ознак, що входять в опис об'єктів.

Логіко-лінгвістичні методи. Засновані на обчисленні висловлень, зокрема на апараті алгебри логіки. Тут як класи та ознаки об'єктів виступають логічні змінні. Цей клас методів можна розбити на два підкласи: логічні і лінгвістичні (структурні) [1, 5]. Логічні методи використовуються в тих випадках, коли є дані лише про детерміновані логічні зв'язки між об'єктами і їх ознаками, при цьому апріорна інформація про кількісний розподіл об'єктів по просторовим, часовим, ваговим, енергетичним або будь-яким іншим інтервалам у просторі ознак невідома. Ці відомості представляються у виді булевих співвідношень, що відображають причинно-наслідкові зв'язки між розглянутими класами об'єктів і їхніми ознаками. Належність розпізнаваного об'єкта до одного з класів визначається на підставі розв'язку булевих рівнянь.

Лінгвістичні або структурні методи побудовані на теорії формальних граматик. При цьому складні об'єкти описуються множиною незвідних (атомарних) елементів і набором граматичних правил. Рішення про належність розпізнаваного об'єкта приймається на підставі синтаксичного аналізу або граматичного розбору.

Методи, засновані на нечіткій логіці. Бурхливо розвивається в останнє десятиліття група методів, у яких для класифікації використовується теорія нечітких множин [12], і на її основі – нечіткі правила. Належність елемента x до нечіткої множини M задається безперервною функцією належності $\mu_M(x)$ ($0 \le \mu_M(x) \le 1$). Для кількісної оцінки правил, що близькі до речень природної мови, використовуються лінгвістичні змінні L, можливі значення яких є множиною термів $T(L)$, де кожен терм являє собою мітку нечіткої множини і задається своєю функцією належності $\mu_{Ti}(x)$.

У такий спосіб виконується перехід від числової змінної x до лінгвістичної L. Визначено логічні операції з нечіткими множинами і формалізовані правила нечіткого висновку, що використовуються для формування вирішальних правил. Вид функції належності $\mu_M(x)$ задається дослідником (**S**, **П**, трикутна, трапецеїдальна та ін.), а її параметри визначаються на навчальній вибірці за критерієм мінімуму помилки класифікації, для чого широке застосування одержали генетичні алгоритми [13]. Оскільки в основі методу лежать експертні оцінки, то він є серйозною альтернативою ймовірнісних методів при недостатньому обсязі навчальної вибірки або при її відсутності.

Методи розпізнавання на основі штучних нейронних мереж (ШНМ). Утворюють велику групу перспективних методів розпізнавання в умовах зашумлених і часто суперечливих даних [1, 12]. Однак використання

більшості ШНМ у системах підтримки прийняття рішень, які, як правило, повинні уточнювати свої знання в процесі експлуатації, утруднене існуючими методами навчання ШНМ. Наприклад, багатошарові ШНМ, що добре зарекомендували себе при вирішенні різноманітних задач розпізнавання і прогнозування, навчаються трудомісткими алгоритмами методу зворотного поширення помилки, застосування яких припускає наявність і використання всієї інформації про всі класи. Поява навіть одного нового класу в загальному випадку припускає повне перенавчання мережі. Подібна ситуація характерна і при використанні інших ШНМ, наприклад мережі Хопфілда, двохнаправленої асоціативної пам'яті, мережі Кохонена і т. д. Деякі з ШНМ не вимагають повного і трудомісткого перенавчання, наприклад мережа Хеммінга, однак ця мережа не може самостійно виділяти нову інформацію і самонавчатися.

Неможливість за допомогою відомих ШНМ вирішити проблему стабільності, тобто збереження раніше отриманих знань при запам'ятовуванні нової інформації, і проблему пластичності до нової інформації, тобто сприйняття нової інформації, що дозволяє як модифікувати й уточнювати збережені в пам'яті образи, так і створювати нові, зумовила необхідність розробки принципово нових ШНМ – мереж адаптивної резонансної теорії АРТ (ART – adaptive resonance theory). Ці мережі якоюсь мірою дозволяють вирішувати суперечливі задачі чутливості до нових даних і збереження раніше отриманих знань.

Нейронна мережа АРТ відносить вхідний об'єкт до одного з відомих класів, якщо він подібний або резонує з прототипом цього класу. Якщо знайдений прототип із визначеною точністю, що задається спеціальним параметром подібності, відповідає вхідному об'єктові, то він модифікується, щоб стати більш схожим на пред'явлений об'єкт. Коли вхідний об'єкт недостатньо подібний до жодного з наявних прототипів, то на його основі створюється новий клас, не спотворюючи характеристик існуючих класів.

Слід зазначити, що типологія методів, розглянута вище, є в деякому сенсі умовною. Існує ряд алгоритмів розпізнавання образів, які не можна однозначно віднести до тієї або іншої групи методів, описаних вище. Алгоритми персептронного типу, що реалізуються в різних архітектурах ШНМ можна віднести до *R*-моделей, тому що розбиття об'єктів на класи виконується шляхом формування поділяючої гіперповерхні зі шматків гіперплощин. З іншого боку, в ряді джерел показаний зв'язок алгоритмів персептронного типу з потенційними функціями.

На підставі вище викладеного можна зробити висновок, що для тих або інших методів і алгоритмів синтезу вирішальних правил є цілком конкретна область застосування, визначена апріорними обмеженнями. Як такі обмеження можуть розглядатися: характеристики ознак, у просторі яких

потрібно відрізняти об'єкти один від одного; клас вирішальних функцій; ймовірнісні характеристики класів і т. д. До того ж багато методів досить критичні до обсягу та репрезентативності навчальної вибірки, тому в системах підтримки прийняття рішень використовується розробка нових, модифікація і комбінація відомих методів.

1.3.3. Вирішальні правила в умовах суттєвої апріорної невизначеності

Байєсівський метод. Байєсівський підхід базується на припущенні, що завдання сформульоване в термінах теорії ймовірностей і відомі всі необхідні величини: апріорні ймовірності $P(\omega_i)$ для класів ω_i ($i = \overline{1,K}$) і умовні щільності розподілу значень вектора ознак $P(X/\omega_i)$ в кожному з класів. Правило Байєса полягає в знаходженні апостеріорної ймовірності $P(\omega_i/X)$, що обчислюється в такий спосіб:

$$P(\omega_i / X) = \frac{P(X / \omega_i)P(\omega_i)}{P(X)}, \tag{1.72}$$

де $P(X)$ – умовна щільність розподілу значень вектора ознак по всій вибірці, яка обчислюється за формулою

$$P(X) = \sum_{j=1}^{K} P(X / \omega_j)P(\omega_j). \tag{1.73}$$

Рішення про належність об'єкта A_0 з вектором ознак X_0 до класу ω_{ji} приймається при виконанні умови, яка забезпечує максимум апостеріорної ймовірності $P(\omega_i/X)$ і відповідно – мінімум середньої ймовірності помилки класифікації:

$$P(\omega_j / X_0) = \max_{i=\overline{1,K}} P(\omega_i / X_0). \tag{1.74}$$

Якщо розглядаються два діагностичних класи (ω_1 і ω_2), то відповідно до цього правила приймається рішення ω_1 при $P(\omega_1/X)>P(\omega_2/X)$ і ω_2 при $P(\omega_2/X)>P(\omega_1/X)$. Величину $P(\omega_i/X)$ у правилі Байєса часто називають правдоподібністю ω_i при даному X і прийняття рішення здійснюється через відношення правдоподібності або через його логарифм

$$L(X) = \log_2 \frac{P(\omega_1 / X)}{P(\omega_2 / X)}. \tag{1.75}$$

Для дихотомічних ознак, з якими в багатьох випадках доводиться мати

справу, p-вимірний вектор ознак X може набувати одного з $n=2^p$ дискретних значень v_1, ..., v_n. Функція щільності $P(X/\omega_i)$ стає сингулярною й заміняється на $P(v_k/\omega_i)$ – умовну ймовірність того, що $X = v_k$ за умови належності об'єктів до класу ω_i. На практиці в дискретному випадку, як і в безперервному, коли число вихідних ознак x_i велике, визначення умовних імовірностей зустрічає значні труднощі й найчастіше не може бути здійснене. З одного боку, це пов'язане, з нереальністю навіть простого перегляду всіх точок дискретного простору дихотомічних ознак. З іншого боку, навіть при набагато меншій кількості ознак для достовірної оцінки умовних імовірностей необхідно мати результати вимірювання досить великої кількості об'єктів.

Розповсюдженим способом подолання зазначених труднощів служить модель, в основі якої лежить припущення про незалежність вихідних дихотомічних ознак. Нехай для визначеності компонента вектора X набувають значення 1 або 0. Позначимо: $p_i = P(x_i = 1/\omega_1)$ – ймовірність того, що ознака x_i дорівнює 1 за умови належності об'єктів до діагностичного класу ω_1, і $q_i = P(x_i = 1/\omega_2)$ – ймовірність рівності 1 ознаки x_i у класі ω_2. У випадку $p_i > q_i$ варто очікувати, що i-та ознака буде частіше набувати значення 1 у класі ω_1, ніж в ω_2. У припущенні про незалежність ознак можна представити $P(X/\omega_i)$ у вигляді добутку ймовірностей:

$$P(X/\omega_1) = \prod_{i=1}^{p} p_i^{x_i}(1-p_i)^{1-x_i};$$
$$P(X/\omega_2) = \prod_{i=1}^{p} q_i^{x_i}(1-q_i)^{1-x_i}. \qquad (1.76)$$

Логарифм відношення правдоподібності в цьому випадку визначається в такий спосіб:

$$L(X) = \sum_{i=1}^{p}\left[x_i \log_2 \frac{p_i}{q_i} + (1-x_i)\log_2 \frac{1-p_i}{1-q_i}\right] + \log_2 \frac{P(\omega_1)}{P(\omega_2)}. \qquad (1.77)$$

З виразу (1.77) видно, що дане рівняння лінійно щодо ознак x_i. Тому можна записати:

$$L(X) = \sum_{i=1}^{p} w_i x_i + w_0, \qquad (1.78)$$

де вагові коефіцієнти w_i обчислюються за формулою

$$w_i = \log_2 \frac{p_1(1-q_i)}{q_1(1-p_i)},$$

а величина порога w_0 – за формулою

$$w_0 = \sum_{i=1}^{p} \log_2 \frac{1-p_i}{1-q_i} + \log_2 \frac{P(\omega_1)}{P(\omega_2)}.$$

Якщо $L(X)>0$, то приймається рішення про належність об'єкта до діагностичного класу ω_1, а якщо $L(X)<0$, то до класу ω_2.

Теорія Демпстера–Шефера та реалізація вирішальних правил на її основі [1]. У даному розділі розглядається один з методів формування нестрогих обчислень, який називається теорією Демпстера–Шефера, або теорією Шефера–Демпстера. Цей метод заснований на роботі Демпстера, який спробував змоделювати невизначеність, задаючи ряд ймовірностей, а не окреме ймовірнісне значення. Шефер доповнив і уточнив результати, отримані Демпстером, у своїй книзі *A Mathematical Theory of Evidence,* опублікованій в 1976 році. Теорія Демпстера–Шефера має добрий теоретичний фундамент. До того ж можна показати, що теорія коефіцієнтів достовірності (див. далі) є частковим випадком теорії Демпстера–Шефера, що дозволяє перевести методи, засновані на використанні коефіцієнтів достовірності на теоретичну основу.

Теорія Демпстера–Шефера заснована на припущенні про те, що задано фіксовану множину взаємовиключних і вичерпних елементів, яка називається *середовищем* і позначається грецькою буквою Θ:

$$\Theta = \{\Theta_1, \Theta_2, \ldots, \Theta_n\}.$$

Термін "середовище" аналогічний терміну "універсум", що застосовується в теорії множин. Таким чином, середовище – це множина об'єктів, які становлять для нас інтерес. Нижче наведено деякі приклади визначення варіантів середовища:

$$\Theta = \{\text{авіалайнер, бомбардувальник, винищувач}\}$$

Припустимо, що всі можливі елементи універсуму знаходяться в заданій множині, а отже, множина є вичерпною. Крім того, щоб спростити наведений тут опис, припустимо, що множина Θ є скінченою, хоча є варіанти середовища Демпстера–Шефера, елементами яких є безперервні змінні, такі як час, відстань, швидкість і т. д.

Один зі способів міркування про структуру множини полягає в тому, що розглядаються питання і відповіді, які відносяться до цієї множини. Припустимо, що дана наведена нижче множина, а питання, що стосується елементів цієї множини, сформульоване так: "Які з цих літаків мають військове призначення?"

Θ = {авіалайнер, бомбардувальник, винищувач}.

Відповіддю стає наступна підмножина множини Θ:

$\{\Theta_2, \Theta_3\}$ = {бомбардувальник, винищувач}.

Таким чином, кожна підмножина множини Θ може інтерпретуватися як можлива відповідь на деяке питання. А оскільки всі елементи є взаємовиключними і середовище – вичерпним, то правильною відповіддю на будь-яке питання може служити тільки одна підмножина.

Усі можливі підмножини в середовищі представлення типів літаків наведено на рис. 1.5.

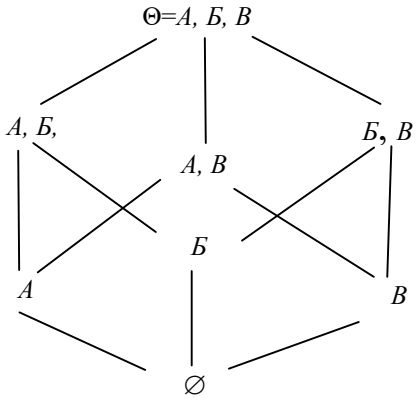


Рис. 1.5. Усі підмножини середовища, у якому розглядаються типи літаків

Лінії на цьому рисунку показують відношення між підмножинами. Для позначення елементів (авіалайнер, бомбардувальник і винищувач) використовуються букви *А, Б, В.* Ця діаграма зображена у вигляді ієрархічної решітки, у якій множина Θ знаходиться вгорі, а порожня множина $\varnothing$ = { } – унизу.

Один зі шляхів від вузла Θ до вузла $\varnothing$ виражає ієрархічне відношення підмножин, що з'єднує батьківські підмножини з дочірніми:

$$\varnothing \subset \{А\} \subset \{АБ\} \subset \{АБВ\}\ .$$

Якщо всі елементи середовища можуть інтерпретуватися як можливі відповіді і якщо тільки одна відповідь є правильною, то середовище називається *рамками розрізнення* (frame of discernment). У даному випадку термін "розрізнення" показує, що існує можливість відрізнити одну правильну відповідь від всіх інших можливих відповідей на питання.

Множина з кількістю елементів N має точно 2^N підмножини, включаючи саму себе. Усі ці підмножини визначають степеневу множину $P(\Theta)$. Таким чином, для середовища, що представляє типи літаків, маємо:

$$P(\Theta) = \{\varnothing, \{А\}, \{Б\}, \{В\}, \{А, Б\}, \{А, В\}, \{Б, В\}, \{А, Б, В\}\}.$$

Степенева множина середовища включає як свої елементи усі відповіді на всі можливі питання з рамок розрізнення. Це означає, що існує взаємно однозначна відповідність між елементами $P(\Theta)$ і підмножинами множини Θ.

Масові функції і незнання. Згідно з байєсівською теорією, апостеріорна імовірність при одержанні нових свідчень змінюється. Аналогічним чином у теорії Демпстера–Шефера може змінюватися ступінь довіри до свідчення. Крім того, у теорії Демпстера–Шефера прийнято розглядати ступінь довіри до свідчення аналогічно *масі* фізичного об'єкта, що позначається буквою *m,* яку можна переміщати, дробити і комбінувати.

Фундаментальне розходження між теорією Демпстера–Шефера і теорією ймовірностей полягає в тому, що в цих теоріях поняття *незнання* трактується по-різному. Відповідно до теорії ймовірностей за відсутності апріорних знань, ймовірність P кожного можливого випадку визначається формулою $P=1/N$, де N – кількість можливих випадків. Таке присвоювання значення P застосовується "у безвихідній ситуації" на підставі принципу байдужості. Крайній випадок застосування принципу байдужості виникає, якщо є тільки два можливих випадки, таких, як наявність або відсутність випадкової дії (наприклад, наявність нафти), що можна позначити символічно як H і H'. У випадках, подібних до цього, P = 50 %, навіть якщо взагалі немає ніяких знань про те, чи є нафта чи ні, оскільки теорія ймовірностей говорить, що справедлива наступна формула:

$$P(H) + P(H') = 1. \qquad (1.78)$$

Таким чином, усе, що не обґрунтовує гіпотезу, повинне її спростовувати, оскільки можливість незнання не припускається.

Якщо подібний підхід застосовується без міркувань, то можуть виявлятися деякі безглузді наслідки. Наприклад, припустимо, що людина міркує, чи є родовище нафти під її будинком чи його нема. Відповідно до принципу байдужості, якщо цілком відсутні які-небудь інші знання, то ймовірність наявності родовища нафти під будинком дорівнює 50 %. Варто тільки про це подумати, можна дійти висновку, що такі шанси наявності нафти є досить вражаючими і дають набагато кращу можливість швидко розбагатіти, чим за допомогою будь-яких інших законних капіталовкладень. А оскільки є 50 % шансів знайти нафту, то чи не варто негайно зняти усі свої заощадження, найняти бурову установку і приступити до буріння свердловини на кухні?!

Але навіть якщо принцип байдужості не використовується, наступне обмеження примусово диктує необхідність присвоювання ймовірності заперечення гіпотези і при відсутності свідчення, що відноситься до заперечення на основі формули (1.78).

Таким чином, теорія ймовірностей вимагає, щоб свідчення, що не обґрунтовує гіпотезу, спростовувало її. З іншого боку, теорія Демпстера–Шефера не змушує призначати ступінь довіри незнанню або спростуванню гіпотези. Замість цього маса привласнюється тільки тим підмножинам середовища, яким бажано призначити деякий ступінь довіри. Весь ступінь довіри, не привласнений конкретній підмножині, розглядається як ступінь відсутності довіри (no belief), або як ступінь недостачі довіри (nonbelief), і зв'язується із середовищем Θ. А ступінь довіри, що спростовує гіпотезу, являє собою *ступінь недовіри* (disbelief), який не слід плутати зі ступенем відсутності довіри.

Наприклад, припустимо, що деякий датчик, такий, як датчик системи впізнання "свій-чужий", не одержує відповіді від радіомаяка-відповідача літака. Робота системи впізнання "свій-чужий" заснована на використанні радіопередавача/приймача, що передає радіограму на літак і приймає відповідь. Якщо літак належить до власного повітряного флоту (є "своїм"), його радіомаяк-відповідач повинен відреагувати на радіограму, відправивши у відповідь ідентифікаційний код. Літаки, що не відповідають, за замовчуванням розглядаються як "чужі". Але літак може не відповісти на сигнали системи впізнання "свій-чужий" з багатьох причин, зокрема, з таких:

- несправність у системі впізнання "свій-чужий";
- несправність у радіомаяку-відповідачі літака;
- відсутність на літаку системи впізнання "свій-чужий";
- зникнення сигналу впізнання "свій-чужий" у перешкодах;
- одержання наказу дотримуватися режиму радіомовчання.

Припустимо, що невдала спроба системи впізнання "свій-чужий" одержати відповідь указує на наявність ступеня довіри 0,7 до свідчення, що розглянутий літак є "чужим", причому як "чужі" літаки розглядаються тільки бомбардувальники і винищувачі. Таким чином, присвоювання маси підмножині $\{Б, В\}$ здійснюється за наступною формулою, у якій m_1 позначає перше свідчення датчика впізнання "свій-чужий":

$$m_1(\{Б, В\})=0{,}7.$$

Інша частина ступеня довіри привласнюється середовищу Θ, як ступінь відсутності довіри:

$$m_1(\Theta) = 1 - 0{,}7 = 0{,}3.$$

Ми довіряємо гіпотезі, що розглянутий літак є "чужим", у ступені 0,7 і

резервуємо судження, що відповідає ступені 0,3, за недовірою і додатковою довірою до гіпотези, що літак "чужий".

Кожна підмножина в степеневій множині середовища, що має масу більше 0, розглядається як *фокальний елемент*, у якому фокусується (або концентрується) доступне свідчення.

У цьому теорія Демпстера–Шефера істотно відрізняється від теорії ймовірностей, у якій було б прийняте таке припущення:

$$P(\text{hostile}) = 0{,}7;\ P(\text{non-hostile}) = 1 - 0{,}7 = 0{,}3.$$

Як показано в табл. 1.5, застосування поняття маси забезпечує набагато більшу свободу вибору в порівнянні з поняттям ймовірності.

Таблиця 1.5 –Порівняння можливостей теорії Демпстера–Шефера і теорії ймовірностей

Теорія Демпстера–Шефера	Теорія ймовірностей
Значення $m(\Theta)$ не обов'язково повинне дорівнювати 1	$\sum_i P_i = 1$
Якщо $X \subseteq Y$, то вимога про дотримання рівності $m(X) = m(Y)$ не є обов'язковою	$P(X) \le P(Y)$
Не потрібна наявність зв'язку між $m(X)$ і $m(X')$	$P(X) + P(X') = 1$

Кожну масу можна формально представити за допомогою функції, що відображає кожен елемент степеневої множини в дійсне число, що знаходиться в інтервалі від 0 до 1. Зазначене відображення формально представляється за допомогою наступної формули:

$$m : P(\Theta) \to [0,1].$$

Відповідно до прийнятої угоди маса пустої множини зазвичай визначається як така, що дорівнює нулю:

$$m(\varnothing) = 0,$$

а сума мас усіх підмножин X степеневої множини дорівнює одиниці:

$$\sum_{X \in P(\theta)} m(X) = 1.$$

Комбінування свідчень. Тепер розглянемо випадок, у якому стають доступними додаткові свідчення. При цьому хотілося б мати можливість комбінувати усі свідчення, щоб зробити кращу оцінку ступеня довіри до свідчення. Для ознайомлення з тим, як це робиться, спочатку розглянемо приклад, у якому застосовується окремий випадок загальної формули комбінування свідчень.

Припустимо, що для розпізнавання літаків застосований датчик другого типу, який визначає розглянутий літак як бомбардувальник, зі ступенем

довіри до свідчення, що дорівнює 0,9. Тепер маси свідчень, отриманих від обох датчиків, набувають такого вигляду:

$$m_1(\{Б, В\})=0,7; \quad m_1(\Theta)=0,3;$$

$$m_2(Б)=0,9; \quad m_2(\Theta)=0,1.$$

У цих формулах змінні m_1 і m_2 відносяться до датчиків першого і другого типів. Отримані свідчення можна скомбінувати за допомогою наступної спеціальної форми правила комбінування Демпстера для одержання такої *комбінованої маси:*

$$m_1 \oplus m_2(Z) = \sum_{X \cap Y = Z} m_1(X) m_2(Y). \qquad (1.79)$$

У формулі (1.79) операція підсумування поширюється на всі елементи, для яких перетинання $X \cap Y = Z$. Знак операції $\oplus$ відповідає операції ортогональної суми, або прямій сумі. Результат цієї операції визначається шляхом підсумовування перетинань добутків мас правої частини правила. Правило комбінування Демпстера дозволяє комбінувати маси для одержання нової маси, що являє собою *консенсус* стосовно оригінальних, можливо конфліктуючих свідчень. Важливо відзначити, що це правило повинне застосовуватися для комбінування свідчень, що мають взаємно незалежні помилки, а це – не те ж саме, що незалежно зібрані свідчення.

У табл. 1.6 у наведено маси і перетинання добутків для середовища з літаками різних типів. Записи в цій таблиці були обчислені шляхом перехресного множення добутків мас по рядках і стовпцях, як показано нижче, де T_{ij} – i-й рядок та j-й стовпець таблиці.

Таблиця 1.6 – Підтвердження свідчень

	$m_2(Б) = 0,9$	$m_2(\Theta) = 0,1$
$m_1(\{Б, В\}) = 0,7$	$\{Б\} = 0,63$	$\{Б, В\} = 0,07$
$m_1(\Theta) = 0,3$	$\{Б\} = 0,27$	$\Theta = 0,03$

Слідом за обчисленням окремих добутків мас, відповідно до описаних формул, виконується додавання добутків за загальними правилами перетинання множин, відповідно до правила Демпстера:

$$m_3(\{Б\}) = m_1 \oplus m_2(\{Б\}) = 0,63 + 0,27 = 0,90 – \text{бомбардувальник};$$
$$m_3(\{Б, В\}) = m_1 \oplus m_2(\{Б, В\}) = 0,07 – \text{бомбардувальник або винищувач};$$
$$m_3(\Theta) = m_1 \oplus m_2(\Theta) = 0,03 – \text{відсутність довіри}.$$

Значення $m_3(\{Б\})$ виражає довіра до того, що розглянутий літак являє собою бомбардувальник і тільки бомбардувальник. Але під значеннями $m_3(\{Б, В\})$ і $m_3(\Theta)$ мається на увазі додаткова інформація. Відповідні множини включають бомбардувальник, тому правдоподібним є припущення,

що їхні ортогональні суми можуть зробити свій внесок у визначення ступеня довіри до того, що розглянутий літак є бомбардувальником. Тому значення суми 0,07 + 0,03 = 0,1 для цих множин може бути додане до ступеня довіри, що стосується підмножини бомбардувальника, для одержання максимального ступеня довіри до гіпотези, що літак може бути бомбардувальником, який дорівнює 0,90, тобто є правдоподібним ступенем довіри. Таким чином, ступінь довіри не обмежується одним значенням, а виражається у виді *ряду ступенів довіри* до свідчення. У даному випадку ряд ступенів довіри починається з мінімального значення 0,9, відповідно до якого відомо, що розглянутий літак — бомбардувальник, і закінчується максимальним правдоподібним значенням ступеня довіри 0,90 + 0,1 = 1, відповідно до якого цей літак може являти собою бомбардувальник. При цьому передбачається, що істинний ступінь довіри знаходиться десь у діапазоні від 0,9 до 1.

У таких міркуваннях на основі свідчень вважається, що свідчення задають *інтервал прояву свідчення* (evidential interval). При цьому в міркуваннях на основі свідчень нижня границя інтервалу називається *обґрунтуванням* (support – Spt); в теорії Демпстера–Шефера вона позначається як Bel (belief). З іншого боку, верхню границю прийнято називати правдоподібністю (plausibility – Pis). Для даного прикладу інтервал свідчень дорівнює [0,90, 1], тобто нижня границя дорівнює 0,90, а верхня границя – 1. Обґрунтування являє собою мінімальний ступінь довіри, заснований на свідченні, а правдоподібність — максимальний ступінь довіри, якого бажано досягти. Узагалі говорячи, діапазони, у яких змінюються Bel і Pis, виражаються співвідношенням

$$0 \le \text{Bel} \le \text{Pis} \le 1. \qquad (1.80)$$

У теорії Демпстера–Шефера нижню і верхню границі іноді називають нижньою і верхньою ймовірностями, відповідно до оригінальної статті Демпстера. У табл. 1.7 показані деякі широко застосовувані інтервали прояву свідчень.

Таблиця 1.7 – Деякі широко застосовувані інтервали прояву свідчень

Інтервал прояву свідчення	Область визначення змінної	Значення
[1, 1]		Цілком істинний.
[0, 0]		Цілком помилковий.
[0, 1]		Цілком невідомий.
[Bel, 1]	$0 < \text{Bel} < 1$	Як правило такий, що обґрунтовує.
[0, Pls]	$0 < \text{Pis} < 1$	Як правило такий, що спростовує.
[Bel, Pis]	$0 < \text{Bel} \le \text{Pis} < 1$	Такий, що і обґрунтовує, і спростовує.

Обґрунтування, або довірча функція, Bel, являє собою загальний ступінь довіри до множини і до всіх її підмножин. Таким чином, Bel – це вся маса, що обґрунтовує множину і визначається в термінах маси:

$$\text{Bel}(X) = \sum_{Y \subseteq X} m(Y) . \qquad (1.81)$$

Маса – це ступінь довіри до множини, а не до якої-небудь з її підмножин, а довірча функція застосовується до множини і до всіх її підмножин. Значення Bel являє собою сумарний ступінь довіри і тому є більш глобальним, ніж локальний ступінь довіри, що виражається масою. Маса і довірча функція зв'язані наступним співвідношенням:

$$m(X) = \sum_{Y \subseteq X} (-1)^{|X-Y|} \text{Bel}(Y) , \qquad (1.82)$$

де $|X - Y|$ – *кардинальність* множини $X - Y = \{x | x \in X \text{ і } x \notin Y\}$.

Таким чином, $|X - Y|$ – це кількість елементів у множині $X - Y$.

Отже, довірчі функції визначаються в термінах мас, тому комбінація двох довірчих функцій також може бути виражена в термінах ортогональних сум мас множини і всіх її підмножин, наприклад, так:

$$\text{Bel}_1 \oplus \text{Bel}_2(\{Б\}) = m_1 \oplus m_2(\{Б\}) + m_1 \oplus m_2(\varnothing) = 0{,}90 + 0 = 0{,}90.$$

У звичайному випадку маса порожньої множини не записується, оскільки вона дорівнює нулю. Сумарний ступінь довіри до підмножини $\{Б, В\}$, що складається з бомбардувальника і винищувача, включає більше підмножин, ніж наведена вище множина:

$$\text{Bel}_1 \oplus \text{Bel}_2(\{Б, В\}) = m_1 \oplus m_2(\{Б, В\}) + m_1 \oplus m_2(\{Б\}) + \\ + m_1 \oplus m_2(В) = 0{,}07 + 0{,}90 + 0 = 0{,}97.$$

У цей вираз включені терми для множин $\{Б\}$ і $\{В\}$, оскільки вони являють собою підмножини множини $\{Б, В\}$.

Підмножині $\{В\}$ маса не привласнена, тому $m(\{В\}) = 0$, і ця підмножина не робить ніякого внеску в суму. У дійсності $m(\{В\})$ і інші маси, що дорівнюють нулю, узагалі не вводилися в табл. 1.7, оскільки результат будь-якого перехресного добутку між ними дорівнює нулю. Якби маси були привласнені кожній підмножині множини $\{А, Б, В\}$, крім порожньої множини, то табл. 1.7 являла б собою таблицю з 49 комірок, відповідно до розрахунку $(2^3 - 1)(2^3 - 1) = 7 \cdot 7 = 49$.

Комбінована довірча функція для Θ, заснована на усіх свідченнях, має такий вигляд:

$$\text{Bel}_1 \oplus \text{Bel}_2(\Theta) = m_1 \oplus m_2(\Theta) + m_1 \oplus m_2(\{Б, В\}) + m_1 \oplus m_2(\{Б\}) =$$

$$= 0{,}03 + 0{,}07 + 0{,}90 = 1.$$

Фактично $\mathrm{Bel}(\Theta) = 1$ у всіх випадках, оскільки сума мас завжди повинна дорівнювати 1. При комбінуванні свідчень просто відбувається перерозподіл мас по різних підмножинах.

Інтервал прояву свідчення множини S, $EI(S)$ може бути визначений у термінах ступеня довіри в такий спосіб:

$$EI(S) = [\mathrm{Bel}(S), 1–\mathrm{Bel}(S')]. \qquad (1.83)$$

Якщо $S = \{Б\}$, то $S' = \{A, B\}$ і має місце наступна формула, оскільки є елементи, відмінні від фокальних, то маса нефокальних елементів дорівнює нулю:

$$\mathrm{Bel}(\{A, B\}) = m_1 \oplus m_2(\{A, B\}) + m_1 \oplus m_2(\{A\}) + m_1 \oplus m_2(\{B\}) = 0 + 0 + 0 = 0.$$

Таким чином, інтервал прояву свідчень для $\{Б\}$ визначається як:

$$EI(\{Б\}) = [0{,}90, 1 – 0] = [0{,}90, 1].$$

Аналогічним чином, якщо $S = \{Б, B\}$, то $S' = \{A\}$ і має місце наступна формула, оскільки $\{A\}$ – нефокальний елемент:

$$Bel(\{A\}) = 0.$$

Крім того, справедливі наведені нижче співвідношення, у яких інтервал прояву свідчень [0,1] відображає сумарний ступінь незнання стосовно підмножини $\{A\}$:

$$\mathrm{Be1}(\{Б, B\}) = \mathrm{Bel}_1 \oplus Bel_2(\{Б, B\}) = 0{,}97;$$

$$EI(\{Б, B\}) = [0.97, 1 – 0] = [0{,}97, 1];$$

$$EI(\{A\}) = [0, 1].$$

Метод обчислення коефіцієнтів достовірності. Даний метод був вперше застосований в комп'ютерній медичній системі MYCIN [1]. Ступінь підтвердження гіпотези H був спочатку визначений як коефіцієнт вірогідності, що визначається як різниця між *ступенем довіри* (belief) і *ступенем недовіри* (disbelief):

$$CF(H, E) = MB(H, E) – MD(H, E), \qquad (1.84)$$

де CF (Certainty Factor) – коефіцієнт вірогідності гіпотези H, обумовлений наявністю свідчення E;

MB (measure of belief) – міра підвищення ступеня довіри до гіпотези H з

огляду на наявність свідчення *E;*

MD (measure of disbelief) – міра підвищення ступеня недовіри до гіпотези *H* з огляду на наявність свідчення *E.*

Коефіцієнт достовірності – це спосіб об'єднання двох значень, ступеня довіри і ступеня недовіри, в одне число. Об'єднання мір довіри і недовіри в одне число здійснюється для досягнення двох цілей:

- насамперед, коефіцієнт достовірності може використовуватися для ранжирування гіпотез у порядку їхньої важливості;
- якщо в пацієнта є деякі симптоми, що свідчать про наявність декількох можливих захворювань, то необхідно, щоб на підставі медичних аналізів у першу чергу було проведене обстеження для діагностування саме того захворювання, якому відповідає найбільше значення *CF*.

Міри довіри і недовіри були визначені в термінах ймовірностей за такими формулами:

$$MB(H,E)=\begin{cases}1, & \text{якщо } P(H)=1;\\ \dfrac{\max[P(H/E),P(H)]-P(H)}{\max[1,0]-P(H)} & \text{якщо } P(H)<1;\end{cases} \quad (1.85)$$

$$MD(H,E)=\begin{cases}1, & \text{якщо } P(H)=0;\\ \dfrac{\min[P(H/E),P(H)]-P(H)}{\min[1,0]-P(H)} & \text{якщо } P(H)>0;\end{cases} \quad (1.86)$$

Відзначимо, що значення max[1,0] завжди дорівнює 1, а значення min[l, 0] завжди дорівнює 0. Але значення 1 і 0 записані в термінах max і min, оскільки це дозволяє показати формальну симетрію між виразами *MB* і *MD*. У табл. 5.1 показані деякі характерні випадки застосування значень *MB*, *MD* і *CF*, що визначені на підставі наведених вище формул.

Таблиця 1.8 – Деякі характерні випадки застосування значень *MB*, *MD* і *CF*

Характеристики	Значення
Інтервали значень	$0 \le MB \le 1$; $0 \le MD \le 1$; $-1 \le CF \le 1$
Деяка істинна гіпотеза $P(H/E) = 1$	$MB = 1$; $MD = 0$; $CF = 1$
Деяка помилкова гіпотеза $P(H'/E) = 1$	$MB = 0$; $MD = 1$; $CF = -1$
Відсутність свідчення $P(H/E) = P(H)$	$MB = 0$; $MD = 0$; $CF = 0$

Коефіцієнт достовірності *CF* показує, який чистий ступінь довіри до гіпотези, що заснований на деякому свідченні. Позитивне значення *CF* говорить про те, що свідчення обґрунтовує гіпотезу, оскільки *MB* > *MD*. Той випадок, у якому *CF* = 1, означає, що свідчення повністю доводить гіпотезу.

З іншого боку, випадок $CF = 0$ відповідає одній з двох можливостей. По-перше, $CF = MB - MD = 0$ може означати, що є нульові рівні і MB, і MD, тобто, що відсутнє яке-небудь свідчення. По-друге, можливо, що $MB = MD$ і обидва ці значення відмінні від нуля, а це означає, що довіра до гіпотези спростовується таким самим ступенем недовіри. На жаль, спростування сильного ступеня довіри такою самою недовірою веде не просто до незнання, але до деякого стану плутанини (а це набагато гірше). Наприклад, що може бути неприємнішим для водія, який під'їхав до перехрестя і не знає, куди повернути, та почув від пасажира, що указує вліво, пораду повернути праворуч!

У системі MYCIN значення *CF* антецедента правила повинне бути більше 0,2, для того щоб антецедент розглядався як істинний і активізував правило. Таке значення 0,2 розглядається як граничне значення, але воно не визначене як фундаментальна аксіома теорії коефіцієнтів вірогідності. Замість цього граничне значення розглядається як довільний спосіб зведення до мінімуму можливості активізації правил, що лише незначною мірою підтверджують гіпотезу. Без використання граничного значення можуть активізуватися численні правила, що є несуттєвими або які взагалі не мають значення, тому ефективність системи істотно зменшується.

У 1977 році приведене вище визначення *CF у* системі MYCIN було змінено і прийняло такий вид:

$$CF = \frac{MB - MD}{1 - \min(MB, MD)} \tag{1.87}$$

Це було зроблено з метою ослаблення впливу єдиних частин свідчень, що спростовують гіпотезу, на численні підтверджуючі частини свідчень. Якщо при використанні зазначеного визначення застосовуються наступні значення $MB = 0{,}999$; $MD = 0{,}799$ то значення *CF* приймає вид:

$$CF = \frac{0{,}999 - 0{,}799}{1 - \min(0{,}999,\ 0{,}799)} = \frac{0{,}200}{1 - 0{,}799} = 0{,}995$$

Це значення досить істотно відрізняється від значення, отриманого відповідно до попереднього визначення (1.84), при якому результат був $CF = 0{,}999 - 0{,}799 = 0{,}200$, і тому не відбувалася активізація правила, оскільки значення не було більше від граничного значення 0,2.

У табл. 1.9 показані правила, які застосовуються в системі MYCIN для комбінування свідчень в антецедентах правил. Слід зауважити, що ці правила збігаються з правилами системи PROSPECTOR, заснованими на нечіткій логіці.

Таблиця 1.9 – Правила комбінування свідчень

Свідчення E	Вірогідність антецедента
$E1$ AND $E2$	min[$CF(H, E1)$, $CF(H, E2)$]
$E1$ *OR* $E2$	max[$CF(H, E1)$, $CF(H, E2)$]
NOT E	-$CF(H, E)$

Наприклад, якщо дано наступний логічний вираз, що застосовується для комбінування свідчень

$$E = (E1 \text{ AND } E2 \text{ AND } E3) \text{ OR } (E4 \text{ AND NOT } E5),$$

то значення свідчення E можна обчислити в такий спосіб:

$$E = \max[\min(E1, E2, E3), \min(E4, -E5)].$$

При використанні значень $E1 = 0{,}9$; $E2 = 0{,}8$; $E3 = 0{,}3$; $E4 = -0{,}5$; $E5 = -0{,}4$ одержуємо наступний результат:

$$E = \max[\min(0{,}9; 0{,}8; 0{,}3), \min(-0{,}5; -(-0{,}4))] = \max[0{,}3; -0{,}5] = 0{,}3.$$

Більш складна ситуація відбувається у разі, коли самі свідчення є випадковими подіями з відомою невизначеністю. Тоді фундаментальна формула визначення коефіцієнта достовірності CF для правила

IF E THEN H

задається наступною формулою:

$$CF(H, e) = CF(E, e)CF(H, E), \qquad (1.88)$$

де $CF(E, e)$ — коефіцієнт достовірності свідчення E, що формує антецедент правила на основі невизначеного свідчення e;

$CF(H, E)$ — коефіцієнт достовірності гіпотези, у якій передбачається, що свідчення відоме з усією вірогідністю, коли $CF(E, e) = 1$;

$CF(H, e)$ — коефіцієнт достовірності гіпотези, заснованої на невизначеному свідченні e.

Таким чином, якщо усі свідчення в антецеденті відомі з усією вірогідністю, то формула для коефіцієнта достовірності гіпотези набуває наступного вигляду, оскільки $CF(E, e) = 1$:

$$CF(H, e) = CF(H, E).$$

Як приклад застосування таких коефіцієнтів достовірності розглянемо значення CF для правила визначення наявності стрептококової інфекції, описаного таким чином:

IF 1) The stain of the organism is gram positive, and

2) The morphology of the organism is coccus, and

3) The growth conformation of the organism is chains THEN There is suggestive evidence (0,7) that the identity of the organism is streptococcus.

У цьому правилі коефіцієнт достовірності гіпотези при наявності вірогідного свідчення визначається наступною формулою й іменується також коефіцієнтом ослаблення (attenuation factor):

$$CF(H, E) = CF(H, E1 \cap E2 \cap E3) = 0{,}7.$$

Це визначення коефіцієнта ослаблення засноване на припущенні, що усі свідчення ($E1$, $E2$ і $E3$) відомі з повною достовірністю. Таким чином, справедлива наступна формула, у якій e_i являють собою свідчення, які спостерігається:

$$CF(E1, e1) = CF(E2, e2) = CF(E3, e3) = 1.$$

Коефіцієнт ослаблення виражає ступінь вірогідності, що відноситься до гіпотези, якщо є деякі достовірні свідчення.

Якщо не відомі усі свідчення з повною достовірністю, то для визначення результуючого значення CF застосовується фундаментальна формула (1.88).

Наприклад, якщо в результаті статистичної обробки чи експертних оцінок визначені ймовірності свідчень

$$CF(E1, e1) = 0{,}5;\ CF(E2, e2) = 0{,}6;\ CF(E3, e3) = 0{,}3,$$

то у цьому випадку одержуємо такий результат:

$$CF(E, e) = CF(E1 \cap E2 \cap E3) = \min[CF(E1, e1), CF(E2, e2), CF(E3, e3)] = \\ = \min[0{,}5;\ 0{,}6;\ 0{,}3] = 0{,}3.$$

Тоді

$$CF(H, e) = CF(E, e)CF(H, E) = 0{,}3 \times 0{,}7 = 0{,}21.$$

1.3.4. Оцінка якості діагностичних моделей. Основи ROC-аналізу

Оцінка якості діагностичних кореляційних та регресійних моделей була розглянута у відповідних розділах.

Розглянемо випадок, коли модель диференціальної діагностики двох станів (D_0 – позитивний результат; D_1 – негативний результат) заснована на порогових вирішальних правилах виду

$$\text{IF } (x > r) \text{ THEN } D_0 \text{ ELSE } D_1,$$

де x – числове значення деякого діагностичного показника, а r – його порогове значення, необхідне для прийняття рішення щодо належності об’єкта діагностики до одного з класів (D_0 або D_1).

Тоді оцінка якості діагностичних моделей і відповідних до них порогових вирішальних правил, а також діагностична вага показника x виконується на основі чотириелементної таблиці спряженості (табл. 1.10), яка будується на результатах класифікації моделлю й на об'єктивній належності об'єктів до класів.

Таблиця 1.10 – Таблиця спряженості результатів класифікації

	Дійсно	
Модель	D_0	D_1
D_0	*TP*	*FP*
D_1	*FN*	*TN*

Що є позитивним результатом D_0, а що – негативним D_1, залежить від конкретної задачі. Якщо прогнозується захворювання, то D_0 – клас "Хворі", а D_1 – "Здорові". І навпаки, якщо визначається ймовірність того, що людина здорова, то D_0 – "Здорові", а D_1 – "Хворі". Елементами табл. 1.10 є:

- *TP* (True Positives) – кількість вірно класифікованих позитивних результатів;
- *TN* (True Negatives) – кількість вірно класифікованих негативних результатів;
- *FN* (False Negatives) – кількість позитивних результатів, що класифіковані як негативні (помилка I роду або "помилковий пропуск");
- *FP* (False Positives) – кількість негативних результатів, що класифіковані як позитивні (помилка II роду або "помилкове виявлення").

Об'єктивна цінність моделі визначається такими показниками:

- чутливістю (Sensitivity) $Se = TP/(TP + FN)\cdot 100$ %;
- специфічністю (Specificity) $Sp = TN/(TN + FP)\cdot 100$ %.

У медичній діагностиці, при класифікації пацієнтів на хворих і здорових, чутлива діагностична модель характеризується гіпердіагностикою – максимальному запобіганні пропуску хворих, а специфічна модель діагностує тільки істинно хворих. Це важливо у випадку, коли, наприклад, лікування хворого пов'язане із серйозними побічними ефектами і коли гіпердіагностика пацієнтів не бажана.

Інтегральним критерієм якості діагностичної моделі є ROC-аналіз [14, 15]. ROC-крива є графіком залежності

$$Se = f(100 - Sp)$$

при зміні параметра моделі, що керує точністю моделі – граничного елемента r. Площа під кривою ROC - AUC ($0{,}5 \le AUC \le 1$) використовується як скалярна міра оцінки прогностичної здатності моделі. У роботі [15] розглядається посилений ROC-аналіз, у роботі [14] проаналізовано

поводження ROC-кривої при нерівновеликих обсягах вибірок класів D_0 і D_1, отримано залежність імовірностей помилок I і II роду від асиметрії обсягів вибірок D_0 і D_1 і визначено умови оптимальності порога.

Контрольні питання

1. Класифікація математичних методів синтезу вирішальних правил.
2. Детерміновані методи синтезу вирішальних правил. Їх реалізація.
3. Класифікація ймовірнісних методів синтезу вирішальних правил.
4. Метод послідовного аналізу (метод Вальда). Його реалізація.
5. Класифікація методів синтезу вирішальних правил на основі теорії розпізнавання образів.
6. Принципи дискримінантного аналізу.
7. Метод порівняння із прототипом (еталоном). Його реалізація.
8. Метод K-найближчих сусідів. Його реалізація.
9. Метод потенційних функцій. Його реалізація.
10. Методи січних площин, узагальненого портрета та голосування.
11. Методи розпізнавання, засновані на нечіткій логіці.
12. Методи розпізнавання на основі штучних нейронних мереж.
13. Застосування ймовірнісної логіки при синтезі вирішального правила. Метод Байєса.
13. Реалізація методу Байєса для незалежних дихотомічних ознак.
14. Теорія Демпстера–Шефера. Її основні визначення та аксіоми.
15. Синтез вирішальних правил на основі логіки Демпстера–Шефера.
16. Синтез вирішальних правил на основі коефіцієнтів достовірності.
17. Оцінка якості діагностичних моделей. Основи ROC-аналізу.

Розділ 2. ПРИКЛАДИ ПОБУДОВИ МОДЕЛЕЙ СКЛАДНИХ ДИНАМІЧНИХ ОБ'ЄКТІВ

2.1. Методи синтезу моделей динамічних об'єктів. Системи диференціальних рівнянь і методи їхніх розв'язань

2.1.1. Метод Ейлера для розв'язання системи звичайних диференціальних рівнянь. Алгоритм його реалізації

Диференціальне рівняння першого порядку – це рівняння вигляду [1]

$$y' = f(x, y)\,, \tag{2.1}$$

із заданою функцією $f\ (x,\ y)$. Функція $y\ (x)$ називається розв'язком цього рівняння, якщо при всіх x виконується рівність

$$y'(x) = f(x, y(x))\,. \tag{2.2}$$

Ще Ньютон, Лейбніц і Ейлер помітили, що розв'язок звичайно містить вільний параметр, а тому він визначається єдиним чином тільки тоді, коли задане початкове значення

$$y(x_0) = y_0\,. \tag{2.3}$$

Французький математик Коші довів існування і єдиність такого розв'язку, тому задача (2.2), (2.3) називається задачею Коші.

Диференціальні рівняння виникають в багатьох застосуваннях: механіці, електриці, екології, біології та ін. Деякі з них можна вирішити в явному вигляді, тобто одержати розв'язок у вигляді формули або, як говорять, аналітично. Але це не завжди можливо, тому використовують різні методи чисельного розв'язання. Чисельні методи розв'язання задачі Коші для звичайних диференціальних рівнянь є наближеними, проте похибка обчислень відома наперед. Оцінки похибки дозволяють одержувати розв'язок диференціальних рівнянь із заданою точністю.

Система диференціальних рівнянь першого порядку має вигляд

$$\begin{aligned} y_1' &= f_1(x, y_1, \ldots, y_n); \qquad y_1(x_0) = y_{10} \\ &\ldots \\ y_n' &= f_n(x, y_1, \ldots, y_n); \qquad y_n(x_0) = y_{n0}. \end{aligned} \tag{2.4}$$

Розглянемо метод Ейлера для розв'язання диференціальних рівнянь, а також задачі Коші для звичайних диференціальних рівнянь. Цей метод був описаний Ейлером (1768) в його «Інтегральному численні». Відрізок

інтегрування розбиваємо на n відрізків точками

$$x_0, x_1, \ldots x_{n-1}, x_n, \tag{2.5}$$

і на кожному відрізку замінюємо розв'язок першим членом його ряду Тейлора:

$$\begin{aligned} y_1 - y_0 &= (x_1 - x_0) f(x_0, y_0); \\ y_2 - y_1 &= (x_2 - x_1) f(x_1, y_1); \\ &\ldots \\ y_n - y_{n-1} &= (x_n - x_{n-1}) f(x_{n-1}, y_{n-1}). \end{aligned} \tag{2.6}$$

Якщо з'єднати y_0 і y_1, y_1 і y_2 і т. д. прямими, то вийде ламана Ейлера, яка описує в першому наближенні шукану криву $y(x)$.

Наближений розв'язок $y(x)$ задачі Коші обчислюється за формулою

$$y_{i+1} = y_i + hf(x_i, y_i). \tag{2.7}$$

Перехід від y_i до y_{i+1} геометрично означає переміщення по дотичній до інтегральної кривої в кожній точці. На наступному кроці переміщення за відсутності похибки округлень відбувається по дотичній до (взагалі кажучи) іншої інтегральної кривої, що проходить через наступну точку. Початковою точкою, з якою починається побудова наближеного розв'язку, є точка, де y_0 – задане початкове значення невідомого розв'язку $y(x)$ рівняння (2.1).

За відсутності похибок округлень локальна похибка методу Ейлера, тобто похибка на одному кроці h, що виникає за рахунок переміщення по дотичній до інтегральної кривої, яка проходить через точку (x_i, y_i), а не по самій інтегральній кривій, є величина $O(h^2)$. Це випливає з розкладання за формулою Тейлора для розв'язання рівняння (2.1), яке проходить через точку (x_i, y_i). Глобальна похибка або, точніше, максимальна похибка розв'язку y(x) дорівнює $O(h)$, тобто вона на одиницю по порядку гірша, ніж локальна похибка. Основний недолік методу Ейлера – невисока щодо h точність. Це є метод першого порядку точності.

Метод простий для розуміння і програмування. Проте якщо бажано, скажімо, одержати 6 точних десяткових знаків, то потрібно виконати порядку мільйона кроків ($h=10^{-6}$), що є не дуже задовільним.

Є декілька шляхів побудови чисельних методів розв'язання задачі Коші більш високої по порядку щодо h точності. Один з них грунтується на використуванні розкладання розв'язання за формулою Тейлора (або, як ще говорять, розкладання в ряд). Проте на практиці переважно мати методи, що вимагають фактичного обчислення тільки значень правої частини рівняння (2.1), а не яких-небудь її похідних. Такими методами є методи Рунге–Кутта.

2.1.2. Метод Рунге–Кутта для розв'язання системи звичайних диференціальних рівнянь. Алгоритм його реалізації

Якщо ми маємо задачу

$$y' = f(x, y) \quad y(x_0) = y_0 , \tag{2.8}$$

те її розв'язання можна зобразити формулою

$$y(x) = y_0 + \int_{x_0}^{x} f(\varsigma) d\varsigma . \tag{2.9}$$

У цій формулі інтеграл потрібно обчислити, якнайбільше точно.

Як приклад розглянемо «правило середньої точки» (або формулу квадратури Гаусса)

$$\begin{aligned} y(x_0 + h) &= y_0 + hf(x_0 + \frac{h}{2}); \\ y(x_1 + h) &= y_1 + hf(x_1 + \frac{h}{2}); \\ &\ldots \\ y(x_{n-1} + h) &= y_{n-1} + hf(x_{n-1} + \frac{h}{2}), \end{aligned} \quad , \tag{2.10}$$

де h – довжина відрізків, на які розбитий інтервал інтегрування**.** Відомо, що оцінка глобальної похибки цієї формули має вигляд $O(h^2)$. Таким чином, якщо бажана точність складає 6 десяткових знаків, її звичайно можна одержати приблизно за тисячу кроків, тобто цей метод в тисячу раз швидший за метод Ейлера. Тому Рунге (1895) поставив наступне питання: чи не можна розповсюдити цей метод і на задачу (2.1)? Перший крок повинен мати вигляд

$$y(x_0 + h) \cong y_0 + hf(x_0 + \frac{h}{2}, y(x_0 + \frac{h}{2})) . \tag{2.11}$$

Але яке значення узяти для $y\left(x_0 + h/2\right)$? За відсутністю кращого природно використовувати один малий крок методом Ейлера довжиною $h/2$.

Тоді одержимо

$$\begin{aligned} k_1 &= f(x_0, y_0); \\ k_2 &= f(x_0 + \frac{h}{2}, y_0 + \frac{h}{2} k_1); , \\ y_1 &= y_0 + hk_2. \end{aligned} \tag{2.12}$$

Для обчислення k_2 використовуємо метод Ейлера з низькою точністю. Проте вирішальною обставиною є перемножування k_2 на h в останньому виразі, внаслідок чого вплив похибки стає менш істотним.

Таким чином, Рунге побудував новий метод, включивши один або два додаткових кроки за Ейлером. Кутта (1901) сформулював загальну схему того, що тепер називають методом Рунге–Кутта.

Найчастіше використовується метод Рунге–Кутта четвертого порядку точності:

$$\begin{aligned} k_1 &= hf(x_i, y_i); \\ k_2 &= hf(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}); \\ k_3 &= hf(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}); \\ k_4 &= hf(x_i + h, y_i + k_3); \\ y_1 &= y_0 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4). \end{aligned} \tag{2.13}$$

Локальна похибка цього методу є $O(h^6)$, а глобальна похибка методу Рунге–Кутта четвертого порядку дорівнює $O(h^4)$, що значно краще від методу Ейлера. У роботі [16] описаний алгоритм автоматичного вибору кроку в ході інтегрування диференціальних рівнянь на комп'ютері для отримання розв'язку з необхідною точністю.

Приклад 1

Розглянемо диференціальне рівняння з початковими умовами [16]

$$y' = y \quad y(0) = 1. \tag{2.14}$$

Це рівняння зручне тим, що можна знайти його аналітичний розв'язок

$$y = e^x, \tag{2.15}$$

і зробити порівняння з чисельним розв'язком на точність. Виберемо крок h=0,1. Знайдемо розв'язок цього рівняння в трьох точках. Отже,

$$
\begin{aligned}
&x_0 = 0, \qquad y_0 = 1;\\
&k_1 = hf(x_0, y_0) = 0{,}1 \cdot 1 = 0{,}100000;\\
&k_2 = hf(x_0 + \frac{h}{2}, y_0 + \frac{k_1}{2}) = 0{,}1 \cdot 1{,}050000 = 0{,}105000;\\
&k_3 = hf(x_0 + \frac{h}{2}, y_0 + \frac{k_2}{2}) = 0{,}1 \cdot 1{,}052500 = 0{,}105250;\\
&k_4 = hf(x_0 + h, y_0 + k_3) = 0{,}1 \cdot 1{,}105250 = 0{,}110525;\\
&y_1 = y_0 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) = 1{,}10517.
\end{aligned}
\tag{2.16}
$$

На наступному кроці маємо

$$
\begin{aligned}
&x_1 = 0{,}1; \qquad y_1 = 1{,}10517;\\
&k_1 = hf(x_1, y_1) = 0{,}1 \cdot 1 = 0{,}110517;\\
&k_2 = hf(x_1 + \frac{h}{2}, y_1 + \frac{k_1}{2}) = 0{,}1 \cdot 1{,}160429 = 0{,}116043;\\
&k_3 = hf(x_1 + \frac{h}{2}, y_1 + \frac{k_2}{2}) = 0{,}1 \cdot 1{,}163191 = 0{,}116319;\\
&k_4 = hf(x_1 + h, y_1 + k_3) = 0{,}1 \cdot 1{,}221489 = 0{,}122149;\\
&y_2 = y_1 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) = 1{,}22140.
\end{aligned}
\tag{2.17}
$$

І нарешті, робимо третій крок

$$
\begin{aligned}
&x_2 = 0{,}2, \qquad y_2 = 1{,}22140;\\
&k_1 = hf(x_2, y_2) = 0{,}1 \cdot 1{,}221400 = 0{,}122140;\\
&k_2 = hf(x_2 + \frac{h}{2}, y_2 + \frac{k_1}{2}) = 0{,}1 \cdot 1{,}282470 = 0{,}128247;\\
&k_3 = hf(x_2 + \frac{h}{2}, y_2 + \frac{k_2}{2}) = 0{,}1 \cdot 1{,}285524 = 0{,}128552;\\
&k_4 = hf(x_2 + h, y_2 + k_3) = 0{,}1 \cdot 1{,}221489 = 0{,}122149;\\
&y_3 = y_2 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) = 1{,}34986.
\end{aligned}
\tag{2.18}
$$

Наведемо таблицю значень e^x та порівняємо їх зі значеннями, що були отримані при чисельному розв'язанні рівняння (2.14).

Таблиця 2.1– Порівняння значень функції e^x з чисельним розв'язком

x	0,1	0,2	0,3
Значення чисельного розв'язку	1,10517	1,22140	1,34986
e^x	1,10517	1,22140	1,34986

Чисельний розрахунок за методом Рунге–Кутта збігається з аналітичним до п'ятого знака, тобто цей метод є достатньо точним. Таким чином, ми навчилися знаходити розв'язок одного диференціального рівняння.

Аналогічно можна побудувати розв'язання системи диференціальних рівнянь. Система звичайних диференціальних рівнянь з початковими умовами має вигляд

$$\begin{aligned} &y_1' = f_1(x, y_1, y_2, \ldots y_n); \\ &y_2' = f_2(x, y_1, y_2, \ldots y_n); \\ &\quad \ldots \\ &y_n' = f_n(x, y_1, y_2, \ldots y_n); \\ &y_1(x_0) = y_{10}; \\ &y_2(x_0) = y_{20}; \\ &\quad \ldots \\ &y_n(x_0) = y_{n0}. \end{aligned} \tag{2.19}$$

Введемо позначення $\overline{Y}$ – вектор значень шуканої функції (рядок),

$\overline{Y}^T = \begin{bmatrix} y_1 \\ y_2 \\ \ldots \\ y_n \end{bmatrix}$ – транспонований вектор значень шуканої функції (стовпець).

Тоді систему диференціальних рівнянь з початковими умовами можна переписати у такому вигляді:

$$\overline{Y}^{T'} = \overline{F}^T(x, \overline{Y}), \qquad \overline{Y}^T(x_0) = \overline{Y_0}^T. \tag{2.20}$$

Таким чином сформульована задача Коші (2.20) для систем звичайних диференціальних рівнянь. Алгоритм розв'язання методом Рунге–Кутта такої системи має вигляд:

$$
\begin{aligned}
&\overline{k_1}^T = h\overline{F}^T(x_i, \overline{Y_i});\\
&\overline{k_2}^T = h\overline{F}^T(x_i + \frac{h}{2}, \overline{Y_i} + \frac{\overline{k_1}}{2});\\
&\overline{k_3}^T = h\overline{F}^T(x_i + \frac{h}{2}, \overline{Y_i} + \frac{\overline{k_2}}{2});\\
&\overline{k_4}^T = h\overline{F}^T(x_i + h, \overline{Y_i} + \overline{k_3});\\
&\overline{Y}_{i+1}^T = \overline{Y}_i^T + \frac{1}{6}(\overline{k}_1^T + 2\overline{k}_2^T + 2\overline{k}_3^T + \overline{k}_4^T).
\end{aligned}
\tag{2.21}
$$

Послідовно застосовуючи до кожного рівняння метод Рунге-Кутта, одержимо розв'язок системи звичайних диференціальних рівнянь.

2.1.3. Системи диференціальних рівнянь із запізнюванням. Приклади їх реалізації й методи розв'язання

Диференціальні рівняння з аргументом, що запізнюється, це – рівняння вигляду [1, 16]

$$y' = f(x, y(x), y(x-\tau)). \tag{2.22}$$

Відмінність цього виду рівнянь від звичайних диференціальних рівнянь полягає у тому, що похідна розв'язку залежить не тільки від однієї попередньої точки, як це було для звичайних диференціальних рівнянь, але і від його значень в попередніх точках, а саме на інтервалі τ.

Запізнювання зустрічається в багатьох моделях прикладної математики; воно також може бути джерелом цікавих математичних явищ, таких, як нестійкість, граничні цикли, періодичний характер розв'язків.

Питання існування розв'язку вирішується таким чином. Припустимо, що на відрізку $x_0 - \tau \le x \le x_0$ розв'язок відомий:

$$y(x) = \varphi(x). \tag{2.23}$$

Тоді на відрізку $x_0 \le x \le x_0 + \tau$ функція $y(x-\tau)$ є відомою функцією x і рівняння (2.22) стає звичайним диференціальним рівнянням, до якого можуть бути застосовані відомі теореми існування. Визначивши $y(x)$ при $x_0 \le x \le x_0 + \tau$ ми можемо потім перейти до обчислення розв'язку при $x_0 + \tau \le x \le x_0 + 2\tau$ і т. д. Цей «метод кроків», таким чином, для всіх x дозволяє одержати розв'язок. Розглянемо наступні приклади.

Приклад 2

Просте диференціальне рівняння з аргументом, що запізнюється, розв'язуємо аналітично, використовуючи метод кроків. Задане рівняння

$x'(t) = 6x(t-1)$ з початковими умовами при $0 \le t \le 1 \quad x(t) = t$;

Знайти $x(t)$ на відрізках [1,2] і [2,3].

Розглянемо інтервал [1,2].

Спочатку визначимо праву частину диференціального рівняння на цьому інтервалі. При $0 \le t \le 1 \quad x(t) = t$. Звідси, якщо прийняти $t_1 = t - 1$, то з $x(t_1) = t_1$ випливає, що $x(t-1) = t - 1$.

Тепер можна записати диференціальне рівняння на цьому інтервалі $x'(t) = 6 \cdot (t-1)$.

Інтегруємо отримане рівняння як звичайне диференціальне рівняння, маємо $x(t) = 3 \cdot (t-1)^2 + C_1$;

Визначимо константу C_1, маємо $x(1) = C_1 = 1; \Rightarrow C_1 = 1$. Таким чином, на інтервалі $[1,2]$ розв'язок має вигляд $x(t) = 3 \cdot (t-1)^2 + 1$. Для подальшого розв'язання визначимо початкові умови наступного відрізку $x(2)$=4.

Розглянемо інтервал [2,3].

На інтервалі $2 \le t \le 3$ рівняння $x'(t) = 6x \cdot (t-1)$ набуває вигляду $x'(t) = 6 \cdot [3(t-2)^2 + 1]$. Справді,

$$\begin{bmatrix} x(t-1) = & 2 \le t \le 3 \\ = x(t_1) = & \\ = 3(t_1 - 1)^2 + 1 = 3(t-2)^2 + 1 & 1 \le t \le 2 \end{bmatrix}.$$

Проінтегруємо отримане рівняння як звичайне диференціальне рівняння, маємо

$$x(t) = 6(t-2)^3 + 6t + C_2; \qquad x(2) = 12 + C_2 = 4; \Rightarrow C_2 = -8.$$

Таким чином, на інтервалі $2 \le t \le 3$ розв'язок має вигляд

$$x(t) = 6(t-2)^3 + 6t - 8.$$

Зверніть увагу, на те, що на кожному інтервалі розв'язок диференціального рівняння з аргументом, що запізнюється, можна записати за допомогою своєї власної формули, тобто вигляд розв'язку залежить від розв'язку на попередньому інтервалі.

Приклад 3

Хай дане рівняння $y'(x) = -y(x-1)$ та навчальна умова $y(x) = 1$ при $-1 \le x \le 0$.

Застосовуючи описаний вище метод кроків, одержимо $y(x) = 1 - x$ при $0 \le x \le 1$.

Після інтегрування маємо

$$y(x) = 1 - x + (x-1)^2/2! \quad \text{при } 1 \le x \le 2.$$

На наступному інтервалі, проробляючи все те саме, отримаємо

$$y(x) = 1 - x + (x-1)^2/2! - (x-2)^3/3! \quad \text{при } 2 \le x \le 3.$$

Цей приклад ясно показує, що розв'язки рівняння з аргументом, що запізнюється, залежать не тільки від початкової умови, але і від всієї історії процесу від $x_0 - \tau$ до x_0.

Тепер визначимо, як диференціальне рівняння з аргументом, що запізнюється, можна розв'язати чисельними методами.

Застосовуючи метод Рунге–Кутта до рівняння з аргументом, що запізнюється, одержимо такі формули для розрахунку:

$$\begin{aligned}
k_1 &= hf(x_i, y(x_i), y(x_i - \tau)); \\
k_2 &= hf(x_i + \frac{h}{2}, y(x_i) + \frac{k_1}{2}, y(x_i + \frac{h}{2} - \tau)); \\
k_3 &= hf(x_i + \frac{h}{2}, y(x_i) + \frac{k_2}{2}, y(x_i + \frac{h}{2} - \tau)); \\
k_4 &= hf(x_i + h, y(x_i) + k_3, y(x_i + h - \tau)); \\
y_1 &= y_0 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4).
\end{aligned} \qquad (2.24)$$

Виникає питання, як задавати значення $y(x_i + \frac{h}{2} - \tau)$ і $y(x_i + h - \tau)$. Але якщо запізнювання постійне і довжина кроку інтегрування вибрана так, що $\tau = kh$, де k – деяке ціле число, то природно використовувати вже обчислені значення розв'язків на попередньому відрізку τ. Тому всі набуті значення розв'язку на попередньому відрізку необхідно зберегти в деякому допоміжному масиві і переписувати його значення на кожному новому відрізку τ.

Контрольні питання

1. Сформулюйте задачу Коші.
2. Що таке початкові умови?
3. Як виглядає розв'язок диференціального рівняння?
4. Що таке система диференціальних рівнянь?
5. Які засоби розв'язання диференціальних рівнянь ви знаєте?
6. У чому полягає основна ідея чисельного розв'язання диференціальних

рівнянь?

7. Як визначити алгоритм методу Ейлера?
8. Яка точність методу Ейлера?
9. Чим відрізняється метод Рунге–Кутта від методу Ейлера?
10. Яка точність метода Рунге–Кутта?
11. Що таке диференціальні рівняння з аргументом, що запізнюється?
12. Де використовуються диференціальні рівняння з аргументом, що запізнюється?
13. Чим відрізняються диференціальні рівняння з аргументом, що запізнюється, від звичайних диференціальних рівнянь?
14. Опишіть алгоритм чисельного розв'язання диференціальних рівнянь з аргументом, що запізнюється.

2.2. Моделювання розвитку популяцій

2.2.1. Моделі розвитку популяцій. Види взаємодії популяцій

Проблема зростання населення стала привертати до себе увагу у всьому світі з тих пір, як Т. Р. Мальтус в 1789 році запропонував свою теорію зростання чисельності популяції організмів [1]. Відповідно до цієї теорії людство може вижити, тільки якщо періоди зростання в геометричній прогресії уриватимуться епідеміями і стихійними бідами.

Ця теорія була піддана критиці К. Марксом. Дійсно, модель Мальтуса задовільно описує тільки дуже короткі періоди життєвого циклу організмів і не є універсальною. Запропонована Мальтусом модель зростання популяції має ряд обмежень, але заслуга Мальтуса у тому, що він вперше звернув увагу на цю проблему і дав математичне формулювання задачі, що поклало початок цілому напряму досліджень в біології – математичній екології.

Вже в XX ст. Ерліх і Медоуз знов звернулися до цієї проблеми і розглянули експоненціальне зростання населення у взаємозв'язку з виснаженням невідновлюваних ресурсів. Математичні моделі дозволяють описувати статеву або територіальну поведінку популяцій, а також невеликі фрагменти біологічної системи, наприклад, моделі популяції одного шкідника бавовника на даному полі. Такі моделі називаються тактичними. Стратегічні моделі жертвують точністю в спробі охопити загальні закони. Моделі популяцій створюють на формальній мові схему найважливіших біологічних процесів, таких, як народження і смерть. Аналіз моделей використовується для проникнення в суть цих біологічних процесів, особливо процесів біологічної взаємодії, яка може надалі визначити області, де дослідження приведе до пояснення основних механізмів регуляції чисельності в природних популяціях. В даний час подібні моделі

використовуються для опису зростання мікроорганізмів, водних співтовариств, а також у області боротьби з комахами-шкідниками. Деякі з цих моделей ми розглянемо надалі.

Всі подібні моделі можна розділити на два класи: моделі зростання популяції (одновидові моделі) і моделі взаємодії популяцій (моделі для двох або більш видів).

Перейдемо до їх розгляду. Проста модель Мальтуса для темпів зростання чисельності популяції організмів була сформульована у вигляді одного диференціального рівняння. У цій моделі як невідома функція береться кількість особин в популяції.

Популяція – це сукупність особин певного вигляду, які тривалий період часу займають певний простір, званий ареалом. Модель Мальтуса розглядає популяцію, яка розвивається ізольовано в умовах необмеженого ареалу і необмежених харчових ресурсів.

Модель Мальтуса записується у вигляді [1]

$$\frac{dx}{dt} = kx\,, \tag{2.25}$$

де x (t) – чисельність популяції; k – деякий коефіцієнт, що характеризується двома чинниками: народженням і смертю. Це рівняння може бути одержане з таких міркувань. Приріст чисельності популяції Δx, який можна виразити формулою

$$\Delta x = x(t + \Delta t) - x(t)\,. \tag{2.26}$$

водночас є пропорційним наявній кількості особин $x(t)$ у момент часу t з коефіцієнтом $k=a-b$, де a і b – відповідно миттєва народжуваність і смертність у популяції. Ці коефіцієнти відображають швидкість розмноження особин даного виду, тобто умови їх існування.

Приріст чисельності популяції пропорційний також відрізку часу, за який відбулося зростання Δt. Одержимо співвідношення

$$\Delta x = (a - b)x(t)\Delta t\,, \tag{2.27}$$

Розділивши ліву і праву частини співвідношення (2.3) на Δt, одержимо

$$\frac{\Delta x}{\Delta t} = (a - b)x(t)\,. \tag{2.28}$$

Рівняння (2.28) і є диференціальним рівнянням (2.25), переписаним інакше. Для розв’язання диференціального рівняння потрібно задати початкові умови, тобто вказати чисельність популяції в якийсь момент часу. Зручно брати час $t=0$, бо це не впливає на загальний вигляд поставленої

задачі. Тоді проста модель зростання популяції буде мати вигляд задачі Коші:

$$\begin{aligned} &\frac{dx}{dt} = kx; \\ &x\big|_{t=0} = x_0. \end{aligned} \tag{2.29}$$

Модель (2.29) має аналітичний розв'язок

$$x(t) = e^{kt}. \tag{2.30}$$

і називається моделлю експоненціального зростання. Таким чином, модель дає уявлення, що ідеалізується, про зростання абсолютно однорідної популяції, особини якої повністю тотожні, популяції нічим не обмеженої в своєму зростанні і тому здатної досягати скільки завгодно великої чисельності. Аналіз формули (2.6) приводить до висновку, що у випадку $a>b$ $x(t)\to\infty$ при $t\to\infty$, популяція необмежено зростає; якщо $a<b$ $x(t)\to 0$ при $t\to\infty$, то популяція гине. Модель Мальтуса має перш за все теоретичний інтерес, але може застосовуватися в окремих випадках. Наприклад, масове і швидке виробництво антибіотиків (точніше, колоній пліснявих грибків, які виділяють пеніцилін) якраз засноване на тенденції необмеженого зростання популяції.

Відмовимося від умови ізольованості ареалу і вважатимемо, що з сусіднього ареалу до неї відбувається постійне поповнення [1]. Хай m – чисельність особин, що прибувають за одиницю часу. Тоді маємо рівняння

$$\frac{dx}{dt} = kx + m. \tag{2.31}$$

Розв'язок рівняння (2.31) має вигляд

$$x(t) = Ce^{kt} - \frac{m}{k}. \tag{2.32}$$

Визначивши постійну C з початкових умов $x(0)=x_0$, одержуємо

$$x(t) = \left(x_0 + \frac{m}{a-b}\right)e^{(a-b)t} - \frac{m}{a-b}. \tag{2.33}$$

З формули (2.33) видно, що у випадку $a<b$ число особин $x(t)$ наближається до граничного значення $m/(b-a)$ при $t\to\infty$, що є реальнішим, ніж необмежене зростання в моделі Мальтуса.

Відмовимося ще і від умови необмеженості ареалу, враховуючи вплив боротьби усередині популяції за загальні ресурси. Цей вплив зменшить

швидкість приросту популяції. Таку модель запропонував П. Ф. Ферхюльст, відповідно до якої зменшення пропорційне квадрату кількості особин $x(t)$ з коефіцієнтом $\beta>0$.

Маємо

$$\frac{dx}{dt} = (a-b)x - \beta x^2 . \qquad (2.34)$$

Квадратичний член $-\beta x^2$ в рівнянні (2.34) указує на те, що спад особин в умовах обмеженості ресурсами в порівнянні із зростанням вільної популяції пропорційний кількості контактів («зіткнень») між особинами.

Введення члена $-\beta x^2$ в рівняння (2.25) дозволяє уникнути біологічної некоректності моделі Мальтуса – необмеженого зростання популяції. Розглянемо розв'язок рівняння (2.34), він має вигляд

$$x(t) = \frac{x_0\gamma}{(\gamma - x_0) + x_0 e^{-(a-b)(t-t_0)}} + x_0 \quad , \qquad (2.35)$$

де $\gamma = (a-b)/\beta$.

При будь-якому початковому стані популяції її чисельність монотонно наближується до деякого значення. Графік функції $x(t)$ наведено на рис. 2.1.

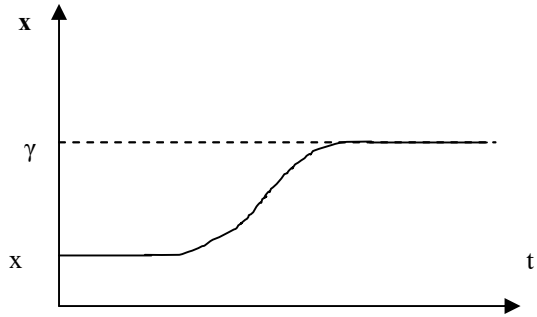


Рис. 2.1 - Графік функції $x(t)$ в моделі Ферхюльста

Аналіз розв'язку (2.35) приводить до висновку, що $x(t) \to \gamma$ при $t \to \infty$. Функція $x(t)$ зростає, і графік її (рис. 2.1) асимптотично наближається до прямої $x(t) = \gamma$. Це означає, що чисельність популяції в моделі Ферхюльста хоча і зростає, проте не може бути більше γ. Величина γ – це теоретично максимально можлива чисельність популяції. Крива на рис. 2.1 нагадує витягнуту букву S. Така крива називається логістичною, і відповідно модель теж іноді називають логістичною. Графік функції $x(t)$ на рис 2.1 добре узгоджується з виглядом експериментальних кривих розвитку багатьох популяцій в умовах обмеженого ареалу. Якщо припустити, що швидкість зростання залежить від чисельності попереднього покоління, то рівняння

(2.34) набуває вигляду

$$\frac{dx}{dt} = (a-b)x - \beta x(t-\tau)x, \tag{2.36}$$

де τ – час запізнювання, в даному випадку середній проміжок часу між двома поколіннями. Диференціальні рівняння із запізнюванням часто використовуються в задачах біології, і ми навели один із прикладів такого використання.

Дотепер ми розглядали вільну (ізольовану) популяцію без урахування її взаємодії з іншими популяціями. Математичне моделювання популяцій дозволяє досліджувати не тільки зростання окремої популяції, але і їх міжвидові відносини, при цьому моделі динаміки вільної популяції служать основою для моделей взаємодіючих популяцій. Для опису міжвидових відносин використовуються системи диференціальних рівнянь. Загальний вигляд рівнянь взаємодії можна записати так [16]:

$$\begin{aligned} \frac{dx}{dt} &= A(x) + k_x B(x,y); \\ \frac{dy}{dt} &= C(y) + k_y D(x,y), \end{aligned} \tag{2.37}$$

де $x(t)$, $y(t)$ – чисельності популяцій; функції $A(x)$, $C(y)$ описують поведінку вільних популяцій x, y; відповідно функції $B(x,y)$, C (x,y) – міжвидові відносини популяцій. У екології прийнято виділяти три основні типи міжвидових відносин. Визначення знаків похідних в системі рівнянь (2.37) визначає характер відносин між популяціями відповідно до загальноприйнятої класифікації.

У табл. 2.2 наведені типи взаємодій в екосистемах і дана їх математична інтерпретація.

Таблиця 2.2 – Типи взаємодії популяцій

Біологічна інтерпретація типу взаємодії	Значення коефіцієнтів	
	k_x	k_y
Симбіоз	+1	+1
Конкуренція за загальний ресурс	-1	-1
Хижак–жертва	+1	-1

У таблиці 2.2 знак «+» означає позитивну, сприятливу дію одного вигляду на інший, а знак «-» – несприятливу. Відносини хижак-жертва є найістотнішими для функціонування екосистем, розглянемо їх докладніше.

2.2.2. Конкуренція популяцій

Перша модель, що описує динаміку чисельності двох популяцій, була запропонована незалежно А. Лоткой і В. Вольтера [16]:

$$\begin{aligned} \frac{dx}{dt} &= ax - bxy; \\ \frac{dy}{dt} &= -cy + dxy, \end{aligned} \tag{2.38}$$

де $x(t)$, $y(t)$ – чисельності популяції жертви і хижака відповідно, a – швидкість розмноження популяції жертви за відсутності хижака; b – питома швидкість споживання популяцією хижака популяції жертви; c – природна смертність хижака; d/b – коефіцієнт переробки спожитої біомаси жертви у власну біомасу.

У основу моделі покладено такі уявлення про характер внутрішньовидових і міжвидових взаємодій в системі хижак–жертва:

1. За відсутності хижака популяція жертви розмножується відповідно до принципу Мальтуса – експоненційно.

2. Популяція хижака за відсутності жертви експоненційно вимирає.

3. Сумарна кількість жертви, споживана популяцією хижака в одиницю часу, лінійно залежить і від густини популяції жертви, і від густини популяції хижака.

4. Споживана хижаком біомаса жертви з постійним коефіцієнтом переробляється в біомасу хижака.

5. Інші чинники, що роблять вплив на динаміку популяцій, не враховуються.

У системі (2.38) чотири параметри. Введемо нові змінні

$$x = \frac{c}{d}u; \quad y = \frac{a}{b}v; \quad t = \frac{\tau}{a}. \tag{2.39}$$

Приходимо до системи рівнянь

$$\begin{aligned} \frac{du}{d\tau} &= u - uv; \\ \frac{dv}{d\tau} &= -\gamma v + uv, \end{aligned} \tag{2.40}$$

де $\gamma = b/a$.

На фазовому портреті (рис. 2.2) системи хижак–жертва відображається періодично повторювана картина, що має місце в реальній взаємодії хижака та жертви. Якщо u – це жертва, а v – це хижак, то при обході кривих на

фазовому портреті маємо такі наслідки їх взаємодії. При деякому постійному значенні хижаків відбувається нарощування жертв (*u* збільшується), далі нарощується число хижаків при деякому спаданні числа жертв.

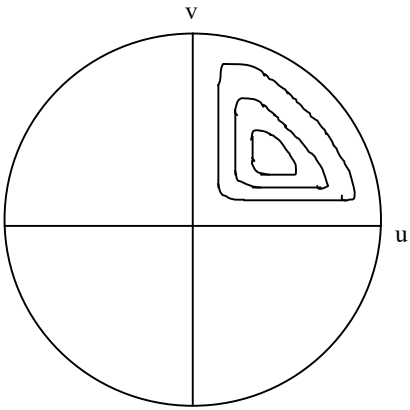


Рис. 2.2 - Схематичне зображення фазового портрета системи (2.40)

При деякому критичному значенні жертв починається різкий спад хижаків до значення рівноваги. Після цього кількість хижаків утримується на постійному рівні, а кількість жертв поступово зростає, і все повторюється за описаним сценарієм. Це відповідає і сезонним змінам у взаємодії популяцій. Описані характерні зміни чисельності популяцій хижака і жертви спостерігаються в природі і підтверджуються статистичними даними. Зокрема, багаторічні спостереження взаємодії рисі і зайця в Канаді відповідають цим законам.

Контрольні питання

1. Хто вперше звернув увагу на проблему зростання населення планети?
2. Чи є проблема зростання населення планети актуальною?
3. Умови зростання популяції, що закладені в моделі Мальтуса?
4. Як ще називається модель Мальтуса і чому?
5. Яке значення має коефіцієнт в модель Мальтуса ?
6. Як ввести обмеженість ареалу в моделі Мальтуса?
7. Що таке логістична модель?
8. Як в модель хижак–жертва ввести конкуренцію за ресурси у кожній популяції?
9. Як побудовані моделі можна використати для прогнозу стану

екосистеми?

10. Як фазовий портрет допомагає проаналізувати стан екосистеми?

2.3. Моделювання розвитку епідеміологічних процесів

2.3.1. Класична модель для опису поширення епідемії

Для того щоб органи охорони здоров'я могли вжити найефективніших заходів в боротьбі з епідемією, необхідно уміти кількісно оцінювати порівняльні достоїнства різних методів: введення карантину, вакцинації, виявлення контактів. Крім того, моніторинг і спеціальні розрахунки дозволяють визначити час настання епідемії, а також прогнозувати час піку епідемії і кількість хворих [1, 16]. Для збору і обробки даних про розповсюдження епідемії створено медичні автоматизовані системи, в яких моделювання епідемічних процесів відіграє найголовнішу роль.

Такі моделі аналогічні розглянутим в попередньому розділі моделям «хижак–жертва», але використовується дещо інша термінологія – «паразит–господар». Розглядаються дві популяції: однорідна популяція паразита, що викликає захворювання, і однорідна популяція господаря (людина). Епідемія – саморегульований процес взаємодії популяцій паразита і господаря. Розглянемо простий варіант такої взаємодії.

Д. Бернуллі був першим, хто застосував диференціальне числення для моделювання розвитку інфекційних захворювань в своїй статті про вакцинацію проти віспи (1760 р.). Класичною вважається модель Кермака – Макендріка, запропонована в 1927 р .

Припустимо, що захворювання передається за допомогою контактів хворих N_2 із здоровими N_1 і що середнє число нових випадків захворювання, які з'являються в інтервалі h_t, буде пропорційне як числу джерел інфекції, так і числу сприйнятливих (здорових) індивідуумів [16]. Якщо частота контактів між членами цієї групи дорівнює β, то середнє число випадків захворювань, що з'являються в інтервалі h_t становитиме

$$dN_1 = -\beta N_1 N_2 dt . \tag{2.41}$$

Звідси маємо перше рівняння системи

$$\frac{dN_1}{dt} = -\beta N_1 N_2 . \tag{2.42}$$

Швидкість зміни числа хворих дорівнює вищенаведеній швидкості захворювання $\beta N_1 N_2$ за вирахуванням швидкості одужання хворих, яка тим вища, чим більше число хворих, тобто

$$\frac{dN_2}{dt} = \beta N_1 N_2 - \gamma N_2 . \qquad (2.43)$$

Коефіцієнт γ – величина, обернена до середнього часу одужання. Швидкість зміни числа людей, що перехворіли, буде дорівнювати швидкості їх одужання із протилежним знаком

$$\frac{dN_3}{dt} = \gamma N_2 . \qquad (2.43)$$

Таким чином, динаміка розповсюдження епідемії при вказаних припущеннях описується наступною системою диференціальних рівнянь

$$\begin{aligned} \frac{dN_1}{dt} &= -\beta N_1 N_2; \\ \frac{dN_2}{dt} &= \beta N_1 N_2 - \gamma N_2; \\ \frac{dN_3}{dt} &= \gamma N_2. \end{aligned} \qquad (2.44)$$

Початкові умови:

$$N_1\big|_{t=0} = N_{10}, \quad N_2\big|_{t=0} = N_{20}, \quad N_3\big|_{t=0} = N_{30}. \qquad (2.45)$$

Рішення системи (2.45) чисельними методами представлене на рис 2.3 з початковими умовами

$$N_1\big|_{t=0} = 0, \quad N_2\big|_{t=0} = 0, \quad N_3\big|_{t=0} = 0. \qquad (2.46)$$

Видно, як спалахує і вибухає епідемія, але врешті-решт, кожен отримує імунітет і далі нічого не відбувається.

У реальній же ситуації ми спостерігаємо повторення спалахів епідемії, класична модель фактично описує один такий спалах, після деякого моменту часу графіки не зазнають змін. Необхідно доповнити модель, врахувати ще якісь чинники, для того щоб на графіках відображалися періодично повторювані процеси, які ми спостерігаємо насправді. Розглянемо моделі з урахуванням періодичної повторюваності епідемій.

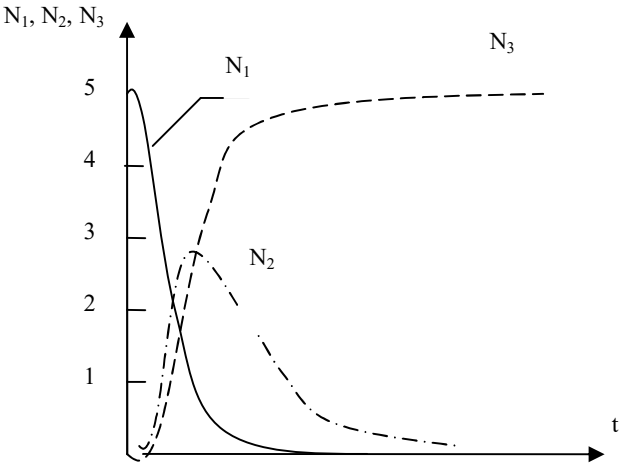


Рис. 2.3 – Розв'язок моделі Кермака-Макендріка

2.3.2. Врахування періодичної повторюваності в моделях епідеміології

Позначимо чисельність сприйнятливих (або здорових) і інфікованих (хворих) через S і I відповідно. Вважаючи, що розмір популяції господаря постійний (N=const), а всі новонароджені сприйнятливі і захворювання з довічним імунітетом, одержимо наступну систему рівнянь, що описують динаміку розповсюдження епідемії:

$$\begin{aligned} \frac{dI}{dt} &= \alpha SI - \beta I; \\ \frac{dS}{dt} &= \gamma(N - S) - \alpha SI. \end{aligned} \tag{2.48}$$

де α – коефіцієнт зараження сприйнятливих; β – коефіцієнт одужання інфікованих; γ – коефіцієнт смертності членів популяції. Член $\alpha \cdot S \cdot I$ визначає число заражених, пропорційне числу зустрічей інфікованих із сприйнятливими, тобто добутку їх чисельностей. Член $\beta \cdot I$ визначає одужання інфікованих, пропорційне їх числу. Член $\gamma(N\text{-}S)$ визначає число народжених сприйнятливих, що відповідає припущенню про постійний розмір популяції господаря.

У цій моделі ми вважали, що людина захворює відразу після контакту з хворим. Проте більшість захворювань має інкубаційний період, в який людина ще не заразлива. Таким чином, число тих, що заразилися, залежить не від зустрічей інфікованих із сприйнятливими, а від зустрічей сприйнятливих з членами популяції, які вже заразливі. Число цих членів

популяції приблизно дорівнює числу інфікованих у момент часу, що відстає на середню тривалість інкубаційного періоду τ від теперішнього часу, тобто t-τ. Перетворена система рівнянь (2.48) має такий вигляд:

$$\begin{aligned} \frac{dI}{dt} &= \alpha SI(t-\tau) - \beta I; \\ \frac{dS}{dt} &= \gamma(N-S) - \alpha SI(t-\tau). \end{aligned} \tag{2.49}$$

Система (2.49) є системою диференціальних рівнянь з аргументом, що запізнюється; у такому вигляді можна відстежувати на моделях повторюваність епідемій.

Повернемося до класичної моделі і внесемо в неї додаткову інформацію для отримання реальної картини повторюваності епідемій. Якщо припустити, що придбаний імунітет через певний час τ_1 втрачається і люди стають знов сприйнятливими до захворювання, ми одержимо періодичні спалахи хвороби. Крім того, як і в попередній моделі, введемо ще інкубаційний період τ_2. Одержимо наступну модель:

$$\begin{aligned} \frac{dN_1}{dt} &= -\alpha N_1 N_2(t-\tau_2) + N_2(t-\tau_1); \\ \frac{dN_2}{dt} &= \alpha N_1 N_2(t-\tau_2) - \gamma N_2; \\ \frac{dN_3}{dt} &= \gamma N_2 - N_2(t-\tau_1). \end{aligned} \tag{2.52}$$

Розв'язання системи (2.52) описує періодичні спалахи захворювання з урахуванням часу τ_1 втрати імунітету, а також інкубаційного періоду τ_2.

2.3.3. Урахування популяційного імунітету в моделях епідеміології. Моделі з урахуванням мінливості збудника

У епідеміології розроблена теорія про саморегуляцію паразитарних систем. У основі цієї теорії лежить уявлення про механізм самозгасання епідемії за рахунок зниження вірулентності збудника під дією імунітету популяції. Під вірулентністю збудника розуміють сукупність властивостей збудника, що визначає здатність його до зараження людини. Механізм самозгасання епідемії пояснює «епідеміологічний парадокс»: згасання епідемії до того, як буде вичерпаний весь прошарок сприйнятливих [17, 18].

На початковому етапі формування імунітету в організмі хворого популяція збудника зазнає зміни; до цього ця популяція складається переважно з вірулентних варіантів. Вірулентні варіанти гинуть (через умови, створювані в організмі унаслідок розвитку імунітету), і спочатку

відбувається направлений відбір маловірулентних варіантів, який йде в організмі імунних людей, а потім – стабілізуючий відбір в імунному колективі.

Тому в моделі необхідно ввести залежність вірулентності від числа людей, що перехворіли і набули імунітету (імунного прошарку). Чим вищий імунітет (більший імунний прошарок) популяції, тим нижче значення вірулентності збудника. Мета моделювання – простежити на епідеміологічній моделі дію вказаного механізму, з тим, щоб перевірити принципову можливість такого способу розв'язання парадокса.

В моделі припускаємо наявність двох механізмів передачі інфекції: механізму контактної передачі (від гостроінфікованних до сприйнятливих) і механізму зараження при зіткненні з осередком захворювань. Крім того, враховуємо групу латентно інфікованих, для того, щоб більш повно перевірити механізм самозгасання епідемії.

Нехай N_1, N_2, N_3, N_4 – відповідно чисельності сприйнятливих, латентно інфікованих, гостро інфікованих і тих, що набули (після перенесення гострої інфекції) тимчасового імунітету. Зміною загальної чисельності популяції (як це прийнято при аналізі епідеміологічних моделей) нехтуємо, вважаючи, що $N_1 + N_2 + N_3 + N_4 = \text{const}$. Перший механізм передачі інфекції грунтується на припущенні про те, що приріст числа латентно інфікованих (ΔN_2) пропорційний числу контактів сприйнятливих (N_1) з гостро інфікованими (N_3): $\Delta N_2 = \gamma \nu N_1 N_3$; другий механізм – на припущенні про лінійну залежність ΔN_2 від вірулентності збудника ν і чисельності сприйнятливих (N_1): $\Delta N_2 = \nu N_1$.

Одержимо таку систему рівнянь:

$$\begin{aligned}
&\frac{dN_1}{dt} = -\lambda\nu N_3 N_1 + bN_4 - \delta\nu N_1; \\
&\frac{dN_2}{dt} = \lambda\nu N_3 N_1 - \beta N_2 + \delta\nu N_1; \\
&\frac{dN_3}{dt} = \beta N_2 - \rho N_3; \\
&\frac{dN_4}{dt} = \rho N_3 - bN_4; \\
&N_1 + N_2 + N_3 + N_4 = 1.
\end{aligned} \tag{2.54}$$

де λ, β, ρ, b, δ – питомі швидкості переходу індивідів з однієї епідемічної групи в іншу. Оскільки нас цікавить можливість вступу епідемії у фазу згасання ще до завершення періоду нарощування імунного прошарку, то нехтуватимемо потоком з групи 4 в групу 1, поклавши $b = 0$ і відкинувши

рівняння для N_4:

$$\frac{dN_1}{dt} = -\nu(\lambda N_3 + \delta)N_1;$$
$$\frac{dN_2}{dt} = \nu(\lambda N_3 + \delta)N_1 - \beta N_2; \qquad (2.55)$$
$$\frac{dN_3}{dt} = \beta N_2 - \rho N_3.$$

Введемо механізм саморегуляції, вважаючи, що зростання імунного прошарку N_4 зменшує вірулентність збудника υ: $\upsilon = k - cN_4$. В цьому випадку, використовуючи співвідношення $N_4 = 1 - (N_1 + N_2 + N_3)$, одержимо

$$\frac{dN_1}{dt} = -(k - c(1 - N_1 - N_2 - N_3))(\lambda N_3 + \delta)N_1;$$
$$\frac{dN_2}{dt} = (k - c(1 - N_1 - N_2 - N_3))(\lambda N_3 + \delta)N_1 - \beta N_2; \qquad (2.56)$$
$$\frac{dN_3}{dt} = \beta N_2 - \rho N_3.$$

Величина $\nu = k - cN_4$ зменшується від початкового значення $\nu_0 = k > 0$, наближуючись до $\nu = 0$, що відповідає поставленим вимогам про залежність вірулентності від імунітету популяції. Дослідження показують, що система (2.56), яка враховує зниження вірулентності під дією імунного процесу, має стійкий стан рівноваги, а це свідчить про можливість самозгасання епідемічного процесу саме за рахунок вказаного механізму, оскільки при виключенні його (модель (2.54) такої можливості немає. Зростання епідемії в моделі (2.54) йде до повного вичерпання імунного прошарку ($N_1 \to 0$), тоді як в моделі (2.56) $N_1 > 0$. Таким чином, принципова можливість існування вказаного механізму простежується на основі зіставлення характерів поведінки моделей (2.54) і (2.56).

Зв'язуючи вірулентність ν з імунітетом популяції (у моделі (2.56) він пропорційний $N4$, перше рівняння системи), розглядатимемо величину k як поріг вірулентності, після якого збудник викликає епідемічний процес. Величина c характеризує темп наростання імунітету популяції. Недоліком залежності $\nu = k - cN_4$ є неможливість відобразити перехід збудника у фазу становлення епідемічного варіанта. Імунітет популяції (ослаблення якого і визначає початок переходу), складніше пов'язаний з N_1 і N_4, ніж прийнято в моделі (2.56). Залежність $\nu = k - cN_4$ є простою аналітичною конструкцією механізму саморегуляції ν.

Розглянуті приклади показують можливість побудови епідеміологічної

моделі, адекватно тієї, що відображає спостережувані залежності, що інтерпретуються як наявність саморегуляції в паразитарній системі.

У моделі (2.56) розвиток епідемії залежить від імунітету популяції. Для цього в рівняння, що описує динаміку сприйнятливих, вірулентність записується як функція від числа людей, що набули імунітету. Можна піти іншим шляхом: ввести ще одне диференціальне рівняння для опису динаміки самої вірулентності. Розглянемо таку модель. Всі змінні мають той же сенс, що і в моделі (2.56). Рівняння моделі мають вигляд

$$\begin{aligned}
\frac{dN_1}{dt} &= -\lambda\nu' N_3 N_1 + bN_4 - \delta\nu' N_1; \\
\frac{dN_2}{dt} &= \lambda\nu' N_3 N_1 - \beta N_2 + \delta\nu' N_1; \\
\frac{dN_3}{dt} &= \beta N_2 - \rho N_3; \\
\frac{dN_4}{dt} &= \rho N_3 - bN_4; \\
\frac{d\nu}{dt} &= (cN_2 - mN_4)\nu.
\end{aligned} \tag{2.57}$$

де ν – вірулентність епідемічного варіанта збудника; $\bar{\nu}$ – мінімальне її значення на початку періоду розповсюдження; $\nu' = \nu - \bar{\nu}$.

Останнє рівняння моделі (2.57) описує динаміку мінливості збудника, що визначається двома способами: залежністю вірулентності від чисельності латентно інфікованих N_2 і від величини колективного імунітету N_4. Графіки розв'язків системи (2.57) наведено на рис. 2.5.

Така модель дозволяє досліджувати динаміку рівня захворюваності N_3, частки носіїв імунітету N_4 і вірулентності збудника ν в ході епідемії при різних співвідношеннях c/m (коефіцієнти в останньому рівнянні моделі 2.57). Результати чисельного експерименту показують, що фазові перетворення в популяції збудника випереджають видимі епідеміологічні прояви.

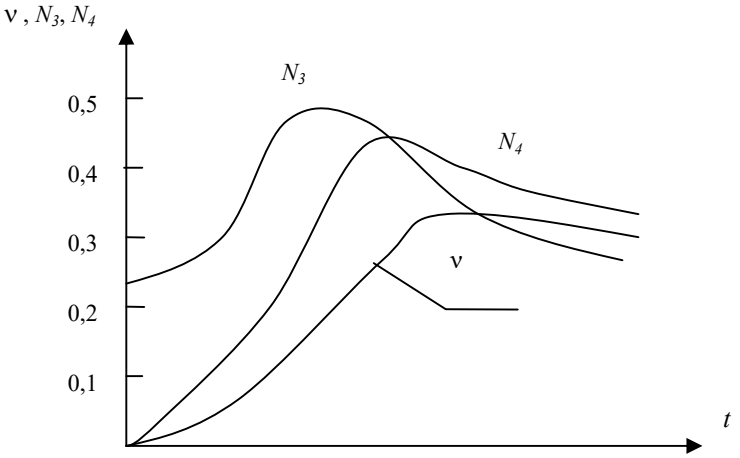


Рис. 2.5 - Динаміка змін рівня захворюваності, носіїв імунітету і вірулентності

Контрольні питання

1. Що можна простежити на моделях розповсюдження епідемії?
2. Опишіть модель Кермака – Макендріка.
3. Які недоліки моделі Кермака – Макендріка вам відомі?
4. Які засоби усунення недоліків моделі Кермака – Макендріка можна вказати?
5. Що таке вірулентність збудника ?
6. Які засоби введення вірулентності збудника в модель ви знаєте?
7. Як описується самозгасання епідемії в моделях?
8. Що таке популяційний імунітет і як можна врахувати його в моделях?

2.4. Моделювання в обчислювальній біології

2.4.1. Завдання пошуку й проблема дешифрування ДНК

Сьогодні активно розвиваються методи математичного моделювання для задач дешифрування ДНК і дослідження структур біологічних макромолекул. Такі моделі потребують використання алгоритмів пошуку підрядка в рядку. Особливість ситуації, що складається зараз в цій області, полягає у тому, що обсяг інформації, який мають в своєму розпорядженні дослідники, набагато більший від того, що можна осмислити, проаналізувати і використати в експериментальній роботі. Крім того, сама інформація

настільки специфічна, що концентрує в собі практично всі проблеми аналізу даних. Це висока розмірність досліджуваних структур, пропуски в даних, відсутність строгих прив'язок в аналізованих послідовностях чисел і символів і таке інше. Тому такою високою є актуальність розвитку нових математичних методів, обчислювальної техніки, програмного забезпечення, вдосконалення способів опису і зберігання геномної інформації. Цими проблемами активно займається біоінформатика. Задачі ці настільки серйозні, що народився новий напрям: комп'ютерна біологія [16].

Основними цільовими обчислювальними задачами комп'ютерної біології на даний момент деякі дослідники називають такі:

1. Розпізнавання білок-ділянок, що кодують в первинній структурі біополімери. Порівняльний аналіз первинних структур біополімерів.
2. Розшифрування просторової структури біополімерів і їх комплексів (рентгеноструктурний аналіз, методи ЯМР).
3. Просторове згортання білків (фолдінг).
4. Моделювання структури і динаміки біомакромолекул.
5. Створення і супровід спеціалізованих баз даних (баз білкових структур, нуклеотіиних послідовностей, шляхів метаболізму, клітинних ансамблів та ін.).

У 1988 р. один з першовідкривачів знаменитої подвійної спіралі ДНК, нобелівський лауреат Дж. Уотсон, публічно висловив думку про те, що наука впритул наблизилася до розкриття хімічної основи спадковості, причому не якого-небудь нижчого організму, а "царя природи" – людини. На той час вже було відомо, що спадковий апарат людини, геном, тобто сукупність всіх генів і міжгенних ділянок ДНК, складає близько 3 млрд. нуклеотидних пар. Ця величина здавалася неозоро великою, і сама думка, що такий обсяг інформації може бути одержаний, здавалася абсолютно фантастичною.

Гени – це основні фізичні елементи спадковості. Вони визначають всі ознаки організму, такі як стать, колір волосся і шкіри, статура, а також сприйнятливість до деяких хвороб. Гени є ділянками ДНК розміром від декількох десятків до декількох тисяч основ, розташованих в певній послідовності. Ця послідовність приводить до вироблення певного білка, який і додає даній людині його особливість. Навіть проста бактерія має близько 4000 генів, а людина – десятки тисяч. ДНК зберігається в хромосомах – у людини їх 23 пари. Кожна з хромосом є однією довгою подвійною спіраллю ДНК.

Структуру ДНК відкрили два молодих учених – американський біохімік Д. Уотсон і англійський фізик, що займався молекулярною біологією, Ф. Крик. Те, що ДНК несе генетичну інформацію, було встановлене раніше. Таємницею залишалася хімічна структура ДНК. Уотсон і Крик встановили, що ДНК складається з двох зчеплених ниток нуклеотидів, згорнутих в

спіраль. Нитки сполучені між собою водневими зв'язками пар органічних основ. Кожен нуклеотид містить одну з чотирьох основ: тимін (Т), аденін (А), цитозін (С) і гуанін (G).

Визначення положення пар основ в ланцюжку ДНК називається секвенуванням генома. Ця робота проводиться за допомогою сучасних хімічних технологій. Інтенсивність секвенувапння особливо зросла після 1998 р., коли почалася робота над проектом «Геном Людини», в якій взяли участь 1100 учених зі всього світу. Крім генома людини в даний час повністю секвеновані геноми ще 600 видів організмів, в тому числі бактерій (середній розмір генома 0,5 Мбайт), дріжджів (12 Мбайт), аскариди (97 Мбайт) і дрозофіли (6,45 Мбайт). Близький до завершення сиквенс генома лабораторної миші (близько 3000 Мбайт).

Серед проблем аналізу первинної структури одне з перших місць посідає питання розпізнавання серед послідовностей нуклеотидів в молекулі ДНК коуючих білок–ділянок (генів).

На рис. 2.6 показана нуклеотидна послідовність людського s–гемоглобіну. Основна задача полягає у виділенні в цій послідовності ділянок, що є генами. На наведеному малюнку така ділянка (екзон) виділена жирним шрифтом. Курсивом показана некодуюча область (інтрон).

atggtgcattttactgctgaggagaaggctgccgtcactagcctgtggagcaagatg
aatgtggaagaggctggaggtgaagccttgggcagg*taagcattggttctcaat*
gcatgggaatgaagggtgaatgttaccctagcaagttgattgggaaagtcctc
*aagatttttgcatctctaattttgtatctgatatggtgtcatttcatag*actcctcgtt
gtttacccctggaccagagatttttgacagctttggaaacctgtcgtctccctctgc

Рис. 2.6 - Розпізнавання білок-кодуючих областей в геномах

Найефективнішим методом визначення біологічної функції гена є пошук однакових послідовностей в базах даних нуклеотидних послідовностей ДНК. Існує дискусійна думка, що розпаралелювання обчислень і використовування суперкомп'ютерів для розв'язання подібного роду задач дозволить не тільки в сотні раз підвищити швидкість розшифрування первинних структур, але і зробити нові відкриття.

Дослідження ведуться також з ряду інших проблем. До них належить задача швидкого пошуку схожих послідовностей в банках молекулярно-генетичної інформації. Це, як відомо, традиційна задача інформатики, яка набула нового гострого звучання у зв'язку із специфікою генетичної інформації і її організацією в розподілених базах даних. Великий інтерес як в науковій, так і в популярній пресі викликають задачі ідентифікація генів, пов'язаних з генетичними хворобами.

Такі задачі ставлять високі вимоги до швидкодії і обсягу пам'яті використовуваних обчислювальних засобів, ще більш зростаючі у зв'язку із завершенням розшифрування геномів ряду організмів, кожний з яких містить

сотні мільйонів нуклеотидів. Час і обсяг пам'яті, яких потребують різні алгоритми аналізу первинних структур біополімерів, як правило, зростають як квадрат або куб довжини досліджуваної первинної структури. У ряді випадків, наприклад в задачі множинного порівняння, зростання складності обчислень з довжиною послідовності ще швидше. Вважається, що перехід до точніших методів і аналізу великих обсягів даних вимагає доступу до обчислювальних ресурсів, які можуть бути забезпечені тільки суперкомп'ютерами.

Так, наприклад, для послідовності, що містить 105 пар основ і 104 структури (кожна завдовжки 103 амінокислотних залишків), при квадратичній залежності швидкості обчислень від довжини первинної структури, необхідно виконати 1015 операцій. Для розв'язання таких задач потрібні суперкомп'ютери продуктивністю в сотні терафлоп. Разом з тим, це не знімає питання про необхідність пошуку і розробки принципово нових ефективних математичних підходів до аналізу структур даних.

2.4.2. Пошук підрядка в рядку за лінійний час

Основною задачею є точний збіг заданого підрядка з рядком, в якому цей підрядок розшукується.

Сформулюємо задачу точного збігу.

Хай заданий рядок *P*, який ми будемо називати зразком або паттерном (pattern), і довший рядок *T*, який ми будемо називати текстом (text). Задача про точний збіг полягає у відшуканні всіх входжень зразка *P* в текст *T*.

Наприклад, якщо *P* = *aba* і T = *bbabaxababay*, то *P* входить в *T*, починаючи з позицій 3, 7 і 9. Помітимо, що два входження *P* можуть перекриватися, як це видно по входженнях *P* в позиціях 7 і 9.

Практичне значення задачі про точні збіги є очевидним кожному, хто використовує комп'ютер, Ця задача виникає в широкому спектрі застосувань. Деякі з найзагальніших застосувань зустрічаються: у текстових редакторах; у таких утилітах, як grep в UNIX; у інформаційно-пошукових текстових системах, таких, як Medline, Lexis і Nexis; у пошукових системах бібліотечних каталогів, які в більшості великих бібліотек замінили звичні карткові каталоги; у інтернетівських браузерах, які просівають величезні кількості текстів у пошуках матеріалів, що містять дані ключові слова; у програмах, які читають інтернетівські новини і можуть відшукувати статті на необхідну тему; у гігантських цифрових бібліотеках, які вже заплановані на найближче майбутнє; у електронних журналах, які вже "публікуються" інтерактивно; у сфері обслуговування телефонних довідників; у інтерактивних енциклопедіях і інших засобах навчання на компакт-дисках; у інтерактивних словниках і тезаурусах, і в різних спеціалізованих базах даних. Як ми вже зазначили, в молекулярній біології є декілька сотень

спеціалізованих баз даних, що містять початкові рядки ДНК, РНК і амінокислот, а також оброблені зразки, одержані з початкових рядків.

Наведемо основні рядкові визначення, що знадобляться нам в подальшому.

Визначення. Рядок S – це впорядкований список символів, записаних підряд зліва направо. Для будь-якого рядка S позначимо через S [i .. j] (суцільний) підрядок S, який починається у позиції i і закінчується у позиції j рядка S. Зокрема, S [1 .. i] називається префіксом рядка S, що закінчується у позиції i, а S [i .. lS] – суфіксом рядка S, що починається у позиції i. Усюди lS позначає число символів в рядку S.

Визначення. Рядок S [i .. j] порожній, якщо $i > j$.

Наприклад, *california* – це рядок, *lifo* – це підрядок, *cal* – це префікс, а *ornia* – це суфікс.

Визначення. Власні префікс, суфікс і підрядок S – це відповідно префікс, суфікс і підрядок S, що не збігаються з S і що є не порожніми.

Звичайно ми використовуватимемо символ S для позначення довільного фіксованого рядка, якому не приписуються додаткові властивості або ролі. Проте якщо відомо, що рядок виконує роль зразка або тексту, то рядок відповідно позначатиметься через P або T.

Перший крок у розв'язанні цієї задачі є таким. Лівий кінець зразка P ставиться впритул до лівого ж кінця тексту T, і всі символи P порівнюються з відповідними символами T зліва направо, допоки або не виявиться незбіг символів, або не вичерпається P. В будь-якому випадку далі зразок P пересувається на один символ вправо і порівняння повторюються. Процес продовжується доти, поки правий кінець зразка P не зайде за правий кінець тексту.

Якщо позначити через n довжину зразка P, а через m довжину тексту T, то в найгіршому випадку число порівнянь в цьому методі матиме порядок $O(nm)$. Цей метод є простим у програмуванні, але його оцінка $O(nm)$ є незадовільною. Бажано було б знак множення замінити на знак складання $O(n + m)$, тобто прискорити пошуку підрядка до лінійного часу. Наприклад, якщо n=1000 і m=10000000, що є можливим в деяких задачах, то це дає суттєвий виграш у часі.

Розглянемо, як цього можна досягти. У багатьох алгоритмів порівняння і аналізу рядків ефективність може зрости, якщо пропускати порівняння. Ці пропуски виходять завдяки вивченню внутрішньої структури або зразка P, або тексту T. При цьому інший рядок може навіть залишатися невідомим алгоритму. Ця частина алгоритму називається препроцесною фазою. За нею слідує фаза пошуку, на якій інформація, одержана в препроцесній фазі, використовується для скорочення роботи з пошуку входжень P в T.

Основний препроцесинг буде описаний для довільного рядка, що

позначається через *S*. У конкретних застосуваннях роль *S* часто грає зразок *P,* але тут ми використовуємо *S* замість *P,* оскільки основний препроцесинг може застосовуватися і в інших ситуаціях.

Наступне визначення вводить основні величини, обчислювані в ході основного препроцесинга рядка.

Визначення. Для даного рядка *S* і позиції $i > 1$ визначимо $Z_i(S)$ як довжину найбільшого підрядка *S,* який починається в *i* і збігається з префіксом *S.*

Іншими словами, $Z_i(S)$ — це довжина найбільшого префікса S[*i*..|*S*|], що збігається з префіксом *S.* Наприклад, якщо *S* = *aabcaabxaaz,* то

Z_5(S)= 3 *(aabc...aabx...),* Z_6(S)= 1 *(aa...ab...),* Z_7(S)=Z_8(*S*)=0, Z_9(5)= *2 (aab...aaz).* Коли аргумент *S* ясний з контексту, ми замість $Z_i(S)$ писатимемо Z_i.

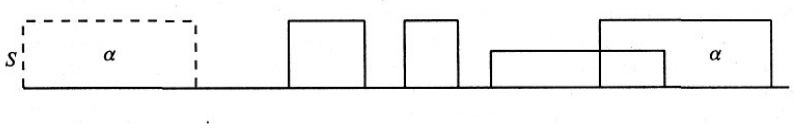


Рис. 2.7 - *Z*-блоки

Щоб ввести наступне поняття, розглянемо блоки, зображені на рис. 2.7. Кожен блок є підрядком *S* максимальної довжини, яка збігається з префіксом *S* і починається не з першої позиції. Блок починається в деякій позиції $j > 1$, такій, що $Z_j > 0$. Його довжина дорівнює Z_j. Такий блок називається Z-блоком.

Визначення. Для будь-якої позиції $i > 1$, в якій $Z_i > 0$, визначимо *Z*-блок в *i* як інтервал, що починається в *i* та закінчується у позиції $i + Z_i - 1$.

Визначення. Для будь-якого $i > 1$ хай r_i — крайній правий кінець *Z*-блоків, що починаються не пізніше за позицію *i.* По-іншому r_i *i* можна визначити як найбільше значення $j + Z_j - 1$ по всіх $1 < j \le i$, для яких $Z_j > 0$ (див. рис. 2.7).

Позначимо значення *j,* на якому досягається цей максимум, через l_i. Таким чином, l_i — це позиція лівого кінця *Z*-блока, що закінчується в r_i. У випадку якщо *Z*-блоків, що закінчуються в r_i більше за один, то як l_i береться лівий кінець будь-якого з них. Наприклад, якщо *S* = *aabaabcaxaabaabcy,* то $Z_{10} = 7$, $r_{15} = 16$ і $l_{15} = 10$.

Обчислення значень *Z* для рядка *S* за лінійний час і є основою препроцесингу, який ми використовуватимемо у всіх класичних алгоритмах пошуку за лінійний час, що вимагають обробки *P.* Але раніше покажемо, як же виконати його за лінійний час (тобто за час $O(|S|)$).

Пряме обчислення, засноване на визначенні, дає час $O(|S|^2)$. Препроцесний алгоритм обчислює Z_i, r_i і l_i послідовно для кожної позиції, починаючи з $i = 2$. Всі набуті значення *Z* зберігаються алгоритмом, але на кожній ітерації *i* використовуються тільки $r_i - 1$ і $l_i - 1$, а інші значення *r* і *l* не

потрібні. Отже, для зберігання останніх обчислених значень алгоритму достатньо мати прості змінні r і l. Таким чином, на кожній ітерації i, якщо наш алгоритм знаходить новий Z-блок (що починається в i), величина змінної r просто зростає до досягнення кінця цього Z-блока і тим зберігає свій статус найправішої позиції.

Спочатку алгоритм визначає значення Z_2, безпосередньо переглядаючи зліва направо і порівнюючи символи $S[2..|S|]$ і $S\ [1..|\ S\ |]$ до їх незбігу. Z_2 дорівнює довжині збіжної частини. Якщо $Z_2 > 0$, то значення $r = r_2$ вважається таким, що дорівнює $Z_2 + 1$, а $l = l_2$ – таким, що дорівнює рівним 2. Інакше обидві величини вважаються такими, що дорівнюють 0. Тепер зробимо індуктивне припущення, що алгоритм коректно обчислив Z, для i до k-1 > 1, і припустимо, що алгоритм знає поточні значення $r= r_{k-1}$ і $l = l_{k-1}$. Далі потрібно обчислити Z_k, $r = r_k$ і $l = l_k$.

Ідея в тому, щоб використовувати для прискорення розрахунку Z_k вже обчислені значення Z. В деяких випадках Z_k можна знайти з попередніх значень Z без додаткових порівнянь. Наприклад, припустимо, що $k = 121$, всі значення від Z_2 до Z_{120} вже обчислені, $r_{120} = 130$ і $l_{120} = 100$. Це означає, що підрядок довжини 31, що починається у позиції 100, збігається з префіксом S (довжини 31). Звідси виходить, що підрядок довжини 10, що починається у позиції 121, повинен збігатися з підрядком довжини 10, що починається у позиції 22, так що значення Z_{22} може бути дуже корисне при обчисленні Z_{121}. Так, якщо, скажімо, $Z_{22} = 3$, то просте міркування показує, що і $Z_{121} = 3$. Отже, значення Z_{121} може бути набуте без додаткових порівнянь взагалі.

Таким чином, описаний Z-алгоритм можна подати у такому вигляді.

При заданих Z, для всіх $1 < i \le k$ -1 і поточних значеннях r і l величина Z_k і зміни для r і l обчислюються так:

begin

1. Якщо $k > r$, то знайти Z_k безпосереднім порівнянням символів до незбігу підрядків, що починаються з позиції k і з позиції 1. Довжина збіжної частини і дає Z_k. Якщо $Z_k > 0$, то маємо $r = k + Z_k$ - 1 і $l = k$.

2. Якщо $k \le r$, то позиція k знаходиться в Z-блоці і, отже, $S[k]$ міститься в підрядку $S[l..r]$ (назвемо його α), такому що $l > 1$ і α збігається з префіксом S. Тому символ $S[k]$ стоїть і в позиції $k' = k - l + 1$. З тих же причин підрядок $S[k..r]$ (назвемо його β) повинен збігатися з підрядком $S[k'..|\beta|]$. Звідси виходить, що підрядок, який починається з позиції k, повинен збігатися щонайменше з префіксом S довжини $\mathrm{mm}\{Z_k, |\ \beta\ |\} = r - k + 1$.

Розглянемо два окремі випадки, залежно від того, на чому досягається цей мінімум.

а) якщо $Z_k < |\beta|$, то $Z_k = Z_k$, і r, l не змінюються.

b) якщо $Z_k \le |\beta|$, то весь підрядок $S[k..r]$ повинен бути префіксом S і $Z_k \le |\beta| = r - k+1$. Проте значення Z_k може бути більше, ніж β, тому потрібно

порівняти символи до незбігу, починаючи з позиції r+1, з символами, починаючи з позиції |β| +1. Хай незбіг відбувся на символі $q \le r$ +1. Тоді Z_k вважається таким, що дорівнює q - k, $r = q$ - 1 і $l = k$.

end.

Перш ніж обговорювати складніші (класичні) методи пошуку точного збігу, покажемо, що вже основний препроцесинг дає алгоритм з лінійним часом. Ось простий з алгоритмів, який ми знаємо.

Хай $S = P\$T$ – рядок, що складається із зразка P і тексту T, між якими стоїть знак $\$$, відсутній в обох рядках. Нагадаємо, що $|P| = n$, $|T| = m$ і $n \le m$. Тому $|S| = n + m + 1 = O(m)$. Обчислимо $Z_i(S)$ для i від 2 до $n + m + 1$. Оскільки \$ не зустрічається в P і T, то $Z_i < C$ для будь-якого $i > 1$. Будь-яке значення $S > n + 1$, для якого $Z_i(S) = n$, визначає входження P в T, що починається з позиції i-(n+1). І навпаки, якщо P входить в T з позиції j, то значення $Z_{(n+1)} + j$ повинне дорівнювати n. Оскільки всі значеня $Z_i(S)$ можна обчислити за час $O(n+m) = O(m)$, то такий алгоритм визначає всі входження P в T за час $O(m)$.

Оскільки функція Z обчислюється для зразка за лінійний час і може прямо використовуватися для розв'язання задачі точного збігу з часом $O(m)$, то навіщо шукати інші методи? Чому потрібно звертати увагу на складніші методи (Кнут–Морріс–Пратт, Бойер–мур, пошук у реальному часі, суфіксні дерева і т. ін.)?

Для задач точного збігу алгоритм Кнута–Морріса–Пратта дещо кращий за пряме використовування Z_i. До того ж він має історичне значення і був пристосований в алгоритмі Ахо–Корасика для пошуку набору зразків в тексті за час, лінійно залежний від довжини тексту. Цю проблему не можна розв'язати добре, використовуючи тільки значення Z_i. Узагальнення методу Кнута–Морріса–Пратта на пошук у реальному часі має перевагу в ситуаціях, коли текст послідовно вводиться (у режимі on-line) і потрібна упевненість, що алгоритм буде готовий до прийому кожного чергового символу. Метод Бойера–Мура важливий тому, що (при вдалій реалізації) його час роботи у гіршому разі також лінійним, але звичайно потрібен сублінійний час і перевіряється тільки частина символів з T. Отже, в більшості випадків цей метод переважає.

2.4.3. Використання суфіксних дерев у завданнях пошуку

Суфіксне дерево – це структура даних, яка виявляє внутрішню побудову рядка більш глибоко, ніж основний препроцесинг. Суфіксні дерева можна використовувати для розв'язання задачі про точні збіги за лінійний час (досягаючи тієї ж межі для якнайгіршого випадку, як в алгоритмі Кнута–Морріса–Пратта і Бойера–Мура), але їх справжня перевага виявляється при розв'язанні за лінійний час багатьох рядкових задач, більш складних, ніж точні збіги. Більш того, суфіксні дерева наводять міст між задачами точного

збігу і неточного збігу.

Класичним засосуванням для суфіксних дерев є задача про підрядок. Нехай заданий текст T довжиною m. За препроцесний час $O(m)$, тобто лінійний, потрібно приготуватися до того, щоб, одержавши невідомий рядок S довжиною n, за час $O(n)$ або знайти входження S в T, або визначити, що S в T не входить. Це означає, що дозволено використовувати препроцесинг з часом, пропорційним довжині тексту, але після цього пошук рядка S повинен виконуватися за час, пропорційний довжині S, незалежно від довжини T. Ці межі досягаються застосуванням суфіксного дерева. Воно будується для тексту за час $O(m)$ в препроцесній стадії, а потім, маючи суфіксне дерево, алгоритм, одержавши рядок довжиною n на вході, шукає його за час $O(n)$.

Для опису алгоритму побудови суфіксних дерев будемо використовувати рядок S довжиною m. Ми не використовуємо позначень P і T (зайнятих під зразок та текст), оскільки до суфіксних дерев звертаються в широкому діапазоні задач. де початковий рядок іноді виконує роль зразка, іноді – тексту, а іноді – ні того, ні іншого.

Визначення. Суфіксне дерево F для m-символьного рядка S – це орієнтоване дерево з коренем, що має m примірників листя, занумерованого від 1 до m. Кожна внутрішня вершина, відмінна від кореня, має не менше 2 дітей, а кожна дуга помічена непорожнім підрядком рядка S (дуговою міткою). Ніякі дві дуги, що виходять з тієї ж самої вершини, не можуть мати позначок, що починаються з того ж самого символу. Головна особливість суфіксного дерева полягає у тому, що для кожного листа i конкатенації міток дуг на шляху від кореня до листа i в точності складає (вимовляє) суфікс рядка S, який починається у позиції i. Тобто суфікс рядка є $S[i .. m]$.

Наприклад, суфіксне дерево для рядка *xabxac* (рис. 2.8) будується таким чином. Шлях із кореня до листа 1 вимовляє повний рядок S = *xabxac*. Шлях до листа 2 вимовляє підрядок (*abxac*), що починається з позиції 2 в повному рядку S. Відповідно шлях до листа 3 вимовляє підрядок (*bxac*), що починається з позиції 3 в повному рядку S, тоді як шлях до листа 5 вимовляє суфікс *ac*, який починається в позиції 5 рядка S. Дуговими мітками в даному випадку є u, w.

Визначення для S не гарантує, що таке дерево дійсно існує для будь-якого рядка S. Проблема в тому, що якщо один суфікс збігається з префіксом іншого суфікса, то побудувати суфіксне дерево за наведеним визначенням неможливо. Якщо припустити, що останній символ рядка S ніде більше не зустрічається, то суфікс не може бути префіксом водночас. Будемо додавати в кінець рядка S символ $.

Визначення. Мітка шляху від кореня до деякої вершини – це конкатенація підрядків, що позначають дуги шляху у порядку їх проходження. Шляхова мітка вершини – це мітка шляху від кореня F до цієї

вершини.

Визначення. Для будь-якої вершини *v* суфіксного дерева рядковою глибиною *v* називається число символів в шляховій мітці *v*.

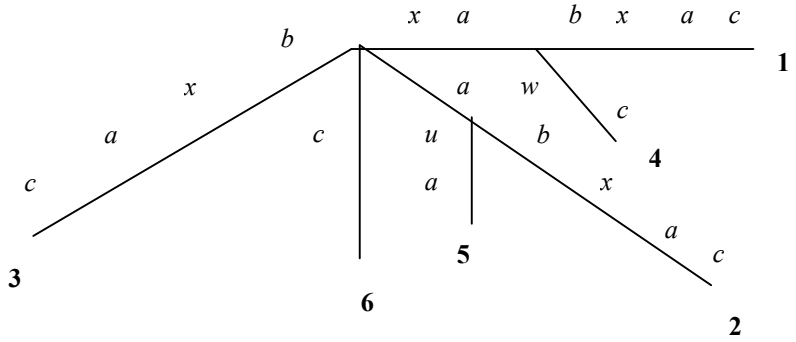


Рис. 2.8 - Суфіксне дерево для рядка *xabxac*

Визначення. Шлях, який закінчується у середині дуги (*u*, *v*), ділить мітку (*u*, *v*) в своїй точці призначення. Визначимо мітку цього шляху як шляхову мітку *u* з додаванням символів дуги (*u*, *v*) до точки призначення, що ділить дугу.

Перш ніж вдаватися в деталі методів побудови суфіксних дерев, подивимося, як суфіксне дерево для рядка використовується при розв'язанні задачі про точні збіги. Заданий зразок *P* довжиною *n* і текст *T* довжиною *m*, потрібно знайти всі входження *P* в *T* за час $O(n+m)$. Ми вже бачили розв'язання цієї задачі. Суфіксні дерева дають інший підхід.

Побудуємо суфіксне дерево *F* для тексту *T* за час $O(m)$. Потім шукатимемо збіги для символів з *P* уздовж єдиного шляху в *F* доти, поки або *P* не вичерпається або черговий збіг не стане неможливим. У другому випадку *P* в *T* не входить. У першому випадку кожен лист в піддереві, що йде з точки останнього збігу, має своїм номером початкову позицію *P* в *T*, а кожна початкова позиція *P* в *T* нумерує такий лист.

Ключем до розуміння першого випадку (коли всі символи з *P* збіглися з шляхом в *T*) є таке спостереження: *P* входить в *T*, починаючи з позиції *j*, в тому і лише в тому випадку, якщо *P* є префіксом $T[j \,..\, m]$. Але це відбувається тоді і тільки тоді, коли *P* позначає початкову частину шляху від кореня до листа *j*. Саме по ньому і проходить алгоритм порівняння.

Цей збіжний шлях єдиний, оскільки ніякі дві дуги, що виходять з вершини, не мають міток, що починаються з того ж самого символу. І оскільки ми припустили, що алфавіт скінчений, то робота в кожній вершині займає константний час, а отже, час на перевірку збігу *P* зі шляхом

пропорційний довжині *P*.

На рис. 2.9 показано фрагмент суфіксного дерева для рядка T =*awyawxawxz*. Зразок *P* = *aw* входить в *T* три рази, починаючи з позицій 1, 4 і 7. Початкові позиції входжень *aw* в *awyawxawxz* збігаються з номерами листів в піддереві вершини зі шляховою міткою *aw* Зразок *P* збігається зі шляхом вниз до вершини, вказаної стрілкою, і, як вважається, листя, які розташовані нижче цієї крапки мають номери 1, 4 і 7.

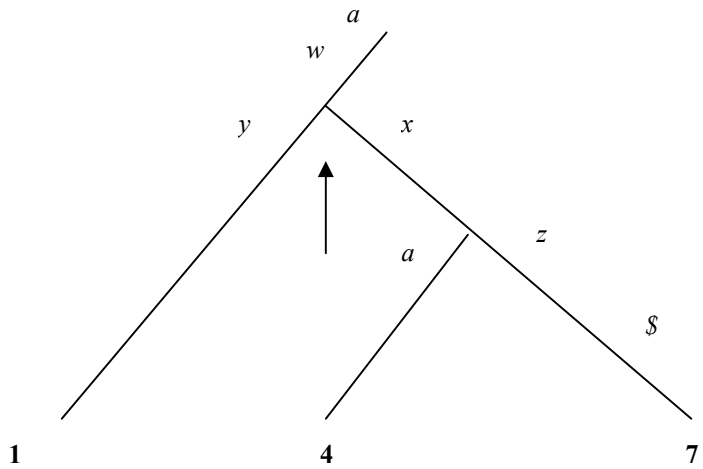


Рис. 2.9 - Три входження *aw* в *awyawxawxz*.

Зразок *P* повністю збігається з деяким шляхом в дереві, і алгоритм може знайти початкові позиції *P* в *T* проходженням по піддереву вниз від кінця збіжного щляху. При цьому перегляді потрібно просто зібрати всі номери листів, що зустрілися. Таким чином, всі входження *P* в *T* можуть бути знайдені за час $O(n+m)$. Такий же час досягається декількома алгоритмами, розглянутими раніше, але розподіл роботи між частинами алгоритму тут інший. Раніше запропоновані алгоритми витрачали час $O(n)$ на препроцесинг *P*, а потім час $O(m)$ – на пошук. Навпаки, при використовуванні суфіксного дерева час $O(m)$ витрачається на препроцесинг, а час $O(n+k)$ – на пошук, де *k* – число входжень *P* в *T*.

Метод Укконена працює швидко. Крім того, він потребує значно менше місця (тобто пам'яті), ніж інші методи. Для більшості задач, де потрібно будувати суфіксне дерево, краще вибрати метод Укконена. Алгоритм, який винайшов Еско Укконен для побудови суфіксного дерева за лінійний час, ймовірно, найпростіший з таких алгоритмів.

Алгоритм Укконена будує послідовність неявних суфіксних дерев,

останнє з яких перетворюється в справжнє суфіксне дерево рядка *S*.

Визначення. Неявне суфіксне дерево *S* – це дерево, що виходить з *S$* видаленням всіх входжень термінального символа *$* з міток дуг дерева, видаленням після цього дуг без міток і видаленням потім вершин, що мають менше двох дітей.

Неявне суфіксне дерево префікса *S* [1..*i*] рядка *S* аналогічно виходить з суфіксного дерева *S* [1..*i*]*$* видаленням символів *$*, дуг і вершин, як описано вище.

Визначення. Позначимо через F_i неявне суфіксне дерево рядка *S* [1..*i*], де *i* змінюється від 1 до *m*.

Алгоритм Укконена будує неявне суфіксне дерево F_i для кожного префікса *S*[1 .. *i*] рядка *S*, починаючи з F_1 і збільшуючи *i* на одиницю, поки не буде побудоване дерево F_m. Справжнє суфіксне дерево для *S* виходить з F_m і вся робота вимагає часу $O(m)$. Розглянемо алгоритм Укконена, представивши спочатку метод, за допомогою якого всі дерева F_i будуються за час $O(m^3)$, а потім оптимізуємо його так, що буде досягнута заявлена швидкість.

Алгоритм Укконена ділиться на *m* фаз. У фазі $i + 1$ дерево F_{i+1} будується з F_i. Кожна фаза $i + 1$ сама ділиться на $i + 1$ продовжень, по одному на кожний з $i + 1$ суфіксів *S*[1 .. *i*+1]. У продовженні *j* фази $i + 1$ алгоритм спочатку знаходить кінець шляху з кореня, поміченого підрядком *S*[*j* .. *i*]. Потім він продовжує цей підрядок, додаючи до його кінця символ *S*[*i*+1], якщо тільки *S*[*i*+1] ще існує. Отже, у фазі $i + 1$ спочатку розміщується в дерево рядок *S*[1 .. $i + 1$], за ним рядки *S*[2 .. $i + 1$], *S*[3 .. $i + 1$], ... (відповідно, в продовженнях 1, 2, 3, ..). Продовження $i + 1$ фази $i + 1$ продовжує порожній суфікс рядка *S*[1 .. *i*], тобто включає в дерево рядок з одного символу S[$i + 1$] (якщо тільки його там не було).

Дерево F_i складається з однієї дуги, поміченої символом *S*[*i*]. Формально алгоритм є таким:

```
Побудувати дерево F1.
for i from 1 to m - 1 do begin {фаза i + 1}
   for j from 1 to i + 1 begin {продовження j}
      знайти в поточному дереві кінець шляху з кореня з міткою S [j .. i].
      Якщо потрібно, продовжити шлях, додавши символ S[i + 1],
      забезпечивши появу рядка S [j.. i + 1] у дереві.
   end;
end;
```

Сформулюємо правила продовження суфіксів.

Щоб перетворити цей загальний опис на алгоритм, треба точно вказати, як виконувати продовження суфікса. Хай *S* [*j* .. *i*] = β – суфікс *S* [1 .. *i*]]. У продовження *j*, коли алгоритм знаходить кінець β в поточному дереві, він продовжує β, щоб забезпечити присутність суфікса β *S* [$i + 1$] в дереві.

Алгоритм діє по одному із наступних трьох правил.

Правило 1. У поточному дереві шлях β закінчується в листі. Це означає, що шлях від кореня з міткою β доходить до кінця деякої "листової" дуги (дуги, що входить в лист). При зміні дерева потрібно додати до кінця мітки цієї листової дуги символ $S[i+1]$.

Правило 2. Жоден шлях з кінця рядка β не починається символом $S[i+1]$, але принаймні один шлях, що починається звідти, є.

У цьому випадку повинна бути створена нова листова дуга, що починається в кінці β і що помічена символом $S[i+1]$. При цьому, якщо β закінчується усередині дуги, то повинна бути створена нова вершина. Лист в кінці нової листової дуги зіставляється з номером *j*.

Правило 3. Деякий шлях з кінця рядка β починається символом $S[i+1]$. У цьому випадку рядок β $S[i+1]$ вже є в поточному дереві, а тому нічого не треба робити.

Як приклад розглянемо (рис. 2.10) неявне суфіксне дерево для рядка *S = axabx*. Перші чотири суфікси закінчуються в листі, а суфікс *x,* що складається з одного символу, закінчується усередині дуги. Коли до рядка додається шостий символ (*b*), то перші чотири суфікси одержують продовження за правилом 1, п'ятий – за правилом 2, а шостий – за правилом 3. Результат показаний на рис. 2.10.

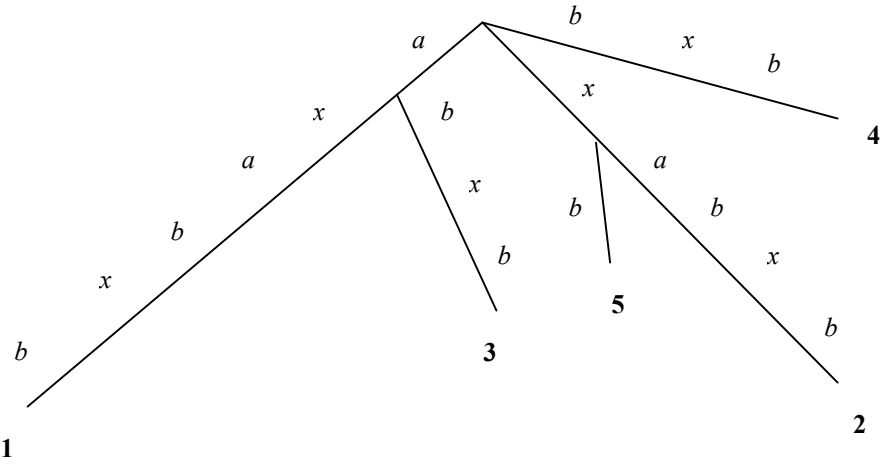


Рис. 2.10 - Розширене неявне суфіксне дерево після додавання символа *b*

Прискорення цього алгоритму розглянуто в роботі [16]. Там показано, що алгоритм Укконена будує справжнє суфіксне дерево для *S* і всі його суфіксні зв'язки за час $O(m)$.

Контрольні питання

1. У чому полягає задача дешифрування ДНК?
2. Як задача дешифрування ДНК формулюється в термінах обробки рядків?
3. Опишіть алгоритм пошуку підрядка в рядку з числом порівнянь порядку $O(nm)$.
4. Як зменшити в алгоритмі число порівнянь до порядку $O(n+m)$.?
5. Що таке префікс, корінь та суфікс рядка?
6. За рахунок чого зменшується число порівнянь в алгоритмі за лінійний час?
7. Опишіть алгоритм пошуку підрядка в рядку з числом порівнянь порядку $O(n+m)$.
8. Що таке препроцесинг ?
9. Що таке суфіксне дерево?
10. Опишіть алгоритм побудови суфіксного дерева.
11. За рахунок чого суфіксні дерева зменшують час пошуку підрядка?
12. У чому перевага методу Укконена – пошуку підрядка в рядку?

2.5. Особливості побудови моделей окремих органів і систем організму

2.5.1. Концептуальна модель систем організму

При істотних змінах параметрів організму в умовах патології виникає необхідність штучного управління потоками речовин і енергії. Це забезпечується за допомогою технічних систем управління, що працюють паралельно з природними фізіологічними механізмами або що беруть на себе повністю ту або іншу функцію організму. Такі технічні системи разом з об'єктом управління – організмом – утворюють складні інженерно-фізіологічні системи (ІФС) або біотехнічні системи (БТС), створення яких є комплексною проблемою і передбачає розробку методів моделювання поведінки організму в такій системі і всієї системи в цілому, формалізацію цілей і критеріїв якості функціонування системи, отримання ефективних управляючих алгоритмів [19].

Поява систем управління багатовимірними нелінійними процесами, що перебувають під впливом великого числа випадкових чинників, призводить до ускладнення використовуваних математичних моделей. Апарат диференціальних і різницевих рівнянь з випадковими коефіцієнтами, стохастичних диференціальних рівнянь, теорії масового обслуговування,

скінченних автоматів ймовірності дозволяє у ряді випадків будувати конструктивні моделі й реалізувати на їх основі ефективні системи управління.

Для моделювання локальних фізіологічних процесів, що перебігають в окремих органах, а також процесів досить загального характеру ряд дослідників застосовує апарат марковських ланцюгів.

При розробці біотехнічних систем використовують методи відновлення функцій організму, що звичайно передбачають різні медичні застосування, які допомажуть вилікуватись хворому. Тут перш за все слід згадати давні традиції, що мають методи відновлення кібернетичних функцій організму – його рецепторних і ефекторних систем. Що стосується рецепторних функцій організму, то тут треба звернути увагу, зокрема, на зорові і слухові протези (оптика, слухові апарати і т. ін.). Вважається, що конструкції таких приладів настільки прості, що при їх розробці математичні методи використовувати недоцільно. Ці методи зможуть знайти застосування лише в системах майбутнього, таких, як повний протез зорового каналу.

Інакше виглядає справа з ефекторними системами. Наприклад, не дивлячись на численні зусилля, до сьогодні не створено досконалих протезів втрачених або пошкоджених органів руху. Однією з причин є недостатнє розуміння механізмів управління цими органами, неможливість їх математичного опису і, як наслідок, інженерного відтворення. Метаболічні функції організму мають значно коротшу історію взаємодії з технічними засобами.

Нагадаємо, яке місце в процесах обміну речовини між організмом і середовищем займають системи організму, що традиційно виділяються. Є два джерела, що забезпечують організм речовинами з навколишнього середовища: шлунково-кишковий тракт (джерело субстратів і палива) і легені (джерело окислювача для забезпечення енерговитрат). Внутрішні транспортні засоби організму переносять ці речовини в біохімічну машину кліток. Універсальним транспортним засобом є система кровообігу (серце і кровоносні судини), яка доносить до кліток все необхідне – початкові субстрати, паливо і окислювачі.

Відведення продуктів обміну (відходів біохімічного виробництва) йде спочатку також через внутрішньоорганізмений транспорт – серцево-судинну і лімфатичну системи. Далі газоподібні речовини (CO_2) покидають організм через легені. Інші, важчі продукти обміну, хімічно перетворюються в системі печінки для – зручності подальшого їх виведення. Нирки виводять їх з організму і є головним його стоком.

Всі системи організму, які видаляють з кровоносного русла непотрібні і шкідливі речовини, можна розглядати як єдину систему очищення організму і евакуації обмінних шлаків. Всі транспортні процеси (отримання,

перетворення, запасання, використання речовин, що надходять, а також відведення відходів, що утворюються) регулюються нервовою і гуморальною системами, а також і залозами внутрішньої секреції. Наприклад, один з етапів процесу перетворення палива – запасання глюкози – регулюється підшлунковою залозою за допомогою гормону інсуліну, що виробляється нею.

В даний час інженерні засоби застосовуються для корекції і відновлення ледве не всіх найважливіших систем організму. Процеси доставки речовин дублюються системами штучного кровообігу і дихання (наприклад, штучним серцем, апаратами штучного кровообігу і т. ін.), а також різного роду допоміжними і управляючими механізмами, що стимулюють природну діяльність цих систем. Тут можна відзначити такі засоби дії на систему кровообігу, як пейсмекери, масажери, системи автоматичного дозування лікарських речовин і т. ін. Задача евакуації шлаків розв'язується різними системами очищення крові (системи плазмофорезу, гемосорбції, гемоперфузії, апарати «штучна нирка» і т. ін.).

У всіх випадках істотну роль при аналізі, конструюванні і оцінці ефективності технічних засобів відновлення організмених фізіологічних функцій відіграють методи математичного моделювання: Ці методи дозволяють інженерам зрозуміти механізми функціонування систем організму і описати їх на тій же мові, що і технічні системи, і тим самим забезпечити базу для ефективного сполучення фізіологічних і технічних елементів в єдиному інженерно-фізіологічному комплексі.

2.5.2. Рівні взаємодії з оточуючим середовищем

Перші математичні моделі ізольованих фізіологічних систем, що описують лише окремі процеси в організмі, були простими і ясними. Найраніша модель серцево-судинної системи, відома під назвою еластичного резервуару Франка (1899 р.) при моделюванні потребує розв'язання одного диференціального рівняння.

Якщо ця модель описувала лише процес зміни артеріального тиску у фазі діастоли, то сучасні моделі гемодинаміки з великою точністю відображають поведінку широкого кола змінних цієї системи у всіх фазах серцевого циклу. Модель де Патера і Ван дер Берга (1964 р.), наприклад, дозволяє досліджувати поведінку тиску і кровострумів серця і всіх крупних судин великого круга у систолу і діастолу.

Бажаючи удосконалити модель, поліпшити ступінь віддзеркалення нею тих або інших властивостей живого організму, дослідник все більш ускладнює її, відходячи від цілей, спочатку поставлених перед моделюванням. Тому необхідно виробити загальні позиції, подібні до принципів адекватного конструювання організму (Н. Рашевській) або

якнайменшої взаємодії (І. М. Гельфанд і М. Л. Цетлін, 1966) і особливо концепції гомеостазу.

Еволюційні закономірності організації живих систем обумовлені рівнем взаємодії з оточуючим середовищем. Організація живих систем пов'язана з поняттям вироблення поведінки і придбання досить стійких якостей, що дають можливість протистояти несприятливим діям навколишнього середовища.

Не торкаючись питань поведінки, пов'язаних з вищою нервовою діяльністю, зупинимося на еволюції деяких фізичних ознак, що сприяють виживанню і розвиваються паралельно з вдосконаленням пристосованої поведінки, зокрема здатності якнайкраще використовувати енергію речовин, що надходять в організм.

Для підтримки стійкості зв'язків системи, кінець кінцем, необхідне постійне поповнення енергії. Організація живих систем повинна забезпечувати високу ефективність використовування енергії зовнішнього середовища.

З цієї точки зору одноклітинні організми знаходяться в найнесприятливішому положенні. Це пов'язане, з одного боку, з недосконалістю механізмів, що перетворюють хімічну енергію живильних речовин в різні інші форми, а з іншого – з великими питомими енерговитратами Часті зміни фізико-хімічних умов навколишнього середовища складають комплекс чинників, що прагнуть порушити властивості клітини.

Одноклітинні системи, що вільно живуть, повинні володіти швидкою адаптацією до цих змін. У цьому полягає одна з причин постійної витрати енергії клітиною.

Інша причина інтенсивного обміну одноклітинного організму – надмірно велика поверхня по відношенню до його об'єму. Це ускладнює контроль над надходженням і виведенням активних речовин. Через ці дві причини тривалість існування одноклітинних надзвичайно коротка. Виживання виду досягається у них інтенсивним розмноженням.

Вищі форми життя мають певний енергетичний виграш. Об'єднані в єдиний організм клітини ізолюють себе від нерегульованого зовнішнього середовища проміжним, стабілізованим сектором, збільшується вага, відносно зменшується поверхня, і як наслідок витрата енергії таких багатоклітинних організмів, що потрібна для підтримки внутрішньоклітинної сталості, значно знижується. Таким чином, вища організація наділяє живі істоти перевагою у використовуванні енергії зовнішнього середовища.

З ускладненням організмів удосконалюються також клітинні механізми трансформації енергії живильних речовин. Якщо нижчі форми одержують

енергію за рахунок бродіння з дуже незначним виходом, то у вищих організмів з'являється ефективний спосіб отримання енергії – окислювальне фосфорилювання.

Чинником, що обумовлює розвиток корисних для організму структур і механізмів, є природний відбір, в процесі якого з'являється можливість вибору з безлічі різних біохімічних реакцій тих, що відповідають енергетичним вимогам і є передумовою економічності обміну речовин. Роль економічності в широкому значенні цього слова набуває важливого значення на всіх рівнях розвитку живих систем.

Так, клітинні механізми, що обумовлюють збереження сталості середовища, контролюють концентрацію метаболітів в клітині, не дивлячись на змінні умови її існування. Цей контроль зводиться до інгібування процесів обміну, що обмежує синтез кінцевих продуктів, запобігаючи їх виділенню в середовище і безцільній для клітини втраті.

У широкому діапазоні фізіологічного функціонування організму ці кислоти надходять до інших органів і тканин і складають додатковий енергетичний фонд.

На рівні окремих систем організму економічність часто набуває форми мінімізації енергії. Так, наприклад, при звичній ходьбі та ж сама швидкість у принципі може бути одержана при різній довжині і частоті кроків, проте несвідомо вибираються такі значення довжини кроку і числа кроків в хвилину, що енергія, яка витрачається при ходьбі, близька до мінімально можливої. Певне значення з погляду економічності біохімічних процесів має той факт, що в організмі людини синтезується лише близько половини з 20 амінокислот, що входять до складу його білків, а інші він одержує з їжею.

У процесі еволюції мінімізуються енерговитрати, ускладнюється їх структура, що у свою чергу, веде до зменшення контакту клітини із зовнішнім середовищем, створення проміжного, більш регульованого внутрішнього середовища, формування спеціалізованих функціональних систем.

У багатоклітинних організмів регуляція сприяє збереженню сталості як внутрішньоклітинного, так і міжклітинного середовища, і від якості і координованості систем, що забезпечують цю сталість, залежить спроможність вижити.

Таким чином відбувається формування системи газообміну, кровообігу, травлення, нейрогуморальної координації. «Закони будови організмів і органічного вдосконалення, – вважав Клод Бернар, – зливаються із законами клітинного життя. Органи додаються до органів і апарати – до систем власне для того, щоб забезпечити і міцніше регулювати клітинне життя. Обов'язок, покладений на них, полягає в тому, щоб з'єднати кількісно і якісно умови клітинного життя. Цей обов'язок неминучий; щоб поповнити його, вони

беруться за справу різним чином, розподіляють між собою роботу, яка є у великій кількості, коли організм складний, і у меншій кількості, коли він простий, але мета завжди та сама».

Так структурна організація забезпечує стійкість клітинних механізмів і виживання системи в цілому. Складність біологічної системи зростає в результаті її взаємодії з навколишнім середовищем, при цьому збільшення її елементів якісно і кількісно змінює її функціонування. З ускладненням системи неминуче відбувається зменшення надійності її функціонування, збільшується вірогідність відмови в одній з ланок єдиної системи, вірогідність відмови вища, чим складніша система. Тому з'являється потреба в механізмах, що забезпечують надійну роботу системи в цілому.

Організація функцій у вищих організмів частіше перебуває під впливом декількох управляючих систем, що і збільшує надійність їх функціонування. Так, наприклад, насосна функція серця знаходиться під контролем екстракардіальних регуляторних механізмів і інтракардіальної регуляції. У свою чергу, останні включають нервові, гуморальні і гідродинамічні компоненти регулювання. Множинність контролю, дублювання способів управлення можна проілюструвати на прикладах інших фізіологічних функцій (дихання, виділення, травлення і т. д.). Подібний принцип функціональної організації забезпечує великий запас надійності.

Таким чином, підвищення стійкості системи, її здатність до виживання є результатом багатовікового відбору. у процесі якого постійно змінювалася як структура, так і функція системи. У процесі еволюції механізми, що регулюють окремі процеси в системі, дублювалися і накладалися один на одного; тонші способи регуляції нашаровувалися па більш стародавні і грубі. Виниклі в результаті такого множинного розвитку організми володіють гнучкою і надійною структурою, кожен елемент якої робить свій внесок в багате розмаїття властивостей системи в цілому.

Діяльність всієї сукупності фізіологічних систем – дихання, кровообігу, терморегуляції, підтримки кислотно-лужної рівноваги – сприяє утворенню постійного внутрішнього середовища організму. Механізми і зв'язки на цьому рівні утворюють, так би мовити, фізіологічний "базис" організму, над яким підноситься його "надбудова", пов'язана з вищою нервовою діяльністю, поведінкою, його мотивацією.

Для кожного організму можна виокремити сукупність його змінних, що тісно пов'язані між собою і що мають близьке відношення до виживання організму, а тому значні зміни будь-якої з них рано чи пізно приводять до значних змін всіх інших. Ці змінні (число їх може бути великим) називають істотними. Істотні змінні можуть описувати локальні фізико-хімічні властивості внутрішнього середовища організму (PH, PO_2, температура, концентрація різних речовин і т. ін.) і фізичні характеристики організму в

цілому (ударний об'єм і частота скорочень серця, хвилинний об'єм дихання, периферичний опір судин і т. ін.).

Кажучи про сталість внутрішнього середовища організму, перш за все мають на увазі змінні, що описують локальні властивості внутрішнього середовища організму. Під сталістю середовища при цьому розуміють відносну сталість цих змінних, коли вони не виходять за так звані фізіологічні межі.

2.5.3. Часові й просторові границі роботи систем організму. Поняття гомеостазу

Кожна з фізіологічних систем «базису» має свої регулюючі ланцюги і механізми, які перешкоджають виникненню значних відхилень своїх істотних змінних. Якщо звернутися до розгляду залежності якої-небудь істотної змінної від параметрів, що описують умови зовнішнього середовища, то легко помітити, що майже завжди ця залежність є пологим «плато», що характеризує малі зміни змінної три відносно великих варіаціях умов зовнішнього середовища. Поступово «плато» переходить в обидві сторони в дві інші ділянки, що характеризуються відносно крутою залежністю змінної від параметрів зовнішнього середовища (рис. 2.11).

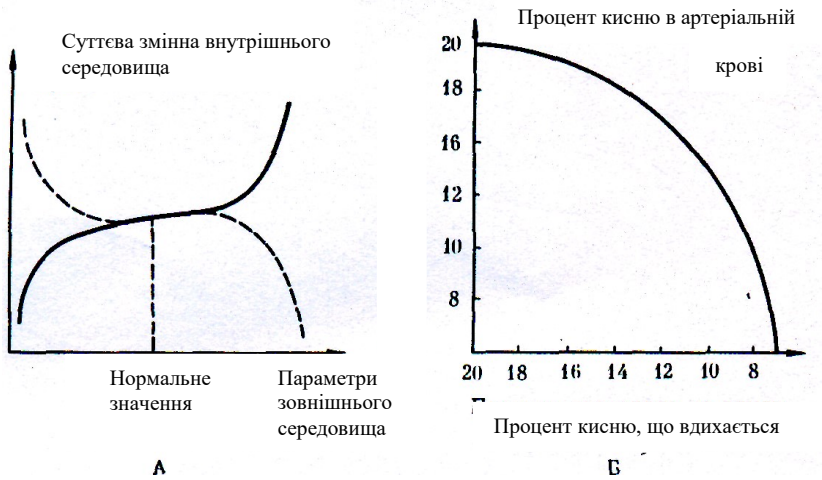


Рис. 2.11 - Залежність істотної змінної від зміни параметрів зовнішнього середовища. А – загальна закономірність; Б – приклад

В межах «плато» діяльність регулюючих ланцюгів виявляється достатньою для згладжування (демпфування) зовнішніх дій, проте коли останні стають дуже великими, гомеостатичні ресурси системи, що

регулюють дану змінну, виявляються недостатніми.

Усі процеси в організмі звичайно розвиваються так, що його змінні не виходять за межі, в яких існуючі зв'язки і механізми можуть ефективно регулювати рівень істотних змінних. Все багатство поведінки організму визначається тим, що він є гомеостатичною системою, що складається з одного комплексу зв'язаних між собою фізіологічних систем.

Якщо всі змінні якої-небудь системи організму не змінюються протягом достатньо довгого часу, то говорять, що ця система перебуває в рівноважному стані. Оскільки всі системи тісно пов'язані між собою, то весь організм може перебувати в рівновазі тільки тоді, коли в рівновазі перебувають всі системи, що його утворюють. При цьому рівноважні значення змінних кожної з систем визначаються спільно всіма системами організму.

Ясно, що будь-яка зміна параметрів або структури, що відбулася в якій-небудь системі організму, негайно розповсюджується по всіх системахм, викликаючи зміни в кожній з них. Якщо це збурення не було дуже великим, то рівновага в системі, що існувала до появи збурюючого чинника, знову відновиться, хоча і при якихось інших значеннях змінних. У цьому сенсі можна говорити про динамічну рівновагу фізіологічного комплексу організму і його істотних змінних. Оскільки при цьому рівноважні точки кожної із змінних, як правило, потрапляють на пологу ділянку залежності (рис. 2.11), то в цілому зсув рівноважного стану буде невеликим.

Проте можуть існувати і такі збурення, які різко змінюють положення рівноважної точки і навіть роблять систему в цілому нестійкою. Доти, поки зміни зовнішнього середовища або параметрів і структури організму такі, що система в цілому стійка, організм залишається життєздатним. Якщо при цьому точка рівноваги лежить в такій області, що всі істотні змінні не виходять за фізіологічні межі, то змінні організму і сам організм перебувають в умовах норми. Якщо система в цілому стійка, але зміни в ній такі, що рівноважна точка зміщується значно, так що частина істотних змінних виходить за фізіологічні межі, то настає патологічний стан організму. Якщо система втрачає стійкість положення рівноваги, організм стає нежиттєздатним. Багатство поведінки організму, можливі варіації його властивостей і процесів в ньому залежать від того, в якому стані він перебуває в даний момент часу.

Якщо організм перебуває в умовах норми, то при слабких збуреннях новий рівноважний стан досягається шляхом малих змін в його системах. При цьому зміни в кожній з систем майже не зачіпають процесів, що відбуваються в сусідніх системах. Можна сказати, що в цьому випадку (процеси в окремих системах перебігають "ізольовано" один від одного, а самі фізіологічні системи поводяться як ізольовані системи) організм

практично розпадається на сукупність окремих систем, незалежно від умов зовнішнього середовища, що реагують на зміни.

Якщо рівноважна точка організму розташована десь в області межі "плато", то будь-які зміни, що відбуваються при цьому в одній системі, передаються у всю решту систем, помітно впливаючи на процеси в них. Ні про яке представлення організму у вигляді сукупності ізольованих систем при цьому не може бути і мови, і в цьому випадку не може бути мови про те, щоб нехтувати впливом систем одна на одну. Особливий інтерес для цілей моделювання фізіологічних систем становить та обставина, що концепція гомеостазу дозволяє формалізувати і будувати на цій основі математичні моделі фізіологічних систем. Ми відзначали, що розглядатимемо гомеостатичні процеси протягом лише коротких відрізків часу. Діяльність організму – це одночасне функціонування всіх рівнів його організації, для кожного з яких характерні свої часові масштаби. Якщо навіть відкинути субклітинний (молекулярний) рівень і обмежитися розглядом макроскопічних структур організму, то умовно часовий характер діяльності організму можна виразити таблицею, наведеною на рис. 2.12.

Рис. 2.12 - Часові і просторові рамки діяльності організму

Грубо кажучи, часові показники лівої частини таблиці (0,3 секунди – 7 хвилин) відповідають сталим гомеостатичним механізмам організму; середні показники (10 – 30 днів) – процесам, пов'язаним з адаптивною поведінкою організму. Праві цифри (15 – 70 років) відповідають генетичним ефектам.

При аналізі складних систем, діяльність яких відбувається в декількох часових масштабах одночасно, дослідник, для спрощення задач, вважає, що характеристики процесів, що перебігають істотно повільніше, за досліджуваний їм час не можуть змінитися значною мірою. Тому впливом повільніших процесів при такому підході нехтують, вважаючи відповідні змінні параметри постійними. З іншого боку, процеси, що перебігають набагато швидше досліджуваних, вважаються тими, що відбуваються скільки завгодно швидко, практично миттєво. Вплив цих процесів на досліджувані явища враховується шляхом опису їх звичайними рівняннями алгебри.

2.5.4. Особливості відображень внутрішніх станів систем організму на системи діагнозів та діагностичних показників. Вибір структури діагностичних моделей

Як було зазначено раніше, організм людини – складна, ієрархічна, багаторівнева система, що самоорганізується та складається з взаємозалежних підсистем різного рівня підпорядкування, які взаємодіють з навколишнім середовищем при обміні речовинами, енергією й інформацією. Розглянемо формалізоване зображення внутрішнього середовища організму і його взаємодію з зовнішнім середовищем. Оптимальність функціонування кожної підсистеми організму визначається її кінцевим корисним ефектом, параметри якого сприймаються рецепторними підсистемами і передаються в центральну нервову систему (ЦНС). На підставі отриманої інформації, а також інформації про стан зовнішнього середовища, отриманої через органи почуттів, ЦНС визначає систему цільових функцій, відповідно до яких включаються різні механізми эфекторних зв'язків для зміни внутрішніх станів підсистем організму.

При цьому на кожному k-му рівні взаємодії внутрішнього і зовнішнього середовища, кожна i-та підсистема організму R_i (серцево-судинна, дихання, нервова, імунна й ін.) перебуває у визначеному j-му стані s_{ij}^{k} з множини

$$S_i = \{s_{ij}^{k}\};\ \ k = \overline{0,14};\ i = \overline{1,n_k};\ j = \overline{1,n_i},$$

де n_k – число підсистем організму на k-му рівні взаємодії з навколишнім середовищем; n_i – число можливих станів i-ї підсистеми на k-му рівні.

Множина станів S_i умовно розбивається на такі підмножини: S_{in} – норма; S_{ig} – прикордонний стан; S_{ip} – патологія. тобто $S_i = S_{in} \cup S_{ig} \cup S_{ip}$. Множина станів всього організму S визначається множиною станів усіх його підсистем:

$$S = \{S_i\} \;\; i = \overline{1, n_r} \,,$$

де $n_r = \max\limits_k n_k$.

Прийнята в медицині система діагнозів $D = \{D_i\}$ $i = \overline{1, n_d}$ є відображенням множини можливих станів організму на прийняту на даному етапі розвитку медицини систему термінів і визначень, передбачених прийнятими класифікаціями і номенклатурою захворювань

$$\{s_{ij}^k\} \to \{D_i\} \,.$$

Таким чином, кожен діагноз є деякою підмножиною станів організму $D_i \subset S$, причому кожен стан організму може відображати декілька рівнів взаємодії його підсистем. Діагноз D_0 – "практично здоровий" – визначається як

$$D_0 = \{s_{ij}^k\} \;\; \forall s_{ij}^k \in (S_{in} \cap S_{ig}) \,,$$

а множина інших діагнозів – як

$$\{D_i\}_{i \neq 0} = \{s_{ij}^k\} \;\; \exists s_{ij}^k \in S_{ip} \,.$$

З розвитком медицини множина діагнозів розширюється шляхом деталізації деяких діагнозів з урахуванням як більш низьких рівнів взаємодії (клітинний, біомолекулярний і нижчий), так і більш високих (рівень тонкого ефірного тіла і вищий). У такій постановці задача діагностики за допомогою комп'ютерної медичної діагностичної системи зводиться до задачі визначення належності поточного стану організму або окремої його підсистеми до одного з формалізованих станів з множини діагнозів $\{D_i\}$.

Вихідними даними для будь-якої комп'ютерної медичної діагностичної системи при діагностиці *i*-ї підсистеми організму є прийнята в медичній практиці система діагностичних ознак *X*:

$$X = \{x_0, \ldots, x_i, \ldots, x_m\},$$

яка якоюсь мірою відображає поточний *j*-й стан *i*-ї підсистеми організму на *k*-му рівні взаємодії s_{ij}^k , тобто множина станів $\{ s_{ij}^k \}$ відображається на множину ознак *X*.

Таким чином, невідомий поточний стан підсистем організму s_{ij}^k відображається як на систему діагнозів $\{D_i\}$, так і на множину ознак, при цьому задачею систем підтримки прийняття рішень в медицині (СППРМ) є визначення залежності $X \to \{D_j\}$. До даного формалізованого зображення підсистем організму слід додати два зауваження:

1) Організм людини постійно взаємодіє з факторами зовнішнього середовищ, і в результаті взаємодії внутрішнього і зовнішнього середовища протягом всього періоду життя (включаючи внутрішньоутробний період), з урахуванням генетичних програм розвитку, формується поточний стан усіх підсистем організму [5], для діагностики яких і розглядається формалізований підхід при обстеженні пацієнта в нормальних клінічних умовах. Однак деякі фактори зовнішнього середовища, такі. як стресові стани, прийом медикаментозних, наркотичних речовин та ін., не тільки формують стани підсистем організму протягом тривалого часу, але й у момент їхнього впливу приводять до стрибкоподібного переходу на інший енергетичний рівень, що характеризується якісно іншим набором станів і інших значень S_{in}, S_{ig} і S_{ip}.

Підхід, що розглядається, можна застосувати й у даних випадках, але при цьому необхідно розширити перелік діагнозів (діагнози в нормальних умовах і в момент впливу несприятливих факторів зовнішнього середовища), і мати у наявності відповідні навчальні вибірки. Відзначена особливість виникає при проектуванні СППРМ спеціального призначення (наприклад, моніторингові системи показників функціонального стану операторів складних людино-машинних комплексів), але деталізація даного підходу виходить за рамки даної роботи.

2) При конкретній реалізації СППРМ може діагностуватися не одна, а декілька патологій (і відповідно кілька прикордонних станів). У цьому випадку підсистема R_i, що діагностується, має внутрішні рівні ієрархії, і підмножини S_{ig} і S_{ip} розбиваються на більш дрібні підмножини за числом патологій (захворювань), що діагностуються.

Розглянемо особливості зв'язків стани – ознаки – діагнози.

У взаємозалежній багаторівневій системі результати будь-яких вимірів (дане твердження справедливе і для показників, отриманих при огляді лікаря або при суб'єктивній симптоматиці) можна вважати непрямими вимірами, тому що, з одного боку, одна ознака x_i (його зміна або прояв) відображає зміну станів багатьох підсистем організму $\{R_0, \ldots, R_i, \ldots, R_l\}$ (наприклад така інтегральна ознака, як температура тіла виявляється при багатьох патологіях):

$$\{ s^k_{0j}, \ldots, s^k_{ij}, \ldots, s^k_{lj} \} \rightarrow x_i ,$$

а з іншого боку, зміна стану однієї підсистеми реєструється зміною багатьох ознак:

$$s^k_{ij} \rightarrow \{x_0, \ldots, x_l, \ldots, x_k\},$$

тобто відношення між $\{ s^k_{ij} \}$ і X мають тип "багато до багатьох".

Різні види процедур формалізації системи діагностичних ознак, що включають стандартизацію їхнього опису, оцінку інформативності і вибір множини інформативних ознак, є недостатніми для одержання якісного комп'ютерного діагнозу, тому що зображення системи ознак у виді вектора не відображає складності відношень $\{ s_{ij}^k \}$ і X.

Відношення $\{ s_{ij}^k \} \rightarrow \{D_j\}$ варто розглядати як відношення "загального до часткового", тому що застосовувана в медичній діагностиці система класифікації діагнозів, яка використовується в медичній діагностиці, фіксує лише стаціонарні області станів підсистем організму, не розкриваючи повною мірою причинно-наслідкових зв'язків, динаміки розвитку і складності взаємодії багаторівневої ієрархічної структури підсистем організму S_R.

З теорії ідентифікації відомо [1, 5], що найкращі результати ідентифікації будуть одержані у тому випадку, коли структура моделі максимальним чином відображатиме структуру об’єкта. Стосовно розглянутої задачі, об’єктом є складна ієрархічна структура підсистем організму S_R, що характеризується множиною станів $\{ s_{ij}^k \}$ (структура зазначеної системи, як правило, нам невідома), а моделлю є залежність системи діагнозів $\{D_j\}$ від системи діагностичних ознак X ($D_j = f(X)$).

Слід зазначити, що деякий формалізований стан діагнозу може відповідати порушенню гомеостазу на декількох рівнях взаємодії, і навіть достовірний діагноз не завжди відображає причину такого порушення (діагноз ставиться на рівні патології органів і систем, а причиною є більш низькі рівні: біомолекулярний, клітинний та ін.). Аналогічним чином формується діагностичний простір ознак, компоненти якого залежать від методики виміру і медичної апаратури, яка використовується, і відповідають різним рівням взаємодії. Крім того, при постановці діагнозу, враховується генетична схильність пацієнта до деяких типів патологій.

Крім вище відзначених особливостей взаємодії системи ознак і діагнозів, передумовою до такої постановки задачі є той факт, що в медичній практиці, на інтуїтивному рівні, широко використовується багаторівнева система діагностичних ознак (симптоми, синдроми, симптомокомплекси), а постановка діагнозу є складним багатоетапним процесом прийняття рішення (від попереднього діагнозу до розгорнутого клінічного діагнозу). При цьому на різних етапах постановки діагнозу використовується система діагностичних ознак різного ступеня деталізації, які одержані у результаті реалізації заздалегідь обраної діагностичної стратегії, тобто цільового добору діагностичних ознак, проведення й аналізу діагностичних процедур.

Сам процес постановки діагнозу є частиною більш загального процесу надання медичної допомоги, який у формалізованому вигляді зображується як цикл, у процесі виконання якого лікар (за допомогою СППРМ) на підставі

аналізу клінічних оцінок (діагностичних ознак X) стану пацієнта s_{ij}^k формує діагноз D_j і приймає рішення про адекватне лікування. Залежно від початкового статусу пацієнта деякі фази циклу можуть бути пропущені або повторені. Так, наприклад, при первинному обстеженні після збору анамнезу лікар може зробити лише попередній висновок про стан пацієнта – первинний діагноз D_j^n.

На даному етапі обстеження лікар має неповний обсяг інформації (набір діагностичних ознак $X^n=\{x^n\}$) про стан пацієнта s_{ij}^k. На підставі набору ознак X^n можна лише визначити належність поточного стану s_{ij}^k до норми ($s_{ij}^k \in S_{in}$), або виявити відхилення від нормального стану. Таким чином, попередній діагноз D_j^n носить можливий (оціночний) характер без належної деталізації.

Розглядаючи з позиції теорії розпізнавання образів уточнені діагнози D_j^y точками в багатовимірному просторі ознак, можна стверджувати, що попередні діагнози D_j^n є кластерами, що поєднують групи уточнених діагнозів $D_j^n = \{D_o^y, \ldots, D_m^y\}$. При виявленні відхилень від норми ($s_{ij}^k \in S_{ig} \cup S_{ip}$) приймається рішення про виконання необхідних діагностичних процедур – або для підтвердження, або для уточнення, або для спростування початкового діагнозу D_j^n.

Критерієм доцільності або корисності проведення того або іншого діагностичного обстеження з обраних раніше груп – є припущення про те, наскільки може змінитися початковий діагноз, якщо будуть відомі результати даного обстеження конкретного пацієнта.

Результатом діагностичних процедур (які проводяться лікарями-фахівцями з використанням СППРМ) є додаткова безліч діагностичних ознак $\{x_i^y\}$, на підставі яких визначається уточнений діагноз пацієнта D_j^y, відповідно до якого проводяться лікувальні процедури $\{V_i\}$. У випадку спростування первинного діагнозу лікар переходить до альтернативного діагнозу і знову проводить відзначені вище етапи формування нового діагнозу.

Якщо на момент звертання за медичною допомогою пацієнт уже має діагноз (наприклад, у випадках загострення хронічних захворювань, при спостереженні за ходом початого раніше лікування), то діагностичні процедури носять характер уточнюючих, допоміжних засобів ведення хворого, виконуються для оцінки ефективності процесу лікування і, при необхідності, для зміни стратегії лікування.

При безперервному моніторингу життєвих показників (у палатах інтенсивної терапії, реанімації, операційних та ін.) можливе створення замкнутих контурів керування забезпеченням життєдіяльності – сигнальні

системи реального часу.

У медицині існує класифікація груп захворювань (і відповідна до неї спеціалізація лікарів-фахівців) за такими наступними напрямками: серцево-судинні, шлунково-кишкові, легеневі, шкірні, інфекційні, очні, хвороби вухо-горла-носа, жіночі, урологічні, психічні, стоматологічні і т. д. Розпізнавання групи захворювань на даному рівні виконується на основі суб'єктивної симптоматики (найчастіше самим пацієнтом без консультації з лікарем) або на основі опитування й огляду сімейним лікарем (лікарем-терапевтом). При цьому використовується невелика кількість так званих сигнальних діагностичних ознак (в основному дихотомічних) при мінімальному використанні вимірів (температура, тиск). Важливою сигнальною ознакою є біль і відчуття дискомфорту, топологія і характер яких дозволяють діагностувати групу захворювань [5]. Таким чином, у традиційній медицині перший рівень діагностики груп захворювань виконується на рівні органів і систем. Слід зазначити, що у східній медицині діагностика виконується на організменому рівні, а групи патологій класифікуються за порушенням енергетичного балансу на рівні меридіанів (і відповідні лікувальні процедури – вплив на біологічно активні точки). Подальша деталізація діагнозу вимагає додаткових обстежень пацієнта на кожному етапі деталізації, причому в кожній групі захворювань проводяться свої обстеження, включаючи біосигнали і медичні зображення.

Відповідно до цього, модель ОД зображується ієрархічною структурою деталізації діагнозу з залученням на кожному етапі деталізації додаткової діагностичної інформації. При цьому якщо на першому етапі використовується множина дихотомічних сигнальних ознак, значення яких не перетинаються (або слабко перетинаються) для діагностики класів даного рівня, то на наступних етапах множина ознак стає різнорідною (дихотомічні, рангові, числові). Ознаки відображають інші рівні взаємодії, стають залежними. Тому необхідно використовувати різні стратегії їх класифікації на початковому і наступному етапах деталізації.

Перший етап реалізується на основі детерміністичних продукційних правил (для кожної групи будується вектор бінарних ознак – симптомокомплекс даної групи), і аналогічний вектор вимірюваних ознак пацієнта послідовно порівнюється із симптомокомплексами всіх груп захворювань. У випадку збігу ознаки з одним із симптомокомплексів відбувається перехід на наступний етап, а у випадку розбіжності ознаки зо всіма симптомокомплексами – виключаються ті групи захворювань, у яких основні сигнальні ознаки протилежні вимірюваним ознакам пацієнта. Для груп, що залишилися, проводиться додаткове обстеження пацієнта з метою деталізації ознак із застосуванням іншої стратегії.

Як наступна стратегія використовується класифікація за допомогою

нейронної мережі, для якої вхідним вектором є розширений вектор ознак, а число вихідних нейронів відповідає числу класів даного рівня (другого рівня). У цьому випадку рішення виходить при неповному збігу вектора ознак пацієнта із симптомокомплексом, тому що рівень збігу для активізації вихідного нейрона задається параметром подібності.

У першій і другій стратегіях можливе застосування експертних оцінок: у першій стратегії експерт приймає рішення про належність пацієнта до однієї з груп при неповному збігу, а в другій стратегії у нейронній мережі додаються додаткові поля у вхідному векторі (додаткові вхідні нейрони) у яких враховується експертна інформація.

Третя стратегія застосовується на тому етапі деталізації ознак, коли є множина різнорідних залежних ознак і застосування перших двох стратегій неефективно. Основою стратегії є синтез структурованих моделей ОД (модель ознак і діагнозів) і синтезу комбінованого РП на цих моделях, що враховують апріорні імовірності й експертні оцінки структур симптомокомплексів.

Контрольні питання

1. Мета створення інженерно-фізіологічних та біотехнічних систем?
2. Які проблеми виникають при створенні біотехнічних систем та як вони вирішуються?
3. Чому треба враховувати еволюційні закономірності організації живих систем при створенні інженерно-фізіологічних та біотехнічних систем?
4. Чому вищі форми життя мають енергетичний виграш?
5. Що таке гомеостаз?
6. У чому суть рівноважного стану організму?
7. Як розподіляються часові показники діяльності організму?
8. Як часовий характер діяльності організму впливає на структуру математичної моделі?
9. Які особливості відображень внутрішніх станів систем організму на системи діагнозів та діагностичних показників?
10. Які особливості зв'язків стани – ознаки – діагнози?
11. Яким чином будується ієрархічна модель об'єктів медичної діагностики?
12. Які стратегії класифікації використовуються на різних рівнях деталізації діагнозу?

2.6. Синтез динамічних моделей роботи підсистем організму

2.6.1. Моделювання вуглеводного обміну в організмі й розвитку цукрового діабету

*Опис роботи системи регуляції цукру в крові***.** Розповсюдження цукрового діабету і зниження вікових меж серед хворих викликають особливу тривогу у лікарів і дозволяють говорити про епідемію цукрового діабету. У промислово розвинених районах 5-6 % населення страждає від цього захворювання. Кожні 10–15 років число хворих на цукровий діабет подвоюється. За даними Всесвітньої організації охорони здоров'я, сьогодні в світі налічується більше 150 млн. хворих на цукровий діабет. Розпізнавання цукрового діабету на ранніх стадіях захворювання, а також успішне лікування цього захворювання можливе тільки при серйозних дослідженнях порушень регуляції цукру крові, у тому числі і на моделях. Побудова моделей вуглеводного обміну в організмі відкриває шлях сучасним комп'ютерним технологіям для їх упровадження в медичну практику лікаря-ендокринолога.

Моделювання процесів вуглеводного обміну дозволяє оцінювати стан людини, виявляти порушення цього обміну [1, 19, 20]. Використовування комп'ютерних технологій, зокрема, візуалізація динаміки поведінки основних параметрів системи мають діагностичне значення.

Оптимізація інсулінотерапії вимагає створення спеціальних пристроїв, що забезпечують постійне мікродозоване введення рідких лікарських засобів хворому з метою досягнення їх ефективної дії. Розробка таких пристроїв – дозаторів інсуліну, вибір дози неможливі без вивчення системи регуляції цукру крові на моделях і побудови відповідних алгоритмів введення інсуліну.

На рис 2.13 наведена загальна схема системи регуляції рівня цукру в крові**.**

Вуглеводи – це речовини, молекули яких складаються з вуглецю, кисню і водню. У результаті обміну речовин вони перетворюються на глюкозу – важливе енергетичне джерело для організму.

Основною задачею системи вуглеводного обміну є підтримка стабільного значення концентрації цукру в крові. Основний орган, що бере участь в регулюванні, – печінка. Вона виконує роль депо цукру. Під дією інсуліну печінка поглинає цукор і запасає його у вигляді глікогену. Через шлунково-кишковий тракт при вживанні їжі в кров і на вхід печінки надходить глюкоза. Швидкість надходження цукру в кров залежить від властивостей їжі, що приймається.

Підшлункова залоза синтезує інсулін залежно від концентрації цукру в крові. Тканини (жирові тканини, м'язи) також запасають цукор під дією

інсуліну.

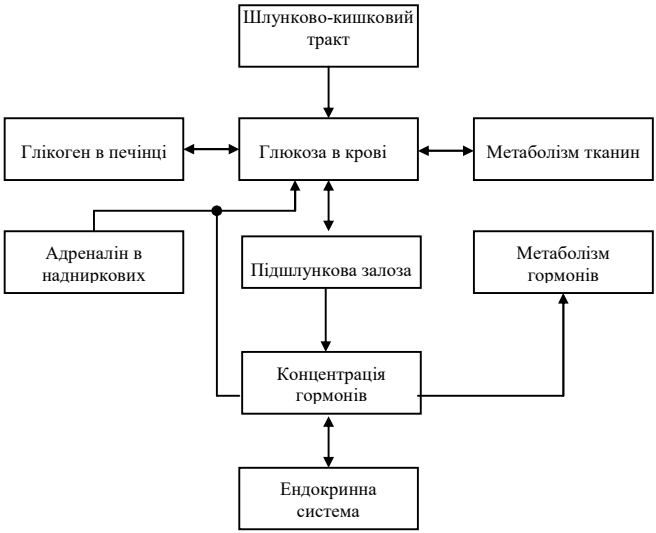


Рис. 2.13 – Загальна схема системи вуглеводного обміну в організмі

Глюкоза забезпечує енергетичні потреби організму. Наприклад, в стресових ситуаціях, коли тварині або людині потрібно рятуватися від переслідування або вступити в сутичку з ворогом, для забезпечення додаткових фізичних навантажень нервова система подає сигнал. і в організмі виробляється адреналін. Адреналін сприяє розпаду глікогену і підвищенню концентрації глюкози в крові. При розробці моделей задачею розробника є створення моделі:

1) яка дозволяє описати динаміку зміни концентрації цукру і гормонів в організмі;
2) дає можливість врахувати різні патологічні зміни вуглеводного обміну в організмі і дослідити гіпотезу виникнення діабету у людини;
3) може бути використана при розробці діагностики захворювання цукровим діабетом;
4) може знайти застосування при розробці алгоритму введення інсуліну хворому при створенні біотехнічних комплексів.

Розглянемо відомі моделі вуглеводного обміну в організмі.

Моделі вуглеводного обміну з урахуванням концентрації глюкози і інсуліну. Одна з найпростіших моделей вуглеводного обміну є модель з урахуванням тільки двох суттєвих змінних: концентрації глюкози і інсуліну. Якісний опис біохімічних процесів в організмі здорової людини в цьому

випадку можна стисло подати так. Стійким станом, що приймається за точку відліку, є рівень цукру в крові при голодуванні y і при нульовому рівні інсуліну i. Зміна цих змінних відбувається в результаті дії декількох незалежних механізмів. Розглянемо ці механізми:

1) якщо рівень цукру в крові перевищує стійкий, то підшлункова залоза виробляє інсулін, що попадає в кровоносне русло; це явище можна описати так:

$$\frac{di}{dt} = \begin{cases} b_1(y - y_н), \ldots\ldots\ldots\ldots при\_ y \geq y_н \\ 0\ldots\ldots\ldots\ldots\ldots\ldots при\_ y < y_н \end{cases}; \tag{2.58}$$

2) концентрація самого інсуліну зменшується під впливом декількох біохімічних процесів; у живому організмі половина вільного інсуліну інактивується за час від 10 до 25 хв. Отже, можна записати

$$\frac{di}{dt} = -b_2 y, \quad y \geq 0; \tag{2.59}$$

3) будь-яке зовнішнє джерело інсуліну можна описати доданком, який для здорового організму дорівнює 0, а для хворого на діабет він буде функцією часу, визначуваною графіком ін'єкцій:

$$\frac{di}{dt} = -b_3 w(t), \quad y \geq 0. \tag{2.60}$$

Сталі b_1, b_2, b_3 за визначенням позитивні. Вони можуть бути названі чутливостями, що є відповідно чутливостями градієнта інсуліну до високого рівня цукру в крові, до рівня інсуліну, до введення інсуліну. Ураховуючи вирази (2.58 – 2.60), динаміку зміни рівня інсуліну в крові можна зобразити у такому вигляді:

$$\frac{di}{dt} = b_1(y - y_н) - b_2 y + b_3 w(t) \tag{2.61}$$

При розгляді динаміки зміни рівня цукру в крові виділяються такі фактори:

1) присутність інсуліну приводить до метаболізму цукру, що знижує його вміст в крові. Чим вищий вміст цукру в крові або рівень інсуліну, тим швидше відбувається це зниження. Звідси випливає, що принаймні, для малих змін цей ефект можна описати так

$$\frac{dy}{dt} = -a_1 yi; \tag{2.62}$$

2) рівень цукру в крові може впасти до рівня, що нижчий від

рівноважного (наприклад, в результаті великого фізичного навантаження або в результаті голодування). Для того щоб підняти його до нормального рівня, вивільняються запаси вуглеводів з печінки. Отже,

$$\frac{dy}{dt} = \begin{cases} a_2(y_н - y).....при.....y < y_н \\ 0.....при.....y \geq y_н \end{cases}; \tag{2.63}$$

3) має місце також незначне природне падіння концентрації цукру, вплив цього чинника малий і включити його в модель можна так

$$\frac{dy}{dt} = -a_2'(y - y_н). \tag{2.64}$$

4) зовнішнім джерелом цукру в крові є споживана їжа, що можна описати так:

$$\frac{dy}{dt} = a_3 z(t). \tag{2.65}$$

Сталі a_1, a_2, a_2', a_3 також позитивні і є відповідно чутливостями градієнта цукру до присутності інсуліну, до низького рівню цукру в крові, до високого рівня цукру в крові, їжі. Ураховуючи вирази (2.62 – 2.65), динаміку зміни рівня цукру в крові можна зобразити у такому вигляді:

$$\frac{dy}{dt} = -a_1 yi - a_2(y_н - y) - a_2'(y - y_н) + a_3 z(t). \tag{2.66}$$

У нормі надходження цукру в кров залежить від їжі, а не від безпосереднього введення його в кров'яне русло. Запаси їжі в організмі поповнюються періодично, а не безперервно; крім того, передбачається, що на будь-якій стадії ці запаси зменшуються експоненційно. Тоді відповідний доданок можна навести у вигляді

$$z(t) = \begin{cases} 0.....при.....t < t_0 \\ Q\exp(-K(t - t_0)).....при.....t \geq t_0, \end{cases} \tag{2.67}$$

де Q – кількість калорій в їжі; K – параметр, t_0 – час їжі.

Таким чином, дану модель можна зобразити системою диференціальних рівнянь

$$\begin{aligned}
\frac{di}{dt} &= b_1(y - y_н) - b_2 y + b_3 w(t) \\
\frac{dy}{dt} &= -a_1 yi - a_2(y_н - y) - a_2'(y - y_н) + a_3 z(t) \\
z(t) &= \begin{cases} 0 \ldots\ldots\ldots\ldots\ldots\ldots \text{при} \ldots\ldots t < t_0 \\ Q\exp(-K(t - t_0)) \ldots \text{при} \ldots\ldots t \geq t_0. \end{cases}
\end{aligned} \qquad (2.68)$$

Початкові умови $y\big|_{t=0} = y_н$; $i\big|_{t=0} = i_0$.

Відзначимо, що відхилення вмісту цукру в ту або іншу сторону (підвищення або зниження щодо рівноважного рівня) компенсується двома різними стабілізуючими процесами. Крім того, перебіг основного процесу видалення цукру залежить від вмісту як цукру, так і інсуліну. Слід підкреслити, що двопараметрична модель (2.68) найпростіша з тих, які можуть зображати реальність.

Один з головних доказів на користь даної простої моделі полягає у тому, що інформація, одержувана в результаті її розв'язання, узгоджується з клінічною практикою. Основним визначуваним параметром є рівень цукру в крові, а система в цілому стабілізується дієтою і ін'єкціями інсуліну. Графіки, одержані на основі розв'язання системи рівнянь (2.68) показують, що при кожному прийомі їжі рівень цукру в крові підіймається, стимулюючи тим самим виробництво інсуліну, наявність інсуліну в крові приводить до зниження вмісту цукру, і, в свою чергу, рівень самого інсуліну знижується унаслідок природного процесу розпаду.

Параметри b_1, b_2, b_3, a_1, a_2, a_2', a_3 визначаються шляхом порівняння результатів моделювання з результатами спостережень. Дослідження поведінки системи на моделі дозволяє зробити певні висновки, які не випливають безпосередньо з результатів вже відомих спостережень.

Модель діабету можна одержати, вводячи інші значення чутливостей цукру до інсуліну a_1 і інсуліну до цукру b_1, стану діабету відповідає зменшення цих значень. Таке зменшення чутливостей a_1 і b_1 робить систему нестійкою, після чого система сама вже не зможе повернутися до стійкого стану. Слабкі нестійкості, тобто діабет в легкій формі при невеликих змінах параметрів, що характеризують чутливість, можна нормалізувати відповідним вибором тільки функції $z(t)$. Це є класичним методом лікування за допомогою дієти. У разі діабету у важкій формі для відновлення стійкості системи необхідне введення в рівняння другої управляючої функції $w(t)$. Основна задача тепер полягає в розумному виборі такого режиму введення інсуліну $w(t)$, щоб поведінка одержаної в результаті системи і відгук рівнів цукру і інсуліну на звичну їжу достатньо добре нвідповідали нормальним. Шуканою відповіддю буде добова кількість інсуліну, що вводиться, яка, не

дивлячись на зміни в деяких межах нормального вмісту цукру протягом дня, підтримуватиме збурення рівня цукру в певних межах, а також підтримуватиме періодичність функціонування системи протягом багатьох днів.

Моделі вуглеводного обміну з урахуванням концентрації глюкози, інсуліну і адреналіну. Основний симптом, що свідчить про захворювання цукровим діабетом, – підвищений в порівнянні з нормою вміст глюкози в крові. Встановлено, що головним регулятором нормального рівня глюкози є інсулін. Порушення діяльності підшлункової залози, пов'язане із зменшенням або відсутністю секреції інсуліну, приводить до підвищення концентрації глюкози в крові і до сповільненого її засвоєння клітинами. Адреналін, що надходить в кров, готує систему до майбутніх енергетичних навантажень і сприяє підвищенню концентрації глюкози в крові.

На початку минулого сторіччя для лікування хворих на цукровий діабет став використовувати штучно синтезований інсулін, який вводиться підшкірно 1-2 або більше разів на добу. Такий спосіб введення не дозволяє нормалізувати добовий профіль глюкози, проте значно продовжує життя хворого і віддаляє ускладнення, якими супроводжується діабет.

У 70-ті роки завдяки розвитку сучасної техніки в практиці лікування діабету стали використовувати портативні насоси, що інфузують в організм розчин інсуліну за наперед заданою програмою. Створення апаратури для автоматичної підтримки заданого профілю глюкози протягом доби є складною технічною задачею. Управління лікуванням діабету при застосуванні технічних засобів стало можливим завдяки розробці спеціальних математичних моделей "організм людини – біотехнічна система". Для цієї мети розробляються достатньо повні описи процесів вуглеводного обміну в організмі і його регуляції підшлунковою залозою.

Розглянемо основні характеристики системи регуляції цукру в крові [19]. Спираючись на відомі відомості з фізіології, виділимо змінні, які впливають на основну координату – рівень цукру в крові. При проведенні активного експерименту (введення глюкози або введення інсуліну) у цієї системи легко може бути виміряна тільки вихідна координата. Решту змінних в експерименті виміряти важко або неможливо; рівень цукру в артеріальній крові y визначається двома компонентами: рівнем цукру на виході печінки y_2 і рівнем цукру на виході тканин організму (венозна кров) y_3. На вхід печінки в порожнисту вену надходить кров, що містить відмінний від артеріальної рівень цукру y_1. Наступними основними змінними, що впливають на поточний вміст цукру в крові, є: адреналін x_1, що виробляється надниркovими; ендогенний інсулін i, що виробляється підшлунковою залозою; екзогенний інсулін i_1, який можна ввести в організм ззовні. Крім

того, нервова система впливає на встановлення нормального значення рівня цукру в крові $y_н$. Відзначимо, що перераховані змінні не вичерпують всіх чинників, що впливають в організмі на рівень цукру в крові.

Уважний розгляд відомих фізіологічних даних, що випливають з аналізу внутрішньої структуризації системи регуляції цукру, дозволив скласти спрощену структурно-функціональну схему цієї системи. Взагалі кажучи, для будь-якої біосистеми структурно-функціональна схема дозволяє змістовніше обговорити вплив змінних одна на одну. Важливо тільки, щоб в структурній схемі тієї або іншої біосистеми достатньо повно був врахований сучасний рівень знання про функціонування досліджуваної біосистеми. Будь-яка структурна схема передбачає фіксацію в ній причинно-наслідкових зв'язків між змінними. Розглянемо, як відображаються в структурній схемі системи регуляції цукру крові зв'язки між змінними. Введення глюкози в шлунок приводить до виникнення процесу всмоктування її в кров у шлунку і кишечнику. Ця кров змінює концентрацію цукру в порожнистій вені y_1 на вході печінки. Таким чином, причиною зміни концентрації цукру y_1 є введення глюкози в дозі G. Решта змінних не впливає на процес зміни концентрації цукру в поржнистій вені.

Вироблення ендогенного інсуліну i підшлунковою залозою посилюється, якщо поточна концентрація цукру в крові y більше норми $y_н$, що потребує в даний момент організм. Отже, причиною зміни концентрації інсуліну в крові є перевищення порівняно з нормою поточного вмісту цукру в крові. Решта змінних, можна вважати, не має впливу на цей процес. Внутрішньом'язова ін'єкція інсуліну I приводить до виникнення процесу розсмоктування його з області ін'єкції з подальшим рознесенням підвищеної концентрації інсуліну i_1 кров'ю по всіх тканинах організму.

Причиною підвищеного вироблення адреналіну x корою надниркових є зменшення поточного вмісту цукру в крові відносно норми. Можна вважати також, що на цей процес не впливають зміни решти змінних системи.

Відомо, що цукор крові може відкладатися в тканинах організму у вигляді глікогену. Цей процес приводить до розбіжності між концентраціями цукру в артеріальній крові y, що притікає до тканин, і у венозній y_3, що відтікає від тканин крові. З фізіологічних досліджень відомо, що підвищена концентрація інсуліну (ендогенного або екзогенного) в плазмі крові сприяє проникненню цукру крові в клітини тканин організму і відкладенню його в глікоген. Розглянуті вище змінні є основними, що визначають динаміку зміни концентрації цукру венозної крові y_3.

Найскладніші процеси відбуваються в печінці – основному регуляторному органі системи. Для неї вхідними змінними є концентрація цукру в порожнистій вені, перевищення поточної концентрації цукру в артеріальній крові відносно норми. Відомо, що інсулін (ендогенний і

екзогенний) діє на тканини печінки так само, як і на решту тканин організму, підсилюючи процес відкладення цукру в глікоген. Підвищена концентрація адреналіну в крові, на відміну від інсуліну, діє на тканини печінки протилежним чином, сприяючи розпаду глікогену і підвищенню концентрації цукру па виході печінки.

Поточна концентрація цукру в артеріальній крові визначається її нормальним стабілізованим значенням $y_н$ і двома компонентами венозної крові: на вході печінки y_2, і на виході решти тканин організму y_3. З погляду постійних часу, система регуляції цукру крові є на порядок більш інерційна, ніж серцево-судинна система, тому впливом серцево-судинної системи на систему регуляції цукру крові можна нехтувати.

Викладені вище фізіологічні гіпотези дозволяють розв'язати задачу ідентифікації структури математичної моделі системи шляхом обчислення в ній тільки істотних зв'язків між змінними. У зв'язку з тим, що в експерименті не вдається контролювати змінні системи, окрім вихідної, при складанні рівнянні динаміки для кожної змінної використовується критерій мінімальної складності математичної моделі. Він передбачає запис рівняння динаміки у вигляді лінійного неоднорідного диференціального рівняння першого порядку. Для останнього рівняння, що визначає концентрацію цукру в артеріальній крові, використовується додаткова фізіологічна гіпотеза, що дозволяє нехтувати динамікою функціонування серцево-судинної системи організму в порівнянні з системою регуляції цукру крові.

Запишемо систему диференціальних рівнянь, яка була одержана на підставі фізіологічних гіпотез [19]:

рівняння всмоктування глюкози в шлунково-кишковому тракті

$$\frac{dy_1}{dt} + a_1 y_1 = \begin{cases} k_1 g^{(1)}, & \text{якщо} \quad 0 \le t \le \tau \\ 0, & \text{якщо} \quad t > \tau \end{cases}; \tag{2.70}$$

рівняння глікогенної функції печінки

$$\frac{dy_2}{dt} + a_4 y_2 = -a_2 (y - y_н) + a_3 y_1 - b_1 i + c_1 x - b_4 i_1; \tag{2.71}$$

рівняння вироблення інсуліну підшлунковою залозою

$$\frac{di}{dt} + b_2 i = a_5 (y - y_н), y - y_н > 0; \tag{2.72}$$

рівняння вироблення адреналіну надниркovими

$$\frac{dx}{dt} + c_2 x = a_6 (y - y_н), y - y_н < 0; \tag{2.73}$$

рівняння використовування глюкози тканинами організму

$$\frac{dy_3}{dt} + a_8 y_3 = a_7\left(y - y_3\right) + b_3 i + b_5 i; \tag{2.74}$$

рівняння розсмоктування введеного внутрішньом'язового інсуліну має вигляд:

$$\frac{di_1}{dt} + b_6 i_1 = \begin{cases} k_2 i_1^{(1)}, & \text{якщо} \quad 0 \le t \le \tau_i \\ 0, & \text{якщо} \quad t > \tau_i \end{cases}; \tag{2.75}$$

Концентрація глюкози в артеріальній крові рівна:

$$y - y_н = y_2 + y_3 \tag{2.76}$$

У рівняннях (2.70) – (2.76):

y – концентрація глюкози в артеріальній крові;

y_1 – концентрація глюкози в порожнистій вені печінки;

y_2 – концентрація глюкози на виході печінки;

y_3 – концентрація глюкози у венозній крові, що відтікає від тканин організму;

i – концентрація ендогенного інсуліну в крові;

x – концентрація адреналіну в крові;

i_1 – концентрація в крові екзогенного інсуліну;

$y_н$ – вміст цукру в крові в нормі;

$g(1)$ – швидкість всмоктування глюкози в шлунково-кишковому тракті;

$i(1)$ – швидкість розсмоктування екзогенного інсуліну після внутрішньом'язової ін'єкції;

τ – час всмоктування глюкози;

τ_i – час розсмоктування екзогенного інсуліну;

a,b,c,k – коефіцієнти розмірності і пропорційності, що підлягають аналізу при синтезі моделі.

Основною задачею системи вуглеводного обміну є підтримка стабільного значення цукру в крові. Разом з тим. при дії зовнішніх чинників рівень цукру може значно відрізнятися від нормального. Так, одноразове надходження в організм глюкози приводить до появи перехідного процесу в системі вуглеводного обміну. Моделювання функціонування системи регуляції рівня цукру в крові можливе шляхом розв'язання системи диференціальних рівнянь на комп'ютері і дослідження поведінки функцій, що цікавлять нас, на графіках. Виберемо для розв'язання системи (2.70 – 2.75) метод Рунге–Кутта. Цей метод дозволяє розв'язати систему з хорошою

точністю за прийнятний час.

Математична модель дозволяє досліджувати фізіологічні системи в різних умовах, зокрема відтворювати ситуації, пов'язані з «гострими» дослідами на живому організмі. Зв'язки в системі можуть змінюватися залежно від умов зовнішнього середовища, причому структура системи середовища може перебудовуватися або за рахунок вікових змін, або штучно в результаті хірургічного втручання. З цієї точки зору важливим є вивчення ролі кожного зв'язку системи у загальному процесі регулювання. Розглянемо поведінку системи регуляції цукру в крові на моделі у разі послідовного розриву основних зв'язків.

Поведінка системи з нормальною структурою при навантаженні її глюкозою показана на рис. 2.14, система виведена із стану рівноваги дозою глюкози 1333 міліграм/кг.

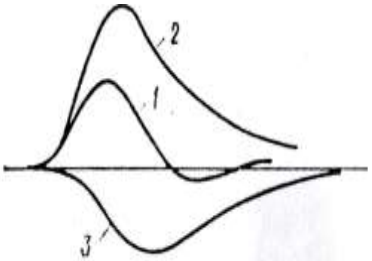


Рис. 2.14 – Характеристики перехідних процесів при навантаженні системи глюкозою у випадку, коли структура системи не порушена: *1* – $y(t)$; *2* – $y_2(t)$; *3* – $y_3(t)$.

Криві *1 – 3* на цьому рисунку описують перехідні процеси зміни рівня цукру: *1* – на виході всієї системи, *2* – в печінковій вені, *3* – в тканинах. Крива *1* – це глікемічна крива, вона є результатом алгебраїчного сумування кривих *2* і *3*. На виході печінки маємо процес підвищення концентрації глюкози після прийому їжі або після введення глюкози. Але ця глюкоза водночас споживається тканинами. Алгебраїчна сума отриманих кривих дає процес змінювання концентрації глюкози в системі.

Якщо печінка не чутлива до інсуліну ($b_1 = 0$), то криві *1 – 3* мають дещо інший вигляд. Поведінка системи в цьому випадку показана на рис. 2.15, де для порівняння з роботою нормальної системи наведено відповідні перехідні процеси у разі, коли печінка не чутлива до інсуліну.

Як видно, глікемічна крива на рис. 2.14 відрізняється від такої на рис. 2.15 висотою максимуму, відсутністю перерегулювання і значної затягнутістю стадії гіперглікемії. Такий вигляд кривих відповідає ослабленій ефективності роботи системи, що є результатом нечутливості печінки до інсуліну.

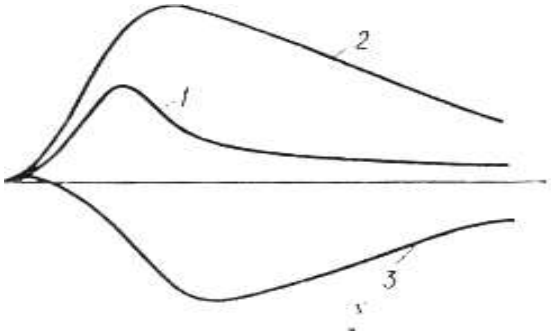


Рис. 2.15 – Характеристики перехідних процесів при навантаженні на систему глюкозою у випадку, коли відсутня чутливість печінки до інсуліну: *1* – $y(t)$; *2* – $y_2(t)$; *3* – $y_3(t)$

Треба відмітити, що в результаті нечутливості печінки до інсуліну в самій печінковій тканині значно ослабиться процес утворення глікогену, тому процес зміни концентрації цукру в крові і печінковій вені буде істотно змінений убік погіршення: максимум кривої, що описує цей процес, завищений, тривалість процесу затягнена (крива *2* на рис. 2.15). Такі зміни в печінковій вені неминуче позначаються на роботі решти частини системи.

Погіршення якості роботи одного органа (печінки) примушує решту органів системи підсилити свою регулюючу дію для боротьби з гіперглікемією. Зважаючи на те, що печінка не здатна перетворити на глікоген певну частину глюкози, тканини повинні поглинути її більше, ніж в здоровій структурі з нерозірваним зв'язком (крива *3* на рис. 2.15), а щоб забезпечити тканинам таку можливість, підшлункова залоза підсилює інсулярну функцію.

Таким чином, у разі, коли печінка не чутлива до інсуліну, система приводить рівень глікемії до початкового, мобілізуючи засоби, що при цьому залишилися, тому і глікемічна крива 1 на рис. 2.15 декілька відрізняється від нормальної.

Якщо тканини не чутливі до інсуліну ($b_3 = 0$), характер поведінки кривих буде іншим. Поведінка системи ілюструється рис. 2.16, коли система при нечутливості тканин до інсуліну виведена із стану рівноваги дозою глюкози 1333 міліграм/кг (порівняйте з рис. 2.14).

Як видно, відсутність чутливості тканин до інсуліну вносить деякі зміни в характер процесів, що вивчаються, в порівнянні з відповідними перехідними процесами в нормальній системі. Дійсно, тканини, не відчуваючи інсуліну, поглинають дуже мало глюкози: тільки ту її частину, яка проникає в них в результаті дифузії. Це і підтверджується виглядом

кривої *3* на рис. 2.16. Мінімум y_3 значно менший, і перехідний процес закінчується набагато швидше, ніж в нормі. Цікаво відзначити, що у момент настання гіпоглікемії з деякою інерцією в тканинах починається зворотний процес видачі глюкози в кров (крива *3* на рис. 2.16).

Природно, що при нечутливості тканин до інсуліну декілька збільшується інсулінова активність печінки, оскільки на її частку тепер доводиться більше інсуліну. Дія його виражається у наявності гіпоглікемічної фази в цукровій кривій печінкової вени (порівняйте з 2 на рис. 2.14).

Вказані зміни в поглинальній здатності тканин відбиваються відповідним чином і на роботі підшлункової залози, яка підсилює функціональну діяльність свого апарату, щоб інсулін, що виділився, впливаючи на печінку, міг певним чином компенсувати порушення інсулінової активності тканин.

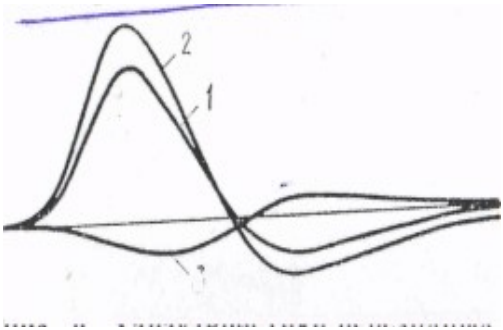


Рис. 2.16 – Характеристики перехідних процесів при навантаженні системи глюкозою у випадку, коли відсутня чутливість тканин до інсуліну: *1* – $y(t)$; *2* – $y_2(t)$; *3* – $y_3(t)$

Хід глікемічної кривої в цьому випадку відображає результат сумісної направленої взаємодії всіх органів в системі зі зміненою структурою. У результаті виключення з системи важливої поглинальної функції тканин гіпоглікемічна крива стає більш вираженою у зв'язку із збільшенням кількості інсуліну в системі. Змінюється і характер глікемічної кривої на виході системи, гіперглікемічний пік її збільшується (крива *1* на мал. 2.14, 2.15, 2.16).

Таким чином, у результаті відсутності інсулярної залежності між підшлунковою залозою і тканинами відбувається погіршення якості перехідного процесу зміни вмісту цукру в крові, збільшення відхилення в обидві сторони у гипер- і гіпоглікемічної фазах.

Якщо печінка не чутлива до адреналіну ($c_1 = 0$), то це еквівалентно усуненню з системи інсулярних зв'язків, що характеризує роботу системи в

умовах гіперглікемії. У цих умовах найістотніша роль в регуляції належить контрінсулярному апарату і зв'язкам, які здійснюються цим апаратом. Один з важливих зв'язків здійснює гормон мозкового шару надниркових – адреналін. На рис. 2.17 наведено рішення рівнянь моделі у разі нечутливості печінки до адреналіну.

Видно, що гіпоглікемічний пік кривої *2* нижчий, ніж кривою *1*, і у разі кривої *2* значно затягнена тривалість гліпоглікемічного стану. Причиною такого затягування процесу є ослаблення функцій контрїнсулярного апарату через відсутність чутливості печінки до адреналіну. З цієї точки зору цікавим є вивчення роботи самої печінки.

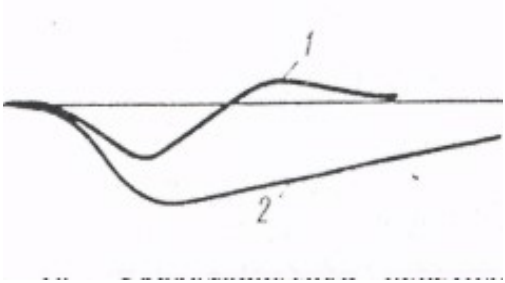


Рис. 2.17 – Характеристики перехідних процесів у системі за відсутності чутливості печінки до адреналіну при дії на неї інсуліном: *1* – $y_2(t)$ в нормі; *2* – $y_2(t)$ за відсутності чутливості печінки до адреналіну

Цікаво відзначити, що тканини в останньому випадку змінюватимуть свою функцію утилізації глюкози на зворотну — виділення її в кров. Про це свідчить не перехід кривих в область гіперглікемії. Ймовірно, при розвинутій глибокій і тривалій гіпоглікемії при нечутливості печінки до адреналіну один гомеостатичний механізм печінки не може привести рівень цукру в крові до норми і до нього підключається тканини.

Це порушення зв'язку в системі викликає певні зміни і у функції надниркових. Працездатність їх, як було показано вище, залежить від рівня цукру в крові. Останній же змінюється із змінами в печінці (відсутність чутливості до адреналіну). Посилення здатності виділення адреналіну надниркових можна пояснити погіршенням якості перехідного процесу зміни вмісту цукру в крові, що виявляється в тривалій і більш вираженій гіпоглікемії. Як видно з приведених результатів, відсутність чутливості печінки до адреналіну викликає цілком визначені, направлені зміни в динаміці функцій всіх інших органів, що борються з гіпоглікемією.

Модельні експерименти з розривом зв'язків говорять про достатньо високу надійність системи, хоча ці порушення і приводять до погіршення якості перехідного процесу на виході системи. Спотворення перехідних процесів на виході пошкоджених ланок більш істотне, і лише мобілізація

елементів, що збереглися, і перебудова їх роботи в ситуації, що склалася, знижують негативний вплив розривів на динаміку вихідного процесу.

Таким чином, дослідження моделі на комп'ютері дозволяє, з одного боку, проводити такі дослідження, яке неможливо провести па живому об'єкті, а з іншого – судити про достовірність структури і параметрів математичної моделі біосистеми.

Моделі з урахуванням концентрації інсуліну й жирів. Тісний взаємозв'язок вуглеводного обміну з іншими видами метаболізму, наприклад ліпідного, підказує, що моделі, обговорювані в цьому розділі, можна використовувати для дослідження основних параметрів вуглеводного і ліпідного метаболізму – рівнів глюкози і вільних жирних кислот (ВЖК) крові. Порушення, що виникають в системі обміну при діабеті, диктують необхідність розробки алгоритмів, які, зокрема, забезпечили б нормалізацію цих основних параметрів вуглеводного і ліпідного метаболізму.

Аналіз зазначених параметрів можливий із застосуванням моделей регуляції глюкози крові з метою управління не тільки рівнем глюкози як основним показником вуглеводного обміну, але і рядом інших речовин в крові хворого, у тому числі і рівнем вільних жирних кислот.

На основі біологічної ідентифікації процесу, що вивчається, і експериментальних даних встановлено, що перевищення, порівняно з нормою, рівня ВЖК приводить до збільшення концентрації глюкози, а введення лікарських речовин (інсуліну і обзідану) знижує концентрацію глюкози. Додаткове введення глюкози включає механізми регуляції, що також знижує її концентрацію. Рівень ВЖК підвищується при збільшенні концентрації глюкози і знижується при введенні інсуліну і обзідану. Додаткове введення ВЖК також приведе до включення механізму зниження цього показника.

Математична модель управління рівнем глюкози і ВЖК крові може бути наведена у вигляді двох диференціальних рівнянь:

$$\frac{dY}{dt} = -a_{11}(Y - Y^*)(1 + r_1 I + r_2 \Theta + \sigma_1 G) + a_{12}(N - N^*), \qquad (2.77)$$

$$\frac{dN}{dt} = a_{21}(Y - Y^*) - a_{22}(N - N^*)(1 + r_3 I + r_4 \Theta + \sigma_2 F), \qquad (2.78)$$

де Y і Y^* – поточний і фізіологічний рівні глюкози в крові; N і N^* – поточний і фізіологічний рівні ВЖК в крові; I, Θ, G і F – швидкість введення інсуліну, обзідану, глюкози і ВЖК відповідно; a_{ij}, r_k, σ_i – коефіцієнти розмірності і пропорційності відповідних змінних.

Таким чином, вуглеводний обмін тісно пов'язаний з рівнем вільних жирних кислот в організмі, тому моделі вуглеводного обміну допомагають

зрозуміти і пояснити проблеми ожиріння. Довгий час вважали, що ожиріння – це є результат споживання багатих калоріями продуктів, а не порушення обміну речовин. Сучасний погляд на цю проблему спирається на теорію вуглеводного обміну в організмі. Теорія про калорії заснована на першому законі термодинаміки (закон збереження енергії) в застосуванні до біологічних об'єктів.

Згідно з цією теоріэю при надмірному споживанні калорій їх надлишок відкладається у вигляді резервного жиру. Але досвід говорить про зворотне. Як вдавалося виживати ув'язненим, що одержували від 700 до 800 кілокалорій на день в концентраційних таборах? З іншого боку, є люди, що споживають щоденно до двох норм кілокалорій і не страждають надмірною вагою. І нарешті, в даний час добре відомо, що після втрати маси в результаті численних дієт людина знову набирає вагу, навіть не збільшуючи число кілокалорій, що приймаються щодня. Низькокалорійна дієта дає ефект на невеликому відрізку часу. Надалі організм пристосовується до низького рівня калорій, а інстинкт виживання все одно спонукає його до створення резерву, тобто до накопичення жирів.

Як ми тільки що встановили, під час вживання їжі рівень глюкози спочатку підвищується – настає так звана гіперглікемія, а після того як підшлункова залоза виділила інсулін, рівень глюкози в крові падає (гіпоглікемія), а потім повертається до колишнього рівня. Пік вмісту цукру спостерігається через півгодини після вживання їжі. Здатність їжі, і перш за все вуглеводів, викликати підвищення рівня цукру в крові (гіперглікемію) визначається глікемічним індексом. Цей термін вперше був введений в 1976 році. Глікемічний індекс відповідає площі під глікемічною кривою в період гіперглікемії, тобто є інтегралом функції $y(t)$. Глікемічний індекс тим вищий, чим вища гіперглікемія, викликана розщеплюванням вуглеводів. Глікемічний індекс глюкози приймають за 100. У табл. 2.3 наведено глікемічні індекси різних продуктів.

Таблиця 2.3 – Глікемічні індекси (ГК) продуктів харчування

Вуглеводи з високим індексом	ГК	Вуглеводи з низьким індексом	ГК
Глюкоза	100	Хліб з муки грубого помелу	50
Печена картопля	95	Горох	50
Білий хліб з борошна вищого ґатунку	95	Вівсяні пластівці	40
Картопляне пюре швидкого приготування	90	Фруктовий сік свіжий без цукру	40
Мед	90	Молочні продукти	35
Морква	85	Житній хліб	30
Шоколад	70	Шоколад чорний (60 % какао)	22
Печиво	70	Зелені овочі, томати, гриби	15

Вуглеводи з високим глікемічним індексом (>50) є причиною надмірної ваги. Процес накопичення або ненакопичення резервного жиру пов'язаний з виділенням інсуліну. Інсулін, впливаючи на глюкозу, що міститься в крові, допомагає їй проникати в тканини організму. Глюкоза або негайно задовольняє енергетичні потреби, або, якщо її кількість велика, сприяє накопиченню резервного жиру. Якщо підшлункова залоза працює нормально, вона виділить рівно стільки інсуліну, скільки його знадобиться для переробки глюкози, що надійшла. Якщо ж вона хвора, то кількість інсуліну, що виділяється, перевищуватиме дозу, необхідну для переробки глюкози. У результаті частина енергії буде відкладена про запас – в жир. Людина з нормальною підшлунковою залозою може їсти все і в будь-якій кількості, залишаючись при цьому в нормальній вазі і не товстіючи. Людина, схильна до ожиріння, має тенденцію до гіперінсулізму.

2.6.2. Моделювання в імунології

Опис роботи імунної системи організму. Імунна система — це сукупність всіх лімфоїдних органів і скупчень лімфоїдних клітин тіла, включаючи тимус (вилочкову залозу), селезінку, лімфатичні вузли, пейєрові бляшки, лімфоцити кісткового мозку і периферичної крові. Все це складає єдиний "дифузний" орган загальною масою близько 1,5 – 2 кг, загальне число лімфоїдних клітин ~ 10^{12}, ці клітини здійснюють найважливіші види імунологічного реагування.

Основною функцією імунної системи є захист організму від живих тіл і речовин, що несуть на собі ознаки генетично чужорідної інформації (таких, як бактерії, віруси, білки, клітини, тканини, змінені власні клітини, у тому числі і ракові), званих *антигенами*. Кожен антиген специфічний, тобто такий, що має в своїй структурі певні особливості, які відрізняють його від інших антигенів, і що викликає специфічну реакцію імунної системи, направлену тільки проти нього одного, а не проти якого-небудь іншого антигена або всіх антигенів взагалі. Функція імунної системи організму – розпізнавання антигена і специфічне реагування на нього незалежно від його походження (або екзогенного – зовнішнього: віруси, бактерії і т. ін., або ендогенного – внутрішнього: змінені власні клітини).

Розпізнавання антигена здійснюється клітинами-лімфоцитами, які виконують центральну роль в імунній системі і називаються імунокомпетентними клітинами або імуноцитами. Лімфоцити утворюються із стовбурових клітин, які виробляються кістковим мозком. Кістковий мозок є джерелом стовбурових клітин і повністю забезпечує ресурсами процес кровотворення, тобто стовбурові клітини є родоначальниці всієї решти клітин крові. Схема утворення лімфоцитів така. Стовбурові клітини виходять

з кісткового мозку в кровострум, циркулюють в організмі, надходять у вилочкову залозу (тимус) і інші лімфоїдні органи. У таких органах стовбурові клітини перетворюються на лімфоцити. (Процес перетворення супроводжується розмноженням і накопиченням лімфоцитів, частина яких знов надходить в кровострум.) Залежно від того, в якому органі відбулося перетворення, розрізняють дві популяції лімфоцитів: *Т*- і *В*-лімфоцити. Якщо перетворення відбулося в тимусі, то виходять *Т*-лімфоцити, а якщо у сумці Фабріциуса у птахів або в її аналогу у ссавців, то *В*-ліфмоцити. Аналогічно розрізняють *Т*- і *В*-системи імунітету, які мають різні функції при реалізації імунної відповіді.

Лімфоцити обох різновидів мають на своїй поверхні рецептори – макромолекулярні структури, за допомогою яких імунокомпетентні клітини розпізнають антигени. Клітинні рецептори (як і антиген) володіють специфічністю, тобто здатністю розпізнати антиген конкретного виду. Наприклад, якщо клітина розпізнає специфічний антиген *А,* то вона не може розпізнати інший специфічний антиген *В.* Всі рецептори однієї клітини володіють однаковою специфічністю.

Процес розпізнавання чужорідних субстанцій полягає у взаємодії клітинних рецепторів з рецепторами антигена (антигенними детермінантами), які, по суті справи, визначають структурну специфічність антигена. Якщо структури клітинних рецепторів і антигенних детермінант компліментарні, тобто "підходять" один до одного, як ключ до замку, то утворюється комплекс "рецептор – детермінанта", який може залишитися на поверхні клітини або відірватися від неї. У будь-якому випадку антиген знаний і блокований. Надалі розвивається та або інша форма імунної реакції.

У крові циркулює 10^{10} лімфоцитів, унаслідок чого імунна система має інформацію про всі антигени, які можуть потрапити в організм. Охоронцями такої інформації є лімфоцити, а носіями – їх клітинні рецептори. Будь-який ангиген, потрапляючи в кровострум організму, має певну вірогідність зустрічі з лімфоцитом, який запрограмовано його розпізнає, і таке розпізнавання спричиняє розвиток імунної реакції.

Антитілами (імуноглобулінами) називаються білки, що виробляються клітинами лімфоїдних органів у відповідь на проникнення в організм антигена. Антитіла володіють здатністю специфічно взаємодіяти з антигеном, внаслідок чого утворюються імунні комплекси "антиген-антитіло", які згодом виводяться з організму.

Процес утворення антитіл (антитілогенез) пояснюється таким чином. У організмі завжди присутня мала популяція імуноцитів, специфічних до даного антигена. При зустрічі антигена з імуноцитом на поверхні клітини відбувається взаємодія між антигенними детермінантами і клітинними рецепторами, у результаті якої імуноцит стимулюється, тобто отримує

здатність до диференціювання (зміни своїх властивостей і якостей) і розподілу. Після 8–9 розподілів утворюється численна популяція (клон) клітин, які більше не діляться. Частина таких клітин (плазматичні клітини) виробляє антитіла тієї ж специфічності, що і у клітинних рецепторів лімфоцита-попередника, із швидкістю $2 \cdot 10^3$ молекул за секунду протягом декількох днів, а потім гине. Інша частина клітин (клітини пам'яті) виробляє антитіла менш інтенсивно, основна їх функція – зберігати інформацію про антиген. тобто клітини пам'яті – це, по суті, нові імуноцити, які можуть знову взаємодіяти з антигеном і викликати описану каскадну реакцію. Клітини пам'яті живуть в організмі протягом декількох років. Описана імунна відповідь називається первинною, оскільки передбачалося, що антиген потрапив в організм вперше.

Вторинна реакція виникає при повторному попаданні в організм того ж самого антигена і кваліфікується як швидша і могутніша імунна відповідь. Річ у тому, що в ході первинної реакції значно поповнюється популяція імуноцитів за рахунок клітин пам'яті. Це збільшує вірогідність зустрічі антигенів і імуноцитів і залучає до реакції більше число імуноцитів: у результаті через каскадний процес значно збільшується число плазматичних клітин і антитіл у порівнянні з первинною відповіддю. У цьому і виявляється дія імунологічної пам'яті.

Отже, ми торкнулися основних понять і явищ імунології. Цього досить, щоб зрозуміти схему роботи імунної системи, яка вибрана за основу при математичному моделюванні.

Модель імунної відповіді організму на вторгнення вірусів. Розглядаючи співвідношення балансу між "народженням" і "загибеллю" кожного компонента процесу на малому інтервалі часу, подібно до того як ми розглядали конкуренцію різних популяцій для моделювання їх взаємодії, можна записати систему рівнянь [21]

$$\frac{dV}{dt} = (\beta - \gamma F)V; \tag{2.79}$$

$$\frac{dF}{dt} = \rho C - \eta\gamma FV - \mu_F F; \tag{2.80}$$

$$\frac{dC}{dt} = \zeta(m)\alpha F(t-\tau)V(t-\tau)\vartheta(t-\tau) - \mu_c(C - C^*); \tag{2.81}$$

$$\frac{dm}{dt} = \sigma V - \mu_m m, \tag{2.82}$$

де V – концентрація вірусів (антигенів);
F – концентрація антитіл;

C – концентрація плазмоклітин;

m – функція, що відтворює ураження органа;

$\beta > 0$ – швидкість (темп) розмноження вірусів;

$\gamma > 0$ – коефіцієнт, що враховує вірогідність зустрічі вірусів з антитілами і силу їх взаємодії;

$\alpha > 0$ – коефіцієнт стимуляції імунної системи;

$\rho > 0$ – швидкість виробництва антитіл однією плазмоклітиною;

$\mu_c, \mu_F > 0$ – величини, обернені до значень тривалості життя плазмоклітин і антитіл відповідно;

$\eta > 0$ – кількість антитіл, необхідна для нейтралізації одного вірусу;

$\sigma > 0$ – темп ураження органа;

$\mu_m > 0$ – швидкість відновлення маси ураженого органа;

$C^* > 0$ – попередній рівень імунокомпетентних клітин (плазмоклітин);

$\tau > 0$ – час, необхідний для формування каскаду плазмоклітин;

$\xi(m)$ – безперервна незростаюча ненегативна функція, що враховує порушення нормальної роботи імунної системи унаслідок значного ураження органа;

$\theta(t)$ – функція Хевісайда; $\theta(t) = 0, \quad t < 0; \quad \theta(t) = 1,, t \geq 0.$

Початкові умови при $t = t^0$

$V(t^0) = V^0, \quad F(t^0) = F^0, \quad C(t^0) = C^0. \quad m(t^0) = m^0.$

Система рівнянь (2.79) – (2.82) записана аналогічно рівнянням для опису взаємодії популяцій, у даному випадку мова йде про популяції вірусів та антитіл. Розглянемо отримані рівняння.

Перше рівняння (2.79) показує приріст вірусів (або антитіл) V відповідно до стану організму, а саме: збільшення вірусів пропорційне добутку кількості вірусів на швидкість розмноження вірусів βV за вирахуванням кількості вірусів, що виводяться з організму внаслідок утворення імунних комплексів “антиген-антитіло” (добуток FV), при цьому враховується ймовірність зустрічі вірусів з антитілами і сила їх взаємодії (коефіцієнт γ) .

Рівняння (2.80) показує приріст антитіл в ході імунної відповіді на вторгнення антигенів. Збільшення антитіл пропорційне добутку кількості плазмоклітин на швидкість виробництва антитіл однією плазмоклітиною ρC за вирахуванням кількості антитіл, що увійшли в імунні комплекси “антиген-антитіло” для виведення антигенів з організму (тут також враховується коефіцієнт γ) та кількості антитіл, що вибувають з причини старіння ($\mu_F F$).

Рівняння, що описує приріст плазмоклітин (2.81) враховує такі чинники:

перший член правої частини описує генерацію плазмоклітин пропорційно розпізнаванню антитілами вірусів ($F(t-\tau)V(t-\tau)$), що відбувається за деякий проміжок часу τ. Цей добуток треба помножити на функцію $\xi(m)$, що враховує пошкодження органа, та на коефіцієнт α, що враховує стимуляцію імунної системи. Другий член цього рівняння описує зменшення плазмоклітин за рахунок їх старіння (коефіцієнт μ_c дорівнює обрненій величині їхнього часу життя).

Останнє рівняння (2.82) системи відтворює динаміку ураження органа внаслідок хвороби. Якщо M – характеристика здорового органа (маса або площа), а M' – відповідна характеристика здорової частини ураженого органа, то функція m є

$$m = 1 - M' / M \ . \qquad (2.5.5)$$

Тобто m є відносною характеристикою ураження органа мішені. Для неураженого органа вона дорівнює нулю, а для органа, що є повністю ураженим, – одиниці. В правій частині рівняння (2.82) перший член характеризує ступінь ураження органа, ця величина пропорційна до кількості антигенів, коефіцієнт σ – деяка константа, що характеризує захворювання. Зменшення цієї характеристики відбувається за рахунок відтворення діяльності організму, коефіцієнт μ_m характеризує таке відтворення.

Математична модель (2.79) – (2.82) описує імунну відповідь організму на вторгнення антитіл відповідно до наступної класіфікації: субклінічна форма, гостра форма з одужанням, гостра форма з летальним результатом, хронічна форма. Проявлення цих форм є такими.

Субклінічна форма зазвичай протікає приховано і не пов'язана з фізіологічними розладами організму. Це звичайний контакт організму з вже відомим антигеном, і організм на даний момент має достатньо ресурсів для його подавлення. В цьому випадку антиген знищується без досягнення концентрації, що викликає помітну імунну та фізіологічну реакцію організму. Крива, що відображає концентрацію вірусів, від деякого початкового значення спадає до нуля. Оскільки організм у своєму повсякденному житті контактує з багатьма антигенами, то зазвичай імунний процес боротьби з ними перебігає субклінічно. В цьому полягає одна з чудових особливостей імунної системи.

Гостра форма захворювання виникає, якщо антиген, який проникає в організм, є невідомим. Процес його розпізнавання і формування плазмоклітин супроводжується збільшенням концентрації антигена за рахунок розмноження. В цьому випадку ми маємо справу з нормальною гострою формою захворювання. На графіку концентрації вірусів їх кількість спочатку зростає, а потім, досягнув максимального значення, спадає до нуля.

Це класична форма протікання захворювання з підвищенням температури, інтоксикацією організму і таке інше, після чого настає полегшення і одужання. Але гостра форма захворювання може мати і летальний результат. Якщо імунна відповідь є пізньою, в органі спостерігаються значні пошкодження, і уражений орган вже не забезпечує нормальну роботу імунної системи, то настає тяжка форма захворювання з можливим летальним результатом. На графіку концентрація антигенів зростає без спадання.

Хронічні захворювання є найбільш тяжкими і можуть тривати роками, вони є стійкими формами імунної відповіді і мають або циклічну, або не залежну від часу динаміку. У випадку хронічної форми концентрація антигенів прямує не до нуля, а до деякого значення $V>0$. В організмі встановлюється рівновага між антигенами і всіма компонентами імунної системи, тобто встановлюється стійка форма захворювання, що класіфікується як хронічна форма захворювання.

Всі описані форми захворювань можна простежити, аналізуючи розв'язання системи (2.79) – (2.82). Для цього можна використати або якісний аналіз системи диференціальних рівнянь, або чисельне розв'язання цієї системи.

Розглянемо стаціонарні розв'язки системи. Для їх знаходження прирівняємо всі похідні до нуля. Тоді отримаємо

$$(\beta - \gamma F)V = 0\,; \tag{2.83}$$

$$\xi(m)\alpha VF - \mu_c(C - C^*) = 0\,; \tag{2.84}$$

$$\rho C - F\mu_F + \eta\gamma VF = 0\,; \tag{2.85}$$

$$\sigma V - \mu_m m = 0\,. \tag{2.86}$$

У рівняннях (2.83) – (2.86) V і F не залежать від часу, тому V= const і F= const. В здоровому організмі, де антигени відсутні, з рівняння (2.85) випливає співвідношення

$$C^* = \frac{\mu_F F^*}{\rho}\,. \tag{2.87}$$

Зірочками помічені значення для здорового організму при V=0. Неважко побачити, що одним з тривіальних розв'язків, що описує стан здорового організму, є такий:

$$m = 0, \quad V = 0, \quad C = C^{,*} \quad F = F^* = \frac{\rho C^*}{\mu_F} \tag{2.88}$$

Можна показати, що цей стан є асимптотично стійким при $\beta<\gamma F^*$. Для цього розглянемо малі збурення невідомих функцій від стану рівноваги,

прийнявши

$$m = m', \quad V = V', \quad C = C^* + C', \quad F = F^* + F'. \tag{2.89}$$

Підставляючи ці вирази в систему (2.79) – (2.82) та відкинувши величини другого порядку малості, отримаємо розв'язок першого з рівнянь у вигляді

$$V' = \varepsilon_1 e^{(\beta - \gamma F^*)t}. \tag{2.90}$$

Для того, щоб розв'язок (2.90) при $t \to \infty$ прямував до нуля, необхідно і достатньо, щоб виконувалась нерівність $\beta < \gamma F^*$.

Аналогічно і для всіх інших рівнянь малі збурення стаціонарних розв'язків зі збігом часу прямують до нуля при $\beta < \gamma F^*$, тобто є асимптотично стійкими. Така поведінка системи відповідає зараженню здорового організму малою дозою антигена $V^0 = \varepsilon$. Аналізуючи розв'язки системи рівнянь (2.83) – (2.86) можна оцінити "малість" дози зараження, яка не приводить до втрати стійкості. Така доза задовольняє нерівності

$$0 < V^0 < \frac{\mu_F(\gamma F^* - \beta)}{\beta\eta\gamma} = V^*.$$

Величину V^* називають імунологічним бар'єром. Кажуть, що імунологічний бар'єр перетнуто, якщо доза зараження V^0 задовольняє нерівності $V^0 > V^*$, у протилежному випадку кажуть, що імунологічний бар'єр не перетнуто. Такий підхід має біологічну інтерпретацію. Якщо при зараженні організму малою дозою антигенів імунологічний бар'єр не можна перетнути, то незалежно від дози зараження розвиток хвороби не відбувається, тобто кількість антигенів в організмі з плином часу зменшується, прямує до нуля, а уражений орган відновлюється. Крім того, підвищення C^* – рівня імунокомпетентних клітин в здоровому організмі (наприклад, за рахунок клітин пам'яті при вакцинації), підвищує імунологічний бар'єр (внаслідок того, що $F^* = \rho C^*/\mu_F$) і тому є ефективним методом профілактики, а можливо і лікування хвороби.

Аналіз моделі захворювання (2.79) – (2.82) дозволяє судити про якісну поведінку розв'язку $V(t)$ – концентрації антигенів при тому або іншому наборі коефіцієнтів. Розглянемо два граничні випадки, які, по суті справи, є межами для розв'язку $V(t)$.

Припустимо, що організм не виробляє антитіл даної специфічності, тобто $F(t) = F^0 = 0$ для всіх $t \geq 0$ і $\rho = 0$. В цьому випадку рівняння для $V(t)$ має вигляд

$$\frac{dV}{dt} = \beta V \ . \tag{2.91}$$

Розв'язок цього рівняння дається формулою

$$V(t) = V^0 e^{\beta t} \ , \tag{2.92}$$

де V^0 – початкова концентрації антигенів (доза зараження) у момент часу t=0. Що стосується динаміки ураження органу, то неважко зрозуміти, що за відсутності відновних процесів в ураженому органі, тобто при μ_m=0,

$$m = \frac{\sigma V^0}{\beta}(e^{\beta t} - 1) \ . \tag{2.93}$$

і при всіх $t \geq 0$

$$V = V^0 e^{\beta t}, \quad F = 0, \quad m = \frac{\sigma V^0}{\beta}(e^{\beta t} - 1) \ . \tag{2.94}$$

Мабуть, такий розв'язок відповідає перебігу хвороби з летальним результатом, оскільки ніяких чинників, що компенсують зростання антигенів, немає.

Розглянутий випадок можна розглядати як граничний. На практиці такі випадки украй рідкісні. Проте іноді відповідь імунної системи на антиген виявляється такою слабкою, що описаний тут ідеальний випадок є для них хорошим наближенням. Така ситуація, наприклад, виникає у деяких людей похилого віку, імунна система яких не має вираженої реакції проти антигену, або у людей з придбаними або природженими імунними дефектами.

Другий граничний випадок реалізується при сильно вираженій імунній відповіді, коли рівень присутніх в організмі антитіл, специфічних до даного антигена, виявляється достатнім для того, щоб знищити всі антигени, що проникли в організм, не включаючи в дію механізм антитілоутворення. В цьому випадку рівняння для $V(t)$ має вигляд

$$\frac{dV}{dt} = (\beta - \gamma F)V \ , \tag{2.95}$$

де $\beta \ll \gamma F$**.** Припускаючи дозу зараження V^0 малою, можна вважати F величиною постійною, визначуваною нормальним рівнем антитіл F^*. Тоді наведене вище рівняння перепишеться таким чином:

$$\frac{dV}{dt} = (\beta - \gamma F^*)V\,, \tag{2.96}$$

і його розв'язок матиме вигляд

$$V = V^0 e^{-(\gamma F^* - \beta)t}\,. \tag{2.97}$$

Це означає, що популяція антигенів в організмі експоненціально зменшуватиметься. У граничному випадку β=0 одержимо

$$V = V^0 e^{-\gamma F^* t}\,. \tag{2.98}$$

Отже, ми знайшли два граничні розв'язки моделі, що відповідають летальному результату і високому імунологічному бар'єру.

Моделювання введення лікарських засобів. Реакцію організму на вторгнення вірусів можна дослідити за допомогою системи диференціальних рівнянь (2.79) – (2.82). У цій моделі передбачається, що основними чинниками вірусного захворювання є: кількість вірусів $V(t)$; число плазмових клітин $C(t)$; з яких виробляються антитіла $F(t)$; маса тканини $m(t)$, що є пошкодженою. Константи і функція $\xi(m)$, що входять в рівняння, характеризують конкретний організм. Управління параметрами системи (2.79) – (2.82) можна розглядати як процес лікування, пов'язаний з переходом з однієї форми хвороби в іншу. Такий перехід досягається за допомогою ліків і є процесом лікування. Наприклад, це перехід з гострої форми в субклінічну або з хронічної форми в гостру, а потім в субклінічну.

Для отримання різних форм захворювання на моделі використовуються результати розв'язку оберненої задачі, тобто знаходження таких коефіцієнтів системи (2.79) – (2.82), при яких досягається мінімальна відмінність між обчислюваними значеннями змінних $V(t)$, $C(t)$, $F(t)$, $m(t)$ і заданими значеннями $V_1(t)$, $C_1(t)$, $F_1(t)$, $m_1(t)$. Величини $V(t)$, $C(t)$, $F(t)$, $m(t)$ обчислюються прямим чисельним інтегруванням. В якості критерія оптимізації використовується мінімаксний критерій, тобто мінімум максимального відхилення обчислюваних залежностей $V(t)$, $C(t)$, $F(t)$, $m(t)$ від заданих $V_1(t)$, $C_1(t)$, $F_1(t)$, $m_1(t)$ в конкретні моменти часу.

У розширеній моделі вводиться нова змінна S і до системи (2.79) – (2.82) додається нове диференціальне рівняння

$$\frac{dS}{dt} = qS(t - \tau_1) - \mu_t(t - \tau_2)\,, \tag{2.99}$$

де S – узагальнений кількісний сумарний критерій несприятливої дії на органи або біосистеми (нервова, ендокринна, кровотворна та ін.) організму

від введення ліків; q – коефіцієнт, що характеризує ступінь несприятливої дії на організм в одиницю часу; τ_1– час затримки вказаної несприятливої дії; μ_t – коефіцієнт, що характеризує відновлення ураженої біосистеми в одиницю часу. Передбачається, що механізм відновлення починає діяти із затримкою $\tau_2 > \tau_1$.

Функція *S(t)* може бути пов'язана з дією на будь-який параметр системи. Тут ми як приклад розглянемо дію на параметр γ. Припустимо, що дія ліків така, що в моделі коефіцієнт γ, пов'язаний з вірогідністю нейтралізації вірусу антитілом, змінюється з часом пропорційно введеній змінній S і після досягнення часу $t = \tau_3$ залишається постійним. Таким чином, $\gamma = \gamma (1 + C_{\text{еф}} S_{\tau 3})$, де $S_{\tau 3} = S(t)$, якщо $t \le \tau_3$, і $S_{\tau 3} = S(\tau_3)$, якщо $t > \tau_3$; $C_{\text{еф}}$ – коефіцієнт ефективності дії ліків.

Отже, замінюючи в системі диференціальних рівнянь коефіцієнт γ і добавляючи в модель вірусного захворювання рівняння для узагальненого критерію несприятливої дії, отримаємо розширену модель вірусного захворювання:

$$\frac{dV}{dt} = (\beta - \gamma(1 + C_{еф} S_{\tau 3})F)V\,; \tag{2.100}$$

$$\frac{dC}{dt} = \zeta(m)(\alpha + \rho F(t-\tau)V(t-\tau) - \mu_C(C - C^*)\,; \tag{2.101}$$

$$\frac{dF}{dt} = \rho C - (\mu_t + \eta\gamma(1 + C_{еф} S_{\tau 3})V)F\,; \tag{2.102}$$

$$\frac{dm}{dt} = \sigma V - \mu_m m\,; \tag{2.103}$$

$$\frac{dS}{dt} = qS(t-\tau_1) - \mu_t(t-\tau_2)\,. \tag{2.104}$$

Помітимо, що

$$S_{\tau 3} = \begin{cases} S(t), & \text{якщо} \quad t \le \tau_3 \\ S(\tau_3), & \text{якщо} \quad t > \tau_3 \end{cases}. \tag{2.105}$$

Система (2.100) – (2.104) дозволяє моделювати дію на біосистему (введення ліків або застосування іншого типу лікування), яка, з одного боку, сприяє лікуванню вірусного захворювання, але з іншого боку може несприятливо впливати на нього.

2.6.3. Моделювання серцево-судинної системи організму

Опис роботи серцево-судинної системи й основні параметри. Модель

Франка. Серце – це елемент системи кровообігу, що є складною системою, яка забезпечує транспорт кисню, вуглекислого газу, живильних речовин, метаболітів, гормонів, імуноцитів і води, перенесення тепла, розповсюдження коливальних рухів і т. ін. Кров забезпечує живильними речовинами і киснем клітини організму, переносить вуглекислий газ у зворотному напрямі. Серце в цій системі виконує насосну функцію.

У вирішенні проблеми боротьби з серцево-судинними захворюваннями особливе місце займає розробка методів оцінки механічних скоротливих властивостей серця. Така оцінка важлива і при діагностиці захворювань, і при визначенні ефективності лікувальних заходів, що робляться для відновлення серцевої діяльності за допомогою фармакологічних препаратів або технічних засобів. Конструктивним підходом до вирішення проблеми оцінки скоротливих властивостей серця може служити розробка строгих математичних моделей механіки серцевого м'яза і цілісного серця.

Такі моделі повинні задовольняти ряд вимог. Найважливіша з них полягає в необхідності побудови простих моделей з мінімальним числом параметрів. Ця вимога диктується істотною неповнотою і, як правило, поганою якістю клінічних і експериментальних даних. Проте моделі повинні, не дивлячись на простоту, достатньо адекватно відображати істотні риси механіки скорочення. Важливою вимогою є можливість тлумачення (формалізація) скоротливих властивостей в термінах даної моделі.

Початок моделюванню в гемодинаміці поклав німецький фізіолог О. Франк [53]. Гемодинаміка вивчає закони руху крові по кровоносних судинах. До основних гемодиамічних показників відноситься тиск і швидкість кровоструму. Тиск P – це сила, що діє з боку крові на судини відносно одиниці площі. Об'ємною швидкістю кровоструму Q називають об'єм рідини, що протікає в одиницю часу через даний переріз судини.

Вперше модель судинної системи, де серце служило насосом, що прокачує кров по судинах, запропонував в 1628 році англійський лікар В. Гарвей.

Основна функція серцево-судинної системи в цілому – забезпечення безперервного руху крові по капілярах з постійною швидкістю кровоструму і перепадом тиску (оскільки саме в капілярах відбувається обмін речовин між кров'ю і тканинами). Серцево-судинна система замкнута, тому для забезпечення руху крові в ній повинен бути періодично діючий насос. Цю роль виконує серце. Періодичне надходження крові з серця перетворюється на постійне надходження її в дрібні судини за допомогою крупних судин: частина крові, що виштовхується з серця під час *систоли*, резервується в крупних судинах завдяки їх еластичності, а потім під час *діастоли* надходить в дрібні судини. Тим самим крупні судини є погоджувальним елементом між серцем і дрібними судинами.

У 1899 р. О. Франк теоретично розвинув ідею про те, що артерії “запасають” кров під час систоли і виштовхують її в дрібні судини під час діастоли. Він поставив мету: розрахувати зміну гемодинамічних показників (наприклад тиску) в часі в деякій точці *x* крупної судини.

Для зручності розгляду Франк виділив дві фази кровоструму в системі “лівий шлуночок серця – крупні судини – дрібні судини*”*.

1 фаза. Це фаза притоку крові в аорту з серця з моменту відкриття аортального клапана до його закриття.

2 фаза. Це фаза вигнання крові з крупних судин в дрібні після закриття аортального клапана.

У моделі Франка зроблені такі припущення:

1. Всі крупні судини об’єднані в один резервуар з еластичними стінками, об’єм якого пропорційний тиску. Вони (а отже, і резервуар) мають високу еластичність*;* гідравлічним опором резервуара нехтують.

2. Система мікросудин предсгавлене як жорстка трубка. Гідравлічний опір жорсткої трубки великий*;* еластичностю дрібних судин нехтують.

3. Еластичність і опір для кожної групи судин постійні в часі і у просторі.

4. Не розглядаються перехідні процеси встановлення руху крові.

5. Існує "зовнішній механізм" закриття і відкриття аортального клапана, що визначається активною діяльністю серця.

Під час руху крові з серця частина її розміщується в крупних судинах, розтягуючи їх, а частина крові протікає в дрібні судини. Звідси можна записати рівняння балансу об’єму крові:

$$Q_c dt = Q dt + dV, \tag{2.106}$$

де $Q_c(t)$ – об’ємна швидкість надходження крові з серця; $Q(t)$ – об’ємна швидкість кровоструму на початку дрібних судин; dV – зміна об’єму крупних судин.

Припускаємо, що зміна об’єму резервуара лінійно залежить від зміни тиску в ньому dP:

$$dV = C dP, \tag{2.107}$$

де C – еластичність, коефіцієнт пропорційності між тиском і об’ємом.

Умовно вважаючи, що тиск на виході жорсткої трубки дорівнює нулю, за законом Пуазейля, який зв’язує падіння тиску в циліндричній трубці з об’ємним кровострумом, одержимо

$$Q = \frac{P}{W}. \tag{2.108}$$

Систему рівнянь (2.106) – (2.108) можна розв’язати щодо $P(t)$, $Q(t)$ або

V(t). Розв'яжемо систему відносно *P(t)*.

1 фаза. З урахуванням виразів (2.106) – (2.108) одержимо рівняння

$$\frac{dP}{dt}+\frac{P}{WC}=\frac{Q_c}{C}. \qquad (2.109)$$

Це неоднорідне лінійне диференціальне рівняння, розв'язання якого визначається видом функції $Q_c(t)$.

Вважатимемо функцію Q_c такою, що дорівнює її середньому значенню Q_0 за час між відкриттям і закриттям аортального клапана. Тоді рівняння (2.109) запишеться так:

$$\frac{dP}{dt}+\frac{P}{WC}=\frac{Q_0}{C}. \qquad (2.110)$$

Вважаючи, що при t=0 тиск $P = P_Д$ і інтегруючи рівняння (2.110), одержимо закон зміни тиску в крупних судинах

$$P(t)=Q_0W(Q_0W-P_Д)e^{-\frac{t}{WC}}. \qquad (2.111)$$

Таким чином, через час систоли t_c спрацьовує механізм закриття аортального клапана, при цьому тиск крові в крупних судинах досягає деякого значення P_c:

$$P_c=Q_0W(Q_0W-P_Д)e^{-\frac{t_c}{WC}}. \qquad (2.112)$$

2 фаза. Друга фаза починається з моменту закриття аортального клапана. Саме цей момент вважається початковим для фази 2. Модель Франка дозволяє аналітично виразити падіння тиску *P(t)* в крупній судині після закриття аортального клапана.

Оскільки кров вже не вищтовхується з серця, то Q_c =0 . Тоді рівняння (2.106) набуває такого вигляду:

$$-Qdt=dV. \qquad (2.113)$$

Знак "–" відображає зменшення об'єму крупної судини з часом. З урахуванням виразу (2.107) одержимо

$$Q=-C\frac{dP}{dt}. \qquad (2.114)$$

Замість Q підставимо вираз з формули (2.108) і одержимо диференціальне рівняння

$$-C\frac{dP}{dt}=\frac{P}{W}. \qquad (2.115)$$

Початкова умова при t=0 (відповідає моменту закриття клапана) $P = P_c$. У результаті одержуємо закон зміни тиску в крупних судинах з моменту закриття аортального клапана:

$$P(t)=P_c e^{-\frac{t}{WC}}. \qquad (2.116)$$

Рівняння (2.116) якісно описує зміну *P(t)* в аорті і відповідає такій же залежності, одержаній експериментально.

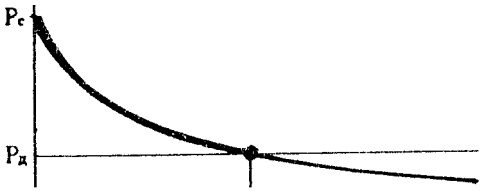


Рис. 2.18 Залежність тиску крові в крупних судинах після закриття аортального клапана

Математичні моделі дали принципово нову можливість вивчення та імітації серцево-судинної системи.

Електричні аналогії в моделях серцево-судинної системи. Моделі судинного русла. Модель В.А. Ліщука. Диференціальне рівняння моделі Франка можна одержати, використовуючи електричні аналогії (у електричному аналогу еластичності відповідає ємНість конденсатора). Для кровоносної системи справедливі ті ж самі закони, що відомі для схем електричних: це і закон Ома, і закони Кирхгофа. Тому для моделювання кровоносної системи можна використовувати електричні кола (без урахування перехідних процесів). В цьому випадку рух крові по судинах моделюватиметься електричним струмом в колі з активних опорів. Системи диференціальних рівнянь для кровоносної системи і електричного кола записуються однаково, а для параметрів можна використовувати табл. 2.4 з еквівалентними величинами в тому і іншому випадках.

Таблиця 2.4 - Еквівалентні величини в електричному колі та серцево-судинній системі

Електричне коло	Кровоносна система
Сила струму у всьому колі I_0	Об'ємна швидкість кровоструму у всій системі Q_0
Падіння напруги U на опорі	Падіння тиску P уздовж судини
Опір R	Гідравлічний опір ділянки W

Для опису роботи шлуночків серця і кровоносної системи в цілому використовуються складніші моделі, які аналітично не розв'язуються, але комп'ютер дає можливість використовувати чисельні розв'язки таких систем.

Електрична модель (рис. 2.19), побудована на основі ідеї Франка, являє собою електричну ємність (конденсатор) зі значенням C і паралельно підключений омічний опір зі значенням R. Ємність відповідає еластичності резервуара, а електричний опір – гідравлічному опору периферичних судин.

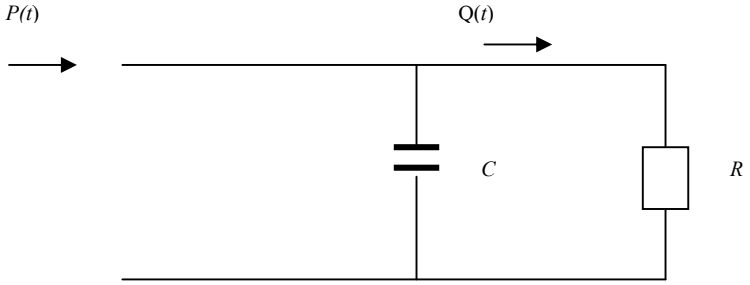


Рис. 2.19 Електрична модель прямої аналогії відповідно до моделі Франка.

В подальшому Ростон у 1962 році розглядав замість одного резервуара два, відповідно дві камери, що моделюють висхідну і низхідну гілку аорти. Електрична схема такої моделі представлена на рис. 2.20.

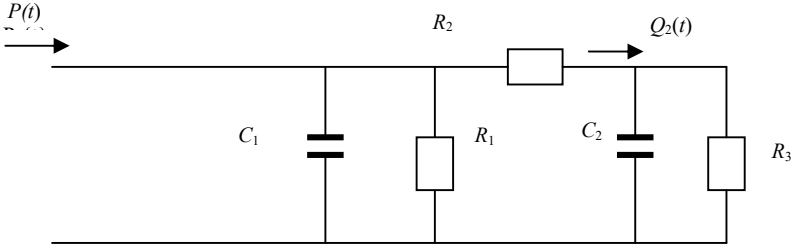


Рис. 2.20 Електрична модель судинного русла Ростона

Поведінка такої моделі описується системою з двох диференціальних рівнянь першого порядку. Кожне з цих рівнянь аналогічна рівнянню, що описує еластичний резервуар моделі Франка, але відрізняється наявністю правої частини, що описує, як надходить кров до цього резервуара.

Запишемо систему цих рівнянь відповідно до електричної аналогії (див. табл. 2.4).

З першого закону Кірхгофа маємо

$$-U_{c_1} + R_1(i_1 - i_2) = 0\,; \tag{2.117}$$

$$R_1(i_2 - i_1) + R_2 i_2 + U_{c_2} = 0\,; \tag{2.118}$$

$$-U_{c_2} + R_3 i_3 = 0\,. \tag{2.119}$$

Ще два диференціальних рівняння отримуємо із співвідношення між напругою і струмом на конденсаторі:

$$\frac{dU_{c_1}}{dt} = -\frac{i_1}{c_1}\,; \tag{2.120}$$

$$\frac{dU_{c_2}}{dt} = \frac{i_2 - I_3}{c_2}\,. \tag{2.121}$$

Система диференціальних рівнянь (2.120) – (2.121) розв'язується сумісно з алгебраїчними рівняннями (2.117) – (2.119) чисельними методами. У простих моделях неможливо дати детальний опис судинної системи з її анатомічною структурою. Такі моделі використовують тільки для аналізу гемодинамічних процесів в крупних судинах. Для моделювання структури, більше або менше наближеної до структури судинної системи, використовують електричні схеми з великим числом *RLC* електричних контурів. Описані системи з 600 елементами. Значення параметрів контура *R*, *L*, і *C* залежить від внутрішнього діаметра судини, товщини її стінки, модуля її пружності, а також від довжини ділянок судини, що моделюється, в яких відокремлюється узагальнена судинна ділянка. Такі системи називаються системами з зосередженими параметрами.

Узагальнена судинна ділянка розглядається як окрема судина або система судин деякої ділянки тканини або органа, які за умовою задачі виділені як відносно самостійна область і які характеризуються усередненими показниками кровоструму, тиску, опору, еластичності і таке інше. Найістотнішими параметрами такої ділянки є об'ємна швидкість кровоструму $q_{ij}(t)$, i=1...n; j = 1. ..n; n – число виділених ділянок, об'єм крові $V_i(t)$, а також підтримка необхідного рівня кров'яного тиску $P_i(t)$. Дослідження судин і судинних ділянок методом зосереджених параметрів ефективне тоді, коли розглядається серцево-судинна система в цілому (або достатньо великі судинні ділянки).

Виконуючи опис в зосереджених параметрах, припускають, що тканини розглянутої ділянки судинної системи мають деяку усереднену еластичність і інерційність *L*.

Подальший виклад вестиметься стосовно *i*-ї узагальненої ділянки, що дозволяє розглядати її безпосередньо як частину всієї системи, що в свою чергу дозволить перейти до моделювання судинного русла в цілому.

Кожна виділена область серцево-судинної системи (ділянка, судина, порожнина) містить деякий об'єм крові $V_i(t)$. Величина $V_i(t)$ визначається в загальному випадку потоками $q_{ji}(t)$ зі всіх ділянок до тієї, що розглядається:

$$V_i(t) = V_i(0) + \int_0^t \sum_{j=1}^{n} q_{ji}(\tau) d\tau, \; j \neq i \,, \tag{2.122}$$

де V_i (0) – об'єм крові на початку дослідження (t = 0). Напрямок потоку з j-го елемента до i-го прийнятий позитивним, що відображене в послідовності індексів ji. Іншими словами, позитивний потік поповнює i-ту ділянку.

Диференціюючи вираз для $V_i(t)$, одержимо рівняння для швидкості зміни об'єму

$$\frac{dV}{dt} = \sum_{j=1}^{n} q_{ji}(t), \; j \neq 0 \,. \tag{2.123}$$

Допускаючи певні обмеження, можна прийняти наступне положення: чим більше крові в судинній ділянці, тим сильніше розтягнуті її тканини (стінки судини) і відповідно тим більшийе тиск в ділянці (і його потенційна енергія).

У лінійному наближенні для повільних процесів дана залежність звичайно описується виразом

$$P_i(t) = e_i[V_i(t) - U_i] \,, \tag{2.124}$$

де e_i – середня (по ділянці) жорсткість, а U_i – об'єм крові, що розпрямляє, але не розтягує ділянку судинного ложа. Оскільки e – величина, обернена до еластичності ($e = c^{-1}$), то можна говорити, що вона характеризує еластичні властивості судини. У загальному випадку залежність тиску від об'єму нелінійна. У зв'язку з цим правильніше фіксувати жорсткість судинної ділянки при деякому певному наповненні $e=\partial P/\partial V$.

Розпрямляючий об'єм крові (U) складає ту найбільшу частину повного об'єму заповнення судинної ділянки, яка сама по собі ще не розтягує стінки судини. Наповнюючи яку-небудь судину або судинну ділянку, кров спочатку розпрямляє її і лише потім, коли об'єм стане достатнім, розтягує.

З метою побудови досить загального математичного опису істотну роль виконують такі відношення, при яких опір потоку крові залежить від його різниці тиску. Припускаючи, що введені відношення при побудові моделі завжди визначені (задаються всі p_{ij}), запишемо вираз для потоку між j-ю і i-ю ділянками:

$$q_{ji}(t) = \rho_{ji} P_j(t) - \rho_{ij} P_i(t)], \tag{2.125}$$

де ρ_{ii} – провідність відповідної ділянки.

У крупних артеріях і венах при розгляді пульсуючого потоку і швидких динамічних реакцій бажано враховувати інерційність L_{ji} потоку

$$\frac{dq_{ji}}{dt} + r_{ji} L_{ji}^{-1} q_{ji}(t) = L_{ji}^{-1}[P_j(t) - P_i(t)]. \tag{2.126}$$

Інерційність розглядається як специфічна властивість. Інерційні властивості потоку визначаються прискореннями маси крові, стінок судин і тканин. Частина енергії, пов'язана з подоланням інерційних сил потоку крові, витрачається на транспорт крові. Навпаки, та частина, яка пов'язана з подоланням інерційностей тканин, для потоку крові втрачається. Істотні інерційні властивості артеріальної системи (крупних судин), де йде накопичення енергії за рахунок інерції при зміні режиму кровообігу (або в різних фазах пульсації) може приводити до коливальних режимів.

Розглянуті властивості і відношення є загальними для багатьох модельованих підсистем кровообігу. Тому вважатимемо, що одержані співвідношення (2.122) – (2.125) є моделлю узагальненої судинної ділянки. Деякі перетворення допоможуть звести згадані рівняння до компактного і зручного вигляду. Визначимо тиск і кровострум за допомогою об'ємів, підставивши вираз (2.124) в (2.122):

$$V_i(t) = \sum_{j=1}^{n} \rho_{ji}(y) P_j(t) - \sum_{j=1}^{n} \rho_{ij}(y) P_i(t) + q_{01}(t), j \neq i. \tag{2.127}$$

Дозволимо рівність $j = i$, припустивши, що

$$\rho_{ii}(t) = -\sum_{k=1}^{n} \rho_{ik}(y), k \neq i (k \neq j). \tag{2.128}$$

Одержимо

$$\frac{dV_i}{dt} = \sum_{j=1}^{n} \rho_{ji}(y) P_j(t) + q_{0i}(t). \tag{2.129}$$

При з'єднанні ділянок i і j декількома судинами величина ρ_{ii} відображає сумарну провідність

$$\rho_{ii}(y) = -\sum_{k=1}^{r} \rho_{ik}{}^{k}(y). \tag{2.130}$$

де r – число каналів між j-м і i-м ділянками, $\rho_{ii}^{\ k}$ – провідність k-го каналу.

Тепер від однієї ділянки треба перейти до моделі системи судинних ділянок. Для цього припустимо, що в рівняннях для однієї ділянки i=1,2,....n. Запишемо n відповідних рівнянь. Таку систему зручно зобразити в матричному вигляді:

$$\frac{d\overline{V}}{dt} = A\overline{V} + B, \tag{2.131}$$

де V – вектор-стовпчик, що характеризує об'єми судинних ділянок:

$$\overline{V} = \begin{vmatrix} V_1 \\ V_2 \\ \cdots \\ V_n \end{vmatrix}, \tag{2.132}$$

A – матриця коефіцієнтів, що визначає зв'язки між ділянками:

$$A = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ & & \cdots & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix}, \tag{2.133}$$

B – вектор-стовпчик вільних членів.

Таким чином, можна перейти до системи судинних ділянок. Ця модель описана в роботі [22] В.А. Ліщуком. Суттєвим моментом при побудові моделей судинної системи є вибір загального числа ділянок, збільшення яких дає зростання точності моделі, але водночас підвищує складність моделі.

Моделі серця. Моделювання автохвильових процесів у серцевому м'язі. Другим важливим елементом моделі серцево-судинної системи є модель серця. Вивчення процесів, що відбуваються в фазу систоли в судинній системі неможливе без побудови моделі серця, бо ці процеси є результатом взаємодії серця і судинної системи.

Ростон в 1959 році при дослідженні еластичного резервуара Франка використав найпростішу модель лівого шлуночка у вигляді джерела імпульсів об'ємної швидкості струму крові. Ці імпульси в часі мають сінусоїдальну форму, що була обрана відповідно до експериментальних даних у першоиу наближенні. Відповідні диференціальні рівняння мають вигляд

$$C\frac{dP}{dt}+\frac{P}{R}=Q(t),\quad Q(t)=A\sin\frac{\pi t}{t_c}. \tag{2.134}$$

Подальші дослідження показали, що такий підхід дає результати, не достатньо близькі до фізіологічних.

На наступному кроці Де Патер і Ван дер Берг створили свою модель серця. Кожен з відділів серця був представлений як джерело електричних імпульсів напруги, що мають певну форму в часі. Послідовно з джерелом була підключена електрична ємність. Для моделювання роботи клапанів серця були застосовані діоди. Форми кривих тиску і кровоструму, одержаних за допомогою цих моделей, були достатньо близькими до фізіологічних. Також в якості моделі шлуночків серця використовували змінну електричну ємність.

Ми розглянули моделі, що описують процеси в серцево-судинній системі в цілому, тобто такі моделі, що відображають процеси кровообігу в судинній системі, включаючи роботу серця, що діє в цій системі як насос.

Але в серці відбуваються процеси, які відносяться до автохвильових. Ці процеси пов'язані з розповсюдженням електричного збудження по серцевому м'язу. Такі процеси зручно моделювати за допомогою аксіоматичних моделей. На відміну від попередніх моделей, що потребують для опису використання математичних рівнянь, аксіоматичні моделі застосовують опис правил (аксіом), що відповідають поведінці моделі і які можна безпосередньо представити у вигляді відповідної програми. Візуалізувати поведінку системи, що моделюється, можна за допомогою графічних засобів [22].

Опис аксіоматики моделі серця полягає в такому. Одна з характерних властивостей багатьох тканин, і в першу чергу нервової тканини, – це здатність до збудження і передачі збудження від одних ділянок до інших. Приблизно один раз за секунду хвиля збудження пробігає по серцевому м'язу, примушуючи його скорочуватися і гнати кров по всьому тілу. Імпульси збудження, розповсюджуючись по нервових волокнах від периферії (рецепторів органів чуття) до спинного і головного мозку, приносять сигнали від зовнішнього світу, а у зворотному напрямі (по інших волокнах) йдуть імпульси-команди м'язам для виконання тих або інших дій. В цілому, розповсюдження збудження – це той матеріальний процес, який в основному забезпечує передачу інформації і регулювання в живому організмі.

Збудження в нервовій клітині може виникнути або само по собі, або під впливом збудження сусідньої клітини, або, нарешті, під дією зовнішнього сигналу, скажімо, електричного струму. Перейшовши в збуджений стан, клітина залишається в ньому якийсь час, а потім збудження зникає і настає так званий рефрактерний період, протягом якого клітина не реагує на

сигнали, що надходять до неї. Потім клітина повертається в первинний стан спокою, з якого вона знову може перейти в збуджений стан.

Таким чином, процес виникнення і розповсюдження збудження має ряд чітко виражених властивостей, базуючись на від яких можна побудувати формальну модель цього явища. А потім для дослідження такої моделі можна застосовувати вже чисто математичні методи. Вперше математичний підхід до дослідження збудливих тканин був здійснений в 1946 році Н. Вінером і А. Розенблютом. Модель роботи однієї клітини має такий аксіоматичний опис.

Під «збудливим середовищем» розумітимемо деяку безліч елементів («клітин»), що мають такі властивості [22].

1) кожен елемент x множини X може перебувати в одному з трьох станів: спокій, збудження і рефрактерність;

2) стан збудження має деяку тривалість τ (різну, взагалі кажучи, для різних елементів), потім елемент переходить на якийсь час до рефрактерного стану, після чого він повертається в стан спокою;

3) від кожного збудженого елемента збудження розповсюджується з деякою швидкістю по безлічі елементів, що перебувають у стані спокої;

4) якщо елемент x не був збуджений протягом деякого певного часу $T(x)$, то після цього часу він примусово переходить в збуджений стан. Час $T(x)$ називається періодом спонтанної активності елемента. При цьому не виключається і той випадок, коли $T(x)=\infty$, тобто коли спонтанна активність відсутня.

Схожі математичні моделі розглядаються і в зовсім інших областях, наприклад в теорії горіння, в задачах про розповсюдження світла в неоднорідному середовищі і т. д. Проте наявність деякого кінцевого «періоду рефрактерності» характерна в першу чергу саме для біологічних процесів.

Описану модель можна досліджувати або аналітичними методами, або шляхом реалізації її на обчислювальній машині. У останньому випадку ми, зрозуміло, вимушені вважати, що множина X (збудливе середовище) складається з деякого скінченного числа елементів (відповідно до можливостей існуючої обчислювальної техніки – порядку декількох тисяч). Для аналітичного дослідження природно припускати, що множина X деяким безперервним середовищем, наприклад, вважати, що X – це шматок площини серцевого м'яза.

За допомогою аксіоматичних моделей можна описати явище синхронізації**.** Розглянемо середовище, де періоди спонтанної активності різних елементів неоднакові. У такому середовищі відбувається синхронізація, причому ритм збудження всього середовища визначається ритмом найшвидшого елемента, тобто елемента з якнайменшим періодом спонтанної активності. Дійсно, хай йдеться всього лише про два зв'язані між

собою елементи A_1 і A_2 з періодами спонтанної активності T_1 і T_2, причому $T_1<T_2$. Коли один з цих елементів збудиться, то він примусить збудитися і другий (ми вважаємо, що час передачі збудження до сусіднього елемента достатньо малий). Через час T_1 збудиться (спонтанно) елемент A_1 і передасть збудження елементу A_2 і т. д. Пара елементів працюватиме синхронно з періодом T_1. Ми зараз нехтували рядом обставин: тим, що передача збудження від одного елемента до іншого вимагає деякого часу, можливістю приходу сигналу від одного елемента до іншого в той момент, коли цей інший знаходиться в стані рефрактерності, і таке інше. Урахування цих чинників ускладнює аналіз, проте сам факт синхронізації елементів з різними періодами залишається вірним при достатньо загальних припущеннях.

Описаний механізм синхронізації важливий, зокрема, для забезпечення нормальної роботи серця. Ритм серцевих скорочень задається так званим синусним вузлом. Цей вузол складається з дуже великого числа клітин (десятки тисяч), кожна з яких володіє спонтанною активністю, причому періоди спонтанної активності різних клітин досить близькі між собою. Найшвидші з цих клітин задають, відповідно до сказаного вище, ритм всього синусного вузла, а тим самим і ритм серця в цілому (решта клітин серця або зовсім позбавлені спонтанної активності, або мають період більший, ніж клітини синусного вузла). Якщо якісь з клітин синусного вузла перестануть функціонувати, то роль провідних елементів перейде до найшвидших з тих, що збереглися і т. д. Такий механізм регуляції дуже надійний і стійкий – виключення якої-небудь частини синусного вузла лише неістотно змінить режим роботи серця. У цьому полягаэ одна з цікавих важливих відмінностей живих систем від більшості технічних пристроїв, де вихід з ладу однієї відповідальної деталі може повністю порушити роботу всієї системи.

Одне з важливих застосувань аксіоматичних моделей збудливих середовищ – це теоретичне дослідження різних патологічних режимів роботи серцевого м'яза. Вперше відповідна аксіоматична модель була сформульована Н. Вінером і А. Розенблютом. Серед різних порушень нормальної роботи серця одне з найважчих є так звана фібриляція. Це явище, добре відоме лікарям і фізіологам, полягає у тому, що замість ритмічних узгоджених скорочень в серці виникають безладні локальні збудження, позбавлені якої-небудь періодичності. Фібриляція шлуночків серця повністю порушує його гемодинамічні функції і через декілька хвилин приводить до смерті. Фібриляція передсердя не є таким катастрофічним порушенням (вона може тривати протягом років), але все ж таки це достатньо важке захворювання.

Певний зв'язок з фібриляцією мають і інші, менш радикальні порушення серцевої діяльності – різні форми аритмії.

Механізм виникнення фібриляції в реальному серці не цілком ясний,

проте деякі висновки про природу цього явища можна зробити навіть на підставі тієї аксіоматичної моделі, яку ми аналізуємо. Розглянемо двовимірне збудливе середовище, по якому періодично розповсюджується хвиля збудження. Припустимо, що в мить, коли по ньому пробігає чергова хвиля, деяка частина середовища виявляється тимчасово загальмованою рис. 2.21 а), тобто несприйнятливою до збудження (така тимчасова незбудливість окремих ділянок може виникати в серцевому м'язі під впливом нервових дій). Припустимо далі, що якраз в той момент, коли хвиля збудження огинає загальмовану область, остання виходить із загальмованого стану.

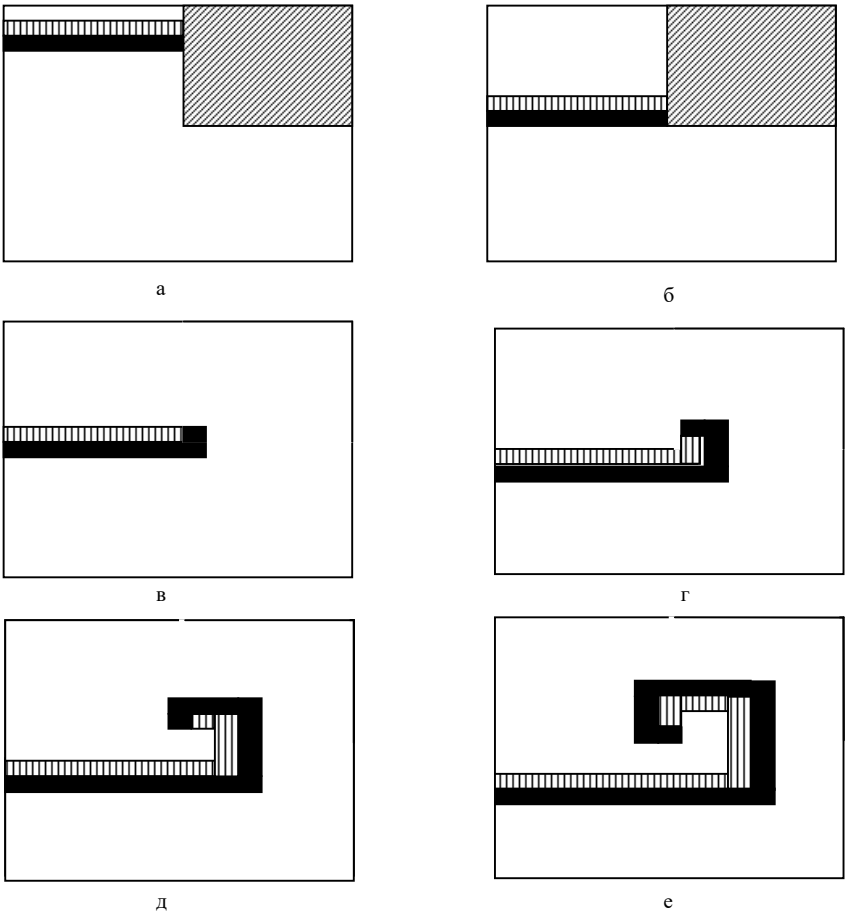


Рис. 2.21. - Схема виникнення реверберatора.

При цьому хвиля збудження почне розповсюджуватися всередину області, що вийшла з гальмування (рис. 2.21 б). Якщо розміри цієї області достатньо великі, то хвиля збудження, пройшовши по ній, обігне рефрактерну ділянку і почне її періодично оббігати (послідовні етапи цього процесу зображені на рис. 2.21 в – 2.21 е). Рисунок відтворено у відповідності до результатів роботи програми, що реалізована по описаній аксіоматичній моделі роботи клітини. У програмі обробляється матриця клітин, кожна клітина переходить зі стану спокою в стан збудження і потім в рефрактерний стан за описаним алгоритмом. Початкові умови (тимчасова незбудливість окремої ділянки) дають можливість показати утворення спіральної хвилі – ревербератора.

Хвиля збудження йде зверху вниз; чорним показане збуджене середовище, вертикальним штрихуванням – рефрактерне середовище; діагональним штрихуванням – незбудлива ділянка середовища (рис. 2.21А), яка стає збудливою якраз в той момент, коли повз нього проходить хвиля збудження (рис. 2.21 б). (рис. 2.21 д) – (рис. 2.21 е) – подальший хід хвилі при виникненні ревербератора.

У подальших роботах такі спіралі одержали назву *реверберато́рів.* Як показує детальніший аналіз, такі ревербератори через деякий час або зникають, або породжують нові ревербератори.

Якщо реверберато́рів з'являється більше, ніж зникає, то в середовищі розвивається лавиноподібний процес наростання безладної активності, аналогічний реальній фібриляції серцевого м'яза. У описаній вище схемі основою виникнення аномального режиму була деяка неоднорідність середовища (тимчасова незбудливість однієї з ділянок середовища).

Проведені рядом авторів дослідження показали, що ця умова носить вельми загальний характер: у основі виникнення режимів типу фібриляції, як правило, лежить та або інша форма неоднорідності середовища. Ця неоднорідність може мати різну природу: різні ділянки середовища можуть початково відрізнятися періодом рефрактерності або іншими характеристиками, неоднорідність (тимчасова) може створюватися якими-небудь попередніми гальмуючими або збудливими діями. Ця обставина добре узгоджується з відомими в медицині фактами: порушення серцевого ритму виникають, як правило, на грунті серцевих захворювань (інфаркти, недостатність коронарного кровообігу і т. д.), що порушують однорідність серцевої тканини. З іншого боку, ще в початковій роботі Н. Вінера і Л. Розенблюта було показано, що і в абсолютно однорідному збудливому середовищі виникнення фібриляції можливе лише за деяких вельми штучних умов її стимуляції.

Медичні системи забезпечення розв'язання для задач моделювання серцево-судинної системи. Не можна не визнати, що математична теорія є найвдалішою формою організації знань, основою наукових досліджень і техніко-економічних розробок. Зміст теорії повинен достатньо повно описувати і адекватно відображати властивості і відношення середовища, що вивчається, а форма – забезпечити можливості розв'язання задач досить широкого класу, який і визначає пізнавальну і практичну цінність теорії. Суть математичного підходу полягає в розвиненості математичної мови разом з універсальністю і строгістю методів аналізу, їх тісного гармонійного взаємозв'язку. Ці властивості складають ту суть математизації, яка визначила її провідну роль в природничо-наукових дисциплінах. Тому як системи, що організують знання і процес дослідження, наука в даний час використовує головним чином математичні (і логічні) обчислення. Розроблений загальний математичний опис серцево-судинної системи відповідає традиційному підходу. Разом з тим ці описи, їх взаємозв'язок і методи дослідження з самого початку орієнтовані на нові можливості. Ці можливості обумовлені виникненням і широким розвитком принципово нових комп'ютерних систем, що об'єднують логічні, математичні і технічні структури.

Відправним поняттям, на якому засновано подальший виклад, є уявлення про автоматизовану організацію знань, методів і наукових досліджень. За аналогєюї з математичними теоріями (що об'єднують відомості за допомогою математичних і логічних засобів), методи і знання певної наукової дисципліни, організовані засобами автоматизованих систем, можна розглядати як автоматизовану організацію теоретичних, експериментальних і прикладних знань, включаючи наукові дослідження і застосування. Поняття «знання» включає категорії, принципи і числення (правила висновку), специфічні для даної дисципліни, а також моделі і методи їх дослідження. Це поняття також репрезентує: експериментальні оцінки; методики планування, ідентифікації, контролю; прогнозування результатів і обробки спостережень; експертні оцінки і системи; адміністративні рішення і «інженерні» методи. Нарешті, сюди можна віднести самі взаємостосунки людина – автоматизована система, які виникають разом з організацією знань і методів технічними (і програмними) засобами, фіксуються в структурі системи, алгоритмах, інструкціях і стають невід'ємною частиною, що багато в чому визначає її ефективність.

Для першої стадії автоматизації (головним чином невеликих виробничих агрегатів) характерна тенденція повної заміни ручного управління (розробка автоматів). Основна тенденція – виробництво без людини.

Чим об'єкти складніше автоматизуються, тим важче виявляється повна автоматизація. В усякому разі з часом стає ясно, що економічно вигідно

залишати людину в контурі управління. При автоматизованій організації знань і досліджень задача полягає в створенні умов, що активно сприяють творчості людини. Ці умови різні для різних наукових напрямів і дисциплін, змінюються з підвищенням рівня і обсягу знань, залежать від індивідуальності дослідника і т. ін. Тому організація знань і форма роботи з ними, як і конкретні задачі і методи, повинні визначатися кожним дослідником індивідуально відповідно до його наукових ідей. Це означає, що мета функціонування автоматизованої системи забезпечення наукових дослідженнь при кожному індивідуальному використанні задає людина.

Розвиток моніторно-комп'ютерних систем у медицині почався із задачі стеження за станом хворого. У зв'язку з цим відзначимо лише, що звичний монітор допомагає лікарю оцінити стан хворого, сигналізує про вихід показників зі встановлених меж, автоматично записує екстремальні значення контрольованих показників зі встановлених меж і автоматично записує екстремальні значення контрольованих показників на магнітофон або самописець.

Така система розроблена і працює в центрі серцево-судинної хірургії ім. А. Н. Бакулєва в Москві. Основною перевагою моніторно-комп'ютерного спостереження в порівнянні з моніторним стеженням є зберігання контрольованих показників в пам'яті комп'ютера. Тим самим лікар дістає можливість спостерігати за об'єктивно, документально і детально відображеною динамікою патофізіологічних процесів. При цьому передбачається програмна обробка даних з метою прогнозування критичних станів з надсиланням застережливого повідомлення на центральну станцію.

У всьому світі в останні роки спостерігається брак середнього медичного персоналу. В інтенсивній терапії загального профілю використання центральних сестринських постів дозволяє скоротити число чергового персоналу (особливо того, що працює вночі). Кардіохірургічні клініки страждають від браку кваліфікованого середнього медичного персоналу. При лікуванні хворих із станом середньої тяжкості інтенсивний моніторний контроль дозволяє знизити робоче навантаження медичної сестри і (при відповідній організації чергувань) кількість обслуговуючого персоналу.

При упровадженні моніторних систем декілька слабшає нервова напруга, що викликається постійним стеженням за станом тяжкохворих і за контрольованими показниками, виключається робота, пов'язана з калібруванням приладу, установленням «нуля», швидкістю запису, вимірюваннями і оцінками результатів і ін. Разом з тим з'являються нові обов'язки сестри і лікаря: контроль за електродами (ЕКГ, РПГ і ін.), розпізнавання помилкових і істинних сигналів тривоги, постійна робота з «інвазивними» датчиками (забезпечення стерильності, асептики), введення

результатів контролю і спостережень через клавіатуру в пам'ять машини, читання і розшифрування інформації, що надходить на дисплей, спілкування з машиною в діалоговому режимі і т. ін. Все це різко підвищує вимогу до кваліфікації і інтелектуального рівня середнього медичного персоналу. Ще більшою мірою зростає інтелектуальне навантаження лікаря.

Така система забезпечує автоматичну інтенсивну терапію. Програмне управління терапією може бути виконане за допомогою автоматичних шприців і роликових насосів. Ці засоби забезпечують точність і надійність дозування лікарських засобів, управління режимом інфузії за допомогою комп'ютера. Для цього передбачено використання пробних і тестових дій, управління, – що об'єднує діагностику і терапію, автоматичний пошук найефективніших доз, поступове зниження дози ліків аж до їх відміни, управління апаратом штучного дихання, контрпульсатором, пейсмекером, дренажами і т. ін.

З часом була усвідомлена визначальна роль проблеми забезпечення рішень лікаря і їй як основній були підпорядковані всі інші задачі моніторно-комп'ютерної проблематики:

- автоматичний контроль;
- автоматичні і автоматизовані вимірювання;
- автоматична діагностика і прогнозування критичних станів;
- організація даних в пам'яті машини;
- оперативний і апостеріорний аналіз (статистичний, аналіз Фур'є, логічний, оптимізація, планування досліджень і ін.);
- наочне подання даних лікарю;
- автоматична терапія;
- діалог лікар – комп'ютер;
- інтерактивний контроль хворого – комп'ютер – лікар;
- об'єктивізація, узагальнення і використання знань лікарів (експертні системи).

Досвід роботи автоматизованої системи забезпечення рішень ІССХ ім. А.Н. Бакулєва показав превалюючe значення розвиненого математичного забезпечення, включаючи:

- операційні системи, орієнтовані на медичне застосування;
- інтегровані пакети програм, включаючи засоби діалогу, графіки, програмування, редакторів тексту, бази даних, електронні таблиці, бібліотеки стандартних програм;
- бібліотеку змістовних медичних програм;
- бібліотеку загальноматематичних програм;
- систему програм для діалогового спілкування лікаря і медичної сестри з комп'ютером
- достатньо розвинений, орієнтований на конкретне медичне

застосування сервіс (програми обслуговування);

- мови моделювання, дозволяючи створити і використати програми для дослідження патофізіологічних процесів.

У цілому проблему забезпечення рішень можна розглядати як організацію знань, методів і самого процесу дослідження засобами автоматизованих інформаційно-управляючих систем.

Задача забезпечення рішень приводить до необхідності організації в пам'яті комп'ютера не тільки даних моніторного контролю після операцій, але і результатів, одержуваних під час операції, а також даних передопераційного контролю і діагностики, включаючи ангіографію, радіоізотопні вимірювання, лунокардіографію, томографію і таке інше. Таким чином, виникла необхідність в автоматизованому контролі всіх етапів лікування хворого в кардіохірургічній клініці і відповідно в автоматизованому забезпеченні цих етапів. З'явилися і зручні технічні засоби для розв'язання цих задач – мережі персональних комп'ютерів.

Автоматизована система підтримки рішень повинна мати модель взаємостосунків цільових установок осіб, що ухвалюють взаємообумовлені рішення.

Оскільки база даних кардіохірургічного відділення працює в інтегрованій системі, що забезпечує одночасно з її роботою підтримку локальної мережі, обробку сигналів аналог-кодових і код-аналогових перетворювачів, роботу інших призначених для користувача програм (наприклад, за статистикою, побудовою графіків і ін.), збір, пошук і передачу даних здійснюють в оперативному режимі, близькому до реального часу. Зазвичай система включає такі класи даних: паспортні дані (ПІБ, номер історії хвороби, ріст, вага. вік й ін.); відомості про об'єм моніторного контролю (наявність катетерів, методика визначення хвилинного об'єму кровообігу та ін.); післяопераційний епікриз з описом операції, даних, необхідних для аналізу ускладнень, що виникають в ранньому післяопераційному періоді, і для вибору лікування: щоденник лікувальних заходів; висновок по веденню хворого на автоматизованій системі забезпечення рішень лікаря; результати аналізу з використанням клініко-математичної класифікації; дані вимірювань, узяті з аналог-коду (тиск в порожнинах серця і магістральних судинах, частота серцевих скорочень, імпеданс і ін.); розрахункові показники, що характеризують функції і властивості підсистем організму, зв'язані математичними моделями (кровообіг серця, дихання); додаткові розрахункові показники (індекси, енергетичні і статистичні оцінки та ін.); системні набори показників, що характеризують окремі стани хворого або груп хворих (у момент надходження у відділення реанімації, до введення ліків і ін.); кількісні оцінки, що характеризують неускладнений перебіг післяопераційного

періоду.

Робота з базою даних здійснюється в діалоговому режимі, за допомогою вибору в меню необхідної позиції. Можливе звернення до показників, що зберігаються в базі, минувши меню, що в окремих випадках прискорює роботу. Показники, внесені в базу даних, доступні для обробки як стандартними засобами (методи статистики, графічні зображення), так і спеціалізованими засобами, що забезпечують підтримку рішень. До останніх належать спеціалізовані програми розрахунку оцінок властивостей і функцій, такі, як «Гарвей», «Індекс», «Респ». «Айболіт» і ін., що були розроблені в центрі серцево-судинної хірургії.

У кожен даний період часу система виконує декілька задач з обслуговування того ж самого хворого, декількох хворих, задачі, що "ініціюються" лікарями, середнім медичним персоналом, оператором, програмістами, і задачі, запрограмовані для виконання в певний момент часу.

Пріоритет задач встановлюється під час генерації. Одне або два ліжка для найважчих хворих зручно виділити наперед. Операційна система генерується відповідно до конфігурації технічних засобів. Важливо передбачити можливість зміни конфігурації або додавання виконуваних задач системою.

Для забезпечення досліджень і рішень лікаря виявилися необхідними пакети програм, що виконують такі функції: розв'язання диференціальних рівнянь; розв'язання лінійних рівнянь алгебри; обчислення основних і непараметричних статистичних оцінок: прогнозування і аналіз часових рядів; генерація випадкових чисел; інтерполяція, апроксимація і згладжування; операції над матрицями; чисельне інтегрування: кореляційний, регресійний і дисперсійний аналіз; обчислення функцій розподілу; лінійне програмування; динамічне програмування. (рис. 2.22).

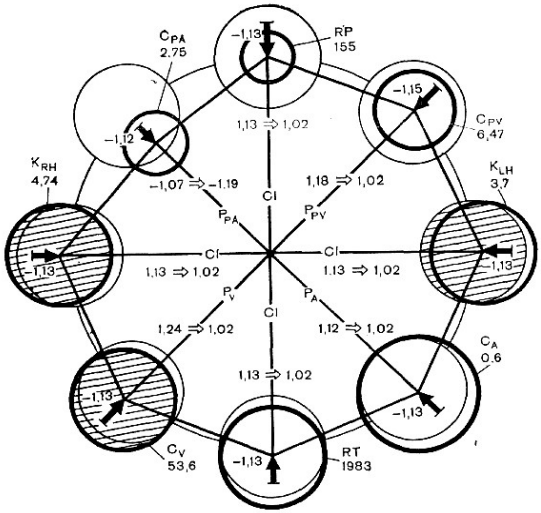


Рис. 2.22 Суміщені схеми усередненої моделі і моделі хворого, що проходить лікування на автоматизованій системі забезпечення рішень після нормалізації стану правого і лівого шлуночків серця і венозного резервуара.

Бібліотека програм включає більше 40 підготовлених до дослідження моделей серцево-судинної системи, судинної ділянки, серця, системи дихання, регуляції та ін.

Показники, що контролюються моніторами і вводяться лікарем, потрапляють в пам'ять системи. На їх основі оцінюються властивості моделі. Використовуючи індивідуальну модель, система знаходить "найслабкішу ланку". Якщо призначене раніше лікування покращує стан слабкої ланки, то система прагне вибрати "оптимальну" дію. Якщо призначене лікування не поліпшило цей стан (або слабка ланка ще не була діагностовано, або змінилося в ході лікування), то тоді лікар може вибрати необхідну тактику лікування. Якщо цей вибір виконаний правильно, то наступний цикл аналізу (через 3–10 хвилин) піде по зовнішньому контуру; інакше – по внутрішньому.

Система потребує певного технічного забезпечення. Це засоби і методи автоматизованого збору даних, їх автоматичної передачі в комп'ютер, організації в пам'яті, оперативної обробки, наочного зображення і аналізу в діалоговому режимі. Тому актуальні не стільки нові технічні розробки, виконані спеціально для автоматизації досліджень і організації знань, скільки комплексне інтегроване застосування наявних технічних і математичних засобів. Розглянемо деякі аспекти.

Введення аналогової інформації здійснюється через перетворювачі

аналогових сигналів в код або спеціальні пристрої (рис. 2.23).

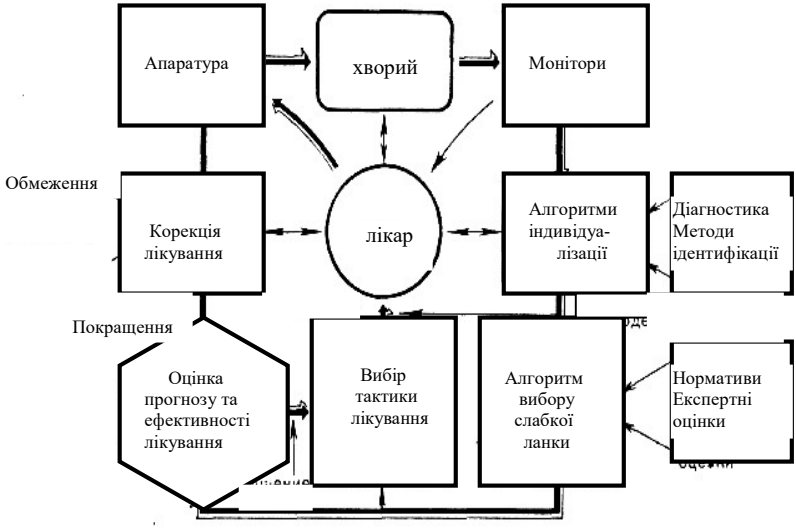


Рис. 2.23 Схема функціонування автоматизованої системи забезпечення рішень

На одне ліжко відділення кардіохірургічної інтенсивної терапії бажано мати не менше 6 каналів. Нижче наведено показники, що автоматично (on-line) надходять в комп'ютер: артеріальний тиск, тиск в легеневій артерії, тиск в лівому передсерді, центральний венозний тиск або тиск у правому передсерді, температура, частота пульсу, частота дихання, крива пульсу (плетізмограма, частота), ЕКГ (3 відведення), об'ємна швидкість видиху, доза ліків, реоплетізмограма.

Криві, що виводяться на дисплей, можна згладжувати і диференціювати; передбачено можливість їх архівування в пам'яті комп'ютера. Насущним залишається об'єктивний контроль дій (крапельниці, відсмоктування, положення хворого, режими апарата штучного тиску, пейсмекера і контрпульсатора), ваги хворого, крововтрати тощо. Вже є досвідчені системи з програмним установленням нуля датчиків і їх автоматичним калібруванням. Фірма General Electric забезпечує тестовий контроль моніторів.

Починаючи з середини 80-х років, виявилася тенденція до інтеграції всіх комп'ютерних служб медичного закладу на основі загальної бази даних, протоколів зв'язку і комп'ютерної мережі. Така тенденція спостерігалась в розвитку медичних комп'ютерних систем найбільш оснащених госпіталів світу. У такому випадку повинна бути забезпечена єдність концептуально-фізіологічного, клінічного, математичного і технічного аспектів проблеми.

Автоматизовані системи забезпечення рішень лікаря на сучасному етапі можуть розглядатися як системи, що організують технічне, методичне і інформаційне забезпечення лікувального процесу і наукових досліджень.

Комплектація системи повинна (за вибором) передбачати можливість поставки: технічно і системно узгоджених між собою процесорів; модулів оперативної і довготривалої пам'яті; периферійного і допоміжного устаткування; каналів зв'язку; засобів введення і виведення інформації; наборів датчиків; пакетів системного (програмного) забезпечення, бібліотек математичних програм (за вибором), медичної операційної системи і пакетів спеціалізованих медичних програм.

Особливе значення має технологія організації лікувального процесу. Суть технології полягає в безперервному порівнянні комп'ютерного прогнозу результатів, ускладнень і, головне, засобів і методів операційного і післяопераційного лікування з лікуванням, що має місце під час операції і після неї. Після цього на основі прогнозу визначається і реалізується оперативна, а також загальна лікувальна тактика. Результати лікування і шляхи їх досягнення фіксуються в пам'яті комп'ютера. Тепер є всі умови, щоб після закінчення, а частково і в ході лікування провести порівняння прогнозу і дійсних результатів і виявити їх розбіжності. На наступному етапі, використовуючи статистичний аналіз, моделювання і сортування даних, виявляються причини неспівпадання прогнозу і результатів. Відповідно до цих причин проводиться корекція діагностики, організації і тактики лікування, а також методів прогнозу. Для кожного класу помилок проводиться детальний аналіз. Такий аналіз не дозволяє усунути вже допущену помилку, але дає можливість не допустити її наступного разу. Організація управління лікувальним процесом на комп'ютері усуває суб'єктивність і тим самим переводить всю технологію планування і управління на наукові методи.

Контрольні питання

1. Як відображається вплив глюкози та інсуліну в вуглеводному обміні в системі диференціальних рівнянь ?
2. Що таке гіперглікемія та гіпоглікемія ?
3. Яка роль інсуліну та адреналіну в розвитку стану гіперглікемії та гіпоглікемії ?
4. Опишіть модель вуглеводного обміну в організмі.
5. Що таке глікемічна крива ?
6. Як утворюється глікемічна крива на графіку ?
7. Як виглядає глікемічна крива при роботі системи вуглеводного обміну в організмі в нормі ?
8. Як позначається на вигляді глікемічної кривої відсутність чутливості печінки до інсуліну ?
9. Як позначається на вигляді глікемічної кривої відсутність чутливості

тканин до інсуліну ?

10. Як враховується в моделі відсутність чутливості печінки до адреналіну ?

11. Як теорія вуглеводного обміну в організмі застосовується в дієтології ?

12. З чого складається імунна система ?

13. Яка функція імунної системи?

14. Як відбувається процес утворення антитіл в організмі у відповідь на вторгнення антигенів?

15. Опишіть математичну модель, що відтворює імунну відповідь організму на вторгнення антитіл.

16. Які можуть бути форми протікання хвороби в організмі людини?

17. Що таке імунологічний бар'єр ?

18. Як можна дослідити на моделі випадок, коли організм не виробляє антитіл даної специфічності?

19. Як можна дослідити на моделі випадок сильно вираженої імунної відповіді?

20. Розтлумачте управління параметрами системи як процесс лікування?

21. Які фактори враховуються при моделюванні дії на біосистему терапевтичних засобів?

22. Опишіть роботу серцево-судинної системи.

23. На яких припущеннях базуєьбся модель Франка?

24. Яке диференціальне рівняння для тиску маємо в моделі Франка

25. Яке відношення електричні кола мають до моделювання серцево-судинної системи ?

26. Які моделі серця ви знаєте ?

27. Як за допомогою аксіоматичних моделей можна дослідити авто-хвильові процеси в серцевому м'язі ?

28. Опишіть аксіоматичну модель фібриляції. Яка може бути програмна реалізація цієї моделі.

29. Яка мета створення автоматизованої системи прийняття рішень ?

30. Як працюють створені автоматизовані системи прийняття рішень для моделювання серцево-судинної системи ?

31. Яке програмне забезпечення потрібно мати в автоматизованій системі прийняття рішень для моделювання серцево-судинної системи ?

3. ЦИФРОВІ ПРОЦЕСОРИ ОБРОБКИ СИГНАЛІВ – ТЕХНІЧНІ ЗАСОБИ КОМП'ЮТЕРІЗАЦІЇ СПЕЦІАЛІЗОВАНИХ СЕРЕДОВИЩ

3.1. Загальна характеристика цифрових процесорів обробки сигналів (ЦПОС) та область їх застосування

Цифрова обробка сигналу (ЦОС) – це арифметична обробка в реальному масштабі часу послідовності значень амплітуди сигналу, визначених через рівні часові проміжки. Прикладами цифрової обробки є:

- фільтрація сигналу;
- згортка двох сигналів (змішування сигналів);
- обчислення значень кореляційної функції двох сигналів;
- підсилення, обмеження або трансформація сигналу;
- пряме/зворотне Фур'є-перетворення сигналу.

Аналогова обробка сигналу, що традиційно використовується в багатьох радіотехнічних пристроях, є у багатьох випадках дешевшим способом досягнення необхідного результату. Проте тоді, коли потрібна висока точність обробки, мініатюрність пристрою, стабільність його характеристик в різних температурних умовах функціонування, цифрова обробка виявляється єдиним прийнятним рішенням [23].

Приклад аналогової фільтрації сигналу наведено на рис. 3.1.

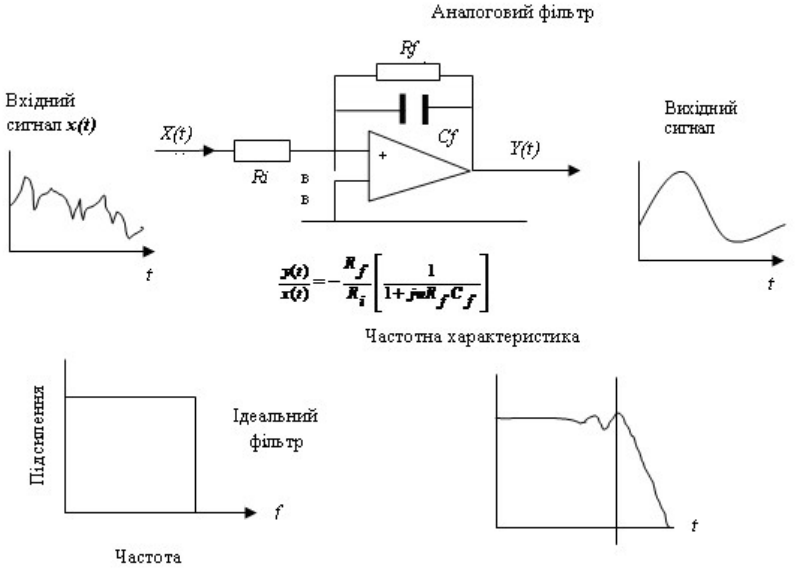


Рис. 3.1. Аналогова обробка сигналу

Використаний у фільтрі операційний підсилювач дозволяє розширити динамічний діапазон сигналів, що обробляються. Форма амплітудно-частотної характеристики фільтра визначається значеннями величин *Rf, Cf* . Для аналогового фільтра складно забезпечити високе значення добротності, характеристики фільтра сильно залежать від температурного режиму. Компоненти фільтра вносять додатковий шум в результуючий сигнал. Аналогові фільтри важко перебудовувати в широкому діапазоні частот.

Аналогічні результати обробки сигналу можуть бути отримані за допомогою цифрової схеми, показаної на рис. 3.2 [23].

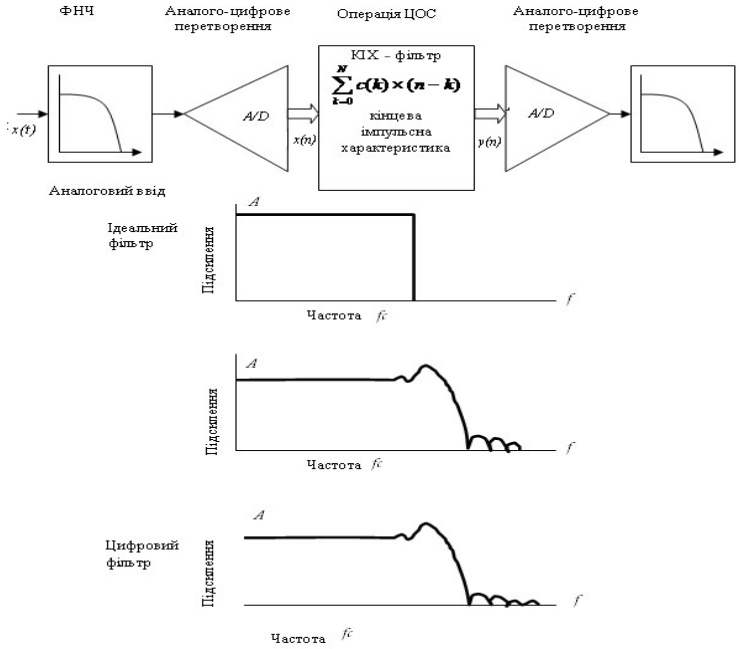


Рис. 3.2. Цифрова обробка сигналу

Компонентами схеми є фільтри низької частоти (ФНЧ), які виконують попереднє і подальше видалення з частотного спектру додаткових гармонік сигналу, аналого-цифровий (АЦП) та цифро-аналоговий (ЦАП) перетворювачі сигналу і власне цифровий фільтр з кінцевою імпульсною характеристикою. Амплітудно-частотна характеристика фільтра визначається значеннями коефіцієнтів фільтра С(к). Змінюючи кількість коефіцієнтів (довжину фільтра) і їх значення, можна отримати фільтр з будь-якою

амплітудно-частотною характеристикою. Шуми, що вносяться (шуми квантування), залежать від частоти і розрядності АЦП і ЦАП, а також від точності обчислень.

Схема перетворення, що виконується над послідовністю відліків сигналу, яка задається математичною формулою, може бути також зображена графічно у вигляді структурної схеми цифрового фільтра.

Існує класифікація фільтрів за виглядом імпульсної характеристики: фільтри з кінцевою імпульсною характеристикою КІХ (FIR) і фільтри з безкінечною імпульсною характеристикою БІХ (IIR).

Структура цифрових фільтрів типу FIR і IIR наведена на рис.3.3 і 3.4 відповідно. На цих рисунках прийняті такі позначення: T – блок затримки на 1 такт; X – блок множення; + – блок складання [23].

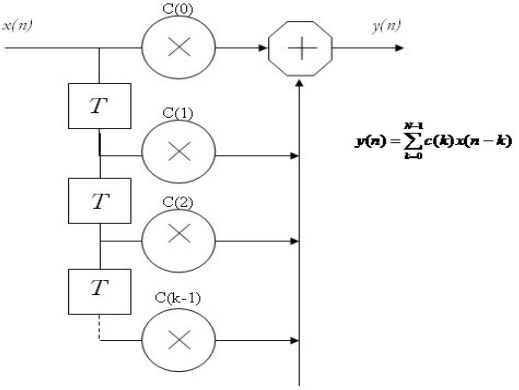


Рис. 3.3. Структура каскаду фільтра типу FIR

$$w(n) = x(n) - a_i 1 \times w(n-1) - a_i 2 \times w(n-2) \quad (3.1)$$

$$y(n) = w(n) + b_i 1 \times w(n-1) + b_i 2 \times w(n-2) \quad (3.2)$$

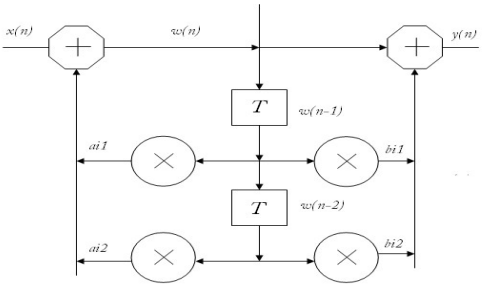


Рис. 3.4. Структура каскаду фільтра типу IIR

Для ефективної реалізації алгоритмів цифрової фільтрації необхідна апаратна підтримка базових операцій: множення з накопиченням (MAC); модульної адресної арифметики; нормування результатів арифметичних операцій.

Іншим перетворенням сигналу, часто виконуваним, є Фур'є-перетворення (пряме і зворотне).

Будь-який сигнал може бути зображений як в часовій області (сукупність графіків в координатах час-амплітуда), так і в частотній області (послідовність графіків в координатах частота-амплітуда). Залежно від складності реалізації обробки може бути вибране або частотне, або часове подання сигналу. Фур'є-перетворення дозволяє здійснювати перенесення сигналу з однієї форми подання в іншу.

Дискретне перетворення Фур'є (ДПФ) в аналітичному вигляді задається формулою [23]

$$X(f) \approx \widetilde{X}(f) = T \sum_{n=-\infty}^{+\infty} \chi(nT) e^{-j2\pi fnT}, \qquad (3.3)$$

де χ(nT) – послідовність відліків сигналу.

Існує велика різноманітність реалізацій дискретного перетворення Фур'є. У ряді алгоритмів використовуються прийоми, що дозволяють скоротити обсяг необхідних обчислень. Ці алгоритми відомі під загальною назвою «швидке перетворення Фур'є» (ШПФ).

На практиці інтервал підсумовування обмежений деяким числом часових відліків *N*, залежним від необхідної точності перетворення. В цьому випадку формула набуває вигляду [23]

$$X(f) \approx \widetilde{X}(f) = T \sum_{n=0}^{N-1} \chi(nT) e^{-j2\pi fnT}; \qquad (3.4)$$

N – називають числом точок перетворення.

Для зменшення числа операцій множення при виконанні ДПФ використовується метод, що отримав назву «проріджування за часом».

Суть даного методу полягає у тому, що перетворення Фур'є за послідовністю з *N* точок може бути виражене через перетворення, що виконані за підпослідовностями цієї послідовності, кожна з яких має довжину *N*/2 точок. Оскільки число множень пропорційне числу точок перетворення, то процедура двократного перетворення по *N*/2 точках (з подальшим об'єднанням результатів) уявляється вигіднішою з погляду часу обчислення. Застосувавши до послідовності відліків процедуру проріджування рекурсивно, отримаємо схему обчислень, зображену на рис. 3.5.

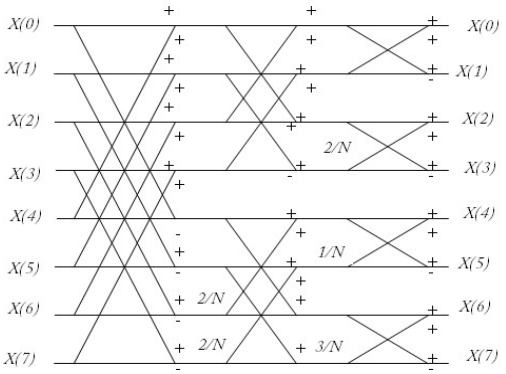


Рис.3.5. Схема ШПФ по 8 точках

На цьому рисунку рис *к/N* означає множення на коефіцієнт $e^{-jk\frac{2\pi}{N}}$.

При реалізації даної схеми перетворення разом з операціями множення і складання використовуються бітові операції. Як видно з рис 3.5, результуючі відліки розташовані в біт-реверсивному порядку, тобто такому, коли позиція елемента визначається реверсією двійкового подання індексу елемента. Для їх переупорядковування необхідно виконати або перестановку елементів масиву, або операцію бітової реверсії індексу при зверненні до елемента масиву. Другий підхід є більш економічним з погляду часу доступу, проте вимагає можливості маніпулювання адресами даних.

У більшості реальних додатків розглянуті базові алгоритми цифрової обробки сигналів повинні виконуватися в режимі реального часу, що ставить підвищені вимоги до продуктивності процесора, що їх реалізує. Апаратна підтримка базових операцій алгоритмів цифрової обробки сигналів є характерною особливістю сигнальних процесорів. Нижче будуть розглянуті основні сімейства сигнальних мікропроцесорів, представлені на світовому ринку.

3.1.1. Організація обчислень в ЦПОС. Особливості архітектури

Процесори ЦПОС можна для зручності розділити на дві категорії: універсальні і спеціалізовані [23].

Існує два типи спеціалізованих пристроїв.

1. Апаратне забезпечення, розроблене для ефективного виконання спеціальних алгоритмів ЦОС, таких, як цифрова фільтрація, швидке перетворення Фур'є. Пристрої даного типу іноді називають алгоритмічними процесорами ЦОС.

2. Апаратне забезпечення, розроблене для спеціального додатку, наприклад, у сфері контролю, телекомунікацій або цифрового аудіо. Пристрої даного типу іноді називають процесорами ЦОС спеціального призначення (спеціалізованими) .

В більшості випадків спеціалізовані процесори виконують такі алгоритми ЦОС, як кодування-декодування. Крім того, вони повинні виконувати інші операції, що відображають специфіку додатку.

Всі універсальні і спеціалізовані процесори можна побудувати за допомогою окремих мікросхем або блоків помножувачів, арифметико-логічних пристроїв (АЛП), елементів пам'яті та ін.

Розглянемо архітектурні особливості процесорів ЦОС, які дозволили застосувати цифрову обробку у реальному часі в багатьох областях.

Більшість доступних зараз універсальних процесорів заснована на архітектурі фон Неймана, при якій операції виконуються послідовно. При обробці інструкції в такому процесорі блоки процесора, не задіяні в кожній фазі інструкції, перебувають у стані очікування до передачі їм управління. Підвищення швидкості процесора досягається за рахунок прискорення роботи окремих блоків.

Якщо пристрій має працювати у реальному часі, то архітектуру процесора ЦОС потрібно оптимізувати під виконання функцій ЦОС. Наприклад, на рис. 3.6 показана загальна апаратна архітектура цифрової обробки сигналів у реальному часі. Вона характеризується такими особливостями:

• містить багатошинну структуру з роздільною пам'яттю для даних і інструкцій програми. Звичайно пам'ять для зберігання даних містить вхідні дані, проміжні значення і вихідні вибірки, а також фіксовані коефіцієнти, наприклад, для цифрової фільтрації або ШПФ. Команди програми зберігаються у спеціально відведених елементах пам'яті;

• порт вводу-виводу дозволяє обмінюватися даними із зовнішніми пристроями (АЦП, ЦАП) або передавати цифрові дані іншим процесорам. Прямий доступ до пам'яті (Direct Memory Access — DMA), якщо він є, дозволяє швидко обмінюватися блоками даних з пам'яттю (ОЗП) для зберігання даних, причому звичайно це відбувається під зовнішнім управлінням;

• містить арифметичні пристрої для логічних і арифметичних операцій, до числа яких виходять АЛП, апаратні помножувачі і схеми зсуву (або помножувачі-накопичувачі).

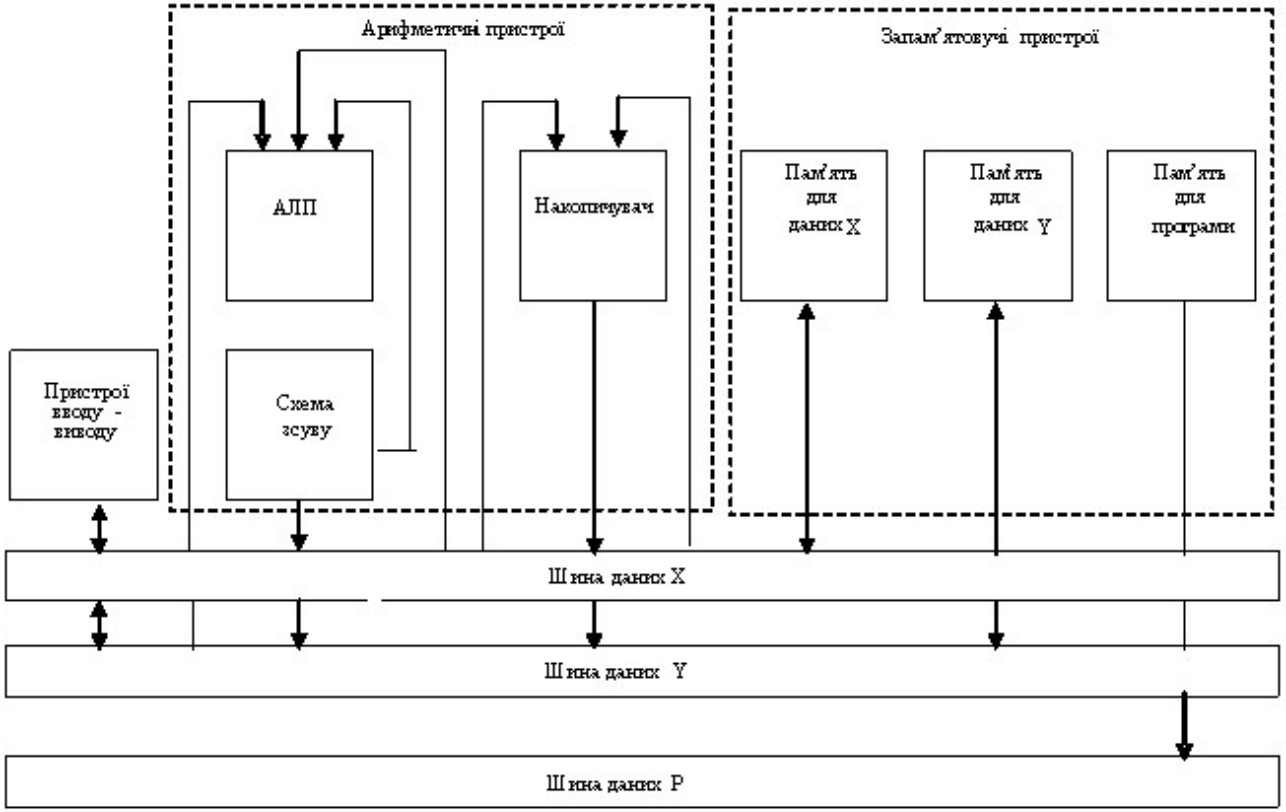


Рис. 3.6. Стандартна універсальна апаратна архітектура обробки сигналів

Для цифрової обробки сигналів використовуються так звані сигнальні мікропроцесори. До їх особливостей можна віднести малорозрядну (40 розрядів і менш) обробку чисел з плаваючою точкою, переважне використання чисел з фіксованою точкою розрядності 32 і менш, а також орієнтацію на нескладну обробку великих масивів даних.

Відмітною особливістю задач цифрової обробки сигналів є потоковий характер обробки великих обсягів даних в реальному режимі часу, який вимагає від технічних засобів високої продуктивності і забезпечення можливості інтенсивного обміну із зовнішніми пристроями. Відповідність до даних вимог досягається завдяки специфічній архітектурі сигнальних процесорів і проблемно-орієнтованій системі команд.

Сигнальні процесори мають високий ступінь спеціалізації. У них широко використовуються методи скорочення тривалості командного циклу, характерні і для універсальних RISC-процесорів, такі, як конвеєризація на рівні окремих мікроінструкцій і інструкцій, розміщення операндів більшості команд в регістрах, використання тіньових регістрів для збереження стану обчислень при перемиканні контексту, розділення шин команд і даних (гарвардська архітектура). У той же час для сигнальних процесорів характерною є наявність апаратного помножувача, що дозволяє виконувати множення двох чисел за один командний такт. В універсальних процесорах множення звичайно реалізується за декілька тактів, як послідовність операцій зсуву і складання. Іншою особливістю сигнальних процесорів є включення до системи команд таких операцій, як множення з накопиченням MAC (C:=AxB+C з вказаним в команді числом виконань в циклі і з правилом зміни індексів елементів масивів A і B), інверсія біт адреси, різноманітні бітові операції. У сигнальних процесорах реалізується апаратна підтримка програмних циклів, кільцевих буферів. Один або декілька операндів витягуються з пам'яті в циклі виконання команди.

Реалізація однотактного множення і команд, що використовують як операнди вміст елементів пам'яті, обумовлює порівняно низькі тактові частоти роботи цих процесорів. Спеціалізація не дозволяє підіймати продуктивність за рахунок швидкого виконання коротких команд типу «регістр-регістр», як це робиться в універсальних процесорах. Цих команд просто немає в програмах обробки сигналів.

3.1.2 MAC-операція. Схеми реалізації в ЦПОС. Точність обчислень

Основними чисельними операціями в цифровій обробці сигналів є множення і складання. Множення в програмній формі сумно відоме своєю трудомісткістю, а якщо використовується арифметика з плаваючою точкою,

складання виявляється навіть ще більш трудомісткою операцією. Щоб максимально прискорити цифрову обробку сигналів у реальному часі, рекомендується використовувати спеціалізовані апаратні помножувачі-накопичувачі для арифметики з плаваючою або фіксованою точкою. Таке апаратне забезпечення тепер стандартно використовується у всіх цифрових процесорах сигналів. У процесорі з фіксованою точкою апаратний помножувач за один такт (звичайно 25 нс) приймає два 16-бітові дробові числа, подані у формі доповнення до двох, і обчислює їх 32-бітовий добуток. Середній час виконання команди множення-накопичення можна значно зменшити, якщо використовувати спеціальні команди повторення.

Типова конфігурація апаратного помножувача-накопичувача ЦОС зображена на рис. 3.7. У такій конфігурації помножувач має пару вхідних регістрів (Рг X та Рг Y), які містять входи помножувача, і 32-бітовий регістр добутку (Рг P), який містить результат множення. Вихід регістра P (product - добуток) з'єднується з накопичувачем подвійної точності, в якому накопичуються добутки.

Подібна конфігурація застосовується в апаратних помножувачах-накопичувачах з плаваючою точкою, за винятком того, що входи і добутки – це нормовані числа, що подані у формі з плаваючою точкою. Помножувачі-накопичувачі з плаваючою точкою дозволяють швидко обчислювати результати алгоритмів ЦОС з мінімальним помилками [23].

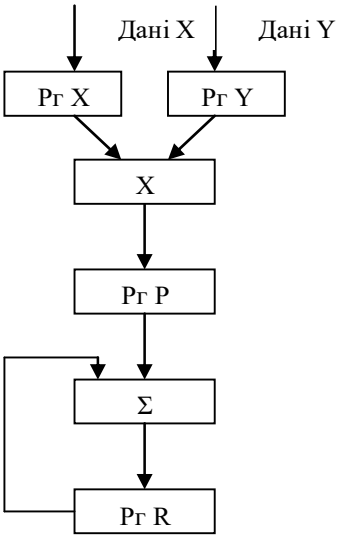


Рис. 3.7. Типова конфігурація апаратного помножувача-накопичувача ЦОС

Операції з плаваючою точкою дозволяють використовувати великий динамічний діапазон і знижують арифметичні помилки, хоча в багатьох додатках достатньо динамічного діапазону, який пропонує подання з фіксованою точкою.

Двома найпоширенішими типами арифметики, що використовуються в сучасних ЦПОС, є арифметики з фіксованою і плаваючою точками. Арифметика з плаваючою точкою – це природний вибір в додатках з широкими і змінними вимогами до динамічного діапазону (динамічний діапазон можна визначити як різницю між найбільшим та найменшим рівнем сигналу, який можна представити, або як різницю між найбільшим сигналом і мінімальним рівнем шуму, що вимірюються в децибелах). Процесори з фіксованою точкою переважні з погляду низької вартості, вони підходять для масового виробництва (наприклад, стільникові телефони і комп'ютерні дисководи). При використанні арифметики з фіксованою точкою виникають питання, пов'язані з обмеженнями динамічного діапазону. Взагалі процесори з плаваючою точкою дорожчі за процесори з фіксованою точкою, хоча останніми роками різниця в ціні істотно зменшилася. Більшість існуючих ЦПОС з плаваючою точкою також підтримує арифметику з фіксованою точкою.

Використання даних у форматі з плаваючою точкою в сигнальній обробці обумовлене декількома причинами. Для багатьох задач, пов'язаних з виконанням інтегральних і диференціальних перетворень, особливе значення має точність обчислень, забезпечити яку дозволяє експоненціальний формат подання даних. Алгоритми компресії, декомпресії, адаптивної фільтрації в ЦОС пов'язані з визначенням логарифмічних залежностей і досить чутливі до точності подання даних в широкому динамічному діапазоні.

Робота з даними у форматі з плаваючою точкою істотно спрощує і прискорює обробку, підвищує надійність програми, оскільки не вимагає виконання операцій округлення і нормалізації даних, відстежування ситуацій втрати значущості і переповнювання.

3.1.3. Фізична організація пам'яті в ЦПОС. Гарвардська архітектура та багатопортова пам'ять

Фізично пристрої пам’яті в ЦПОС можна розподілити на внутрішні та зовнішні. До внутрішніх належать:

• оперативний запам’ятовуючий пристрій (ОЗП), присутній у всіх типах ЦПОС;

• постійний запам’ятовуючий пристрій (ПЗП), присутній у деяких типах ЦПОС;

• напівпостійний запам’ятовуючий пристрій (НПЗП), присутній у деяких типах ЦПОС.

Ємність ОЗП, ПЗП та НПЗП залежить від типу ЦПОС і буде розглянута нижче для кожного типу процесора окремо.

Зовнішню пам'ять можна розподілити:

• на пам'ять для збереження даних;

• пам'ять для збереження програм.

Принципова особливість гарвардської (двошинної) архітектури полягає у тому, що пам'ять для зберігання даних і пам'ять для зберігання програми розташовуються в різних місцях, припускаючи повне поєднання в часі операцій виклику команди з пам'яті і її виконання [23]. Стандартна гарвардська архітектура наведена на рис. 3.8. Звичайні мікропроцесори, такі, як Intel 6502, характеризуються одношинною структурою, через яку передаються і дані, і команди. У гарвардській архітектурі, де команди програми і дані зберігаються в різних областях пам'яті, виклик наступної команди може співпадати з виконанням поточної команди. Як правило пам'ять програми містить програмний код, тоді як пам'ять для зберігання даних включає змінні, наприклад, вибірки вхідних даних.

Стандартна гарвардська архітектура використовується лише в декількох процесорах ЦОС (наприклад, Motorola DSP56000), в більшості пристроїв застосовується модифікована гарвардська архітектура (наприклад, сімейство процесорів TMS320). У модифікованій архітектурі також виділяються роздільні області пам'яті для зберігання програми і даних, але на відміну від строгої гарвардської архітектури тут дозволений зв'язок між двома областями пам'яті.

Для взаємодії із зовнішніми пристроями (АЦП, ЦАП та ін.) або з іншими процесорами в ЦПОС передбачена система портів вводу-виводу. За способом передачі даних (послідовний чи паралельний) та за розрядністю номенклатура портів вводу-виводу залежить від конкретного типу ЦПОС.

3.1.4. Логічна організація пам’яті та режими адресації

Комбінація гарвардської архітектури з швидкодіючою та багатопортовою пам’яттю дозволяє реалізувати вимоги до ефективної організації взаємодії з пам’яттю, якої потребують алгоритми ЦОС.

Деякі ЦПОС мають спеціальний кеш команд, який є пам’яттю невеликого розміру, що входить до складу ядра процесора та призначається для збереження команд. Попереднє завантаження команд до кеша дозволяє звільнити шини процесора для доступу до даних. Найпростіші процесори мають кеш на одну команду. В більш складних ЦПОС ємність кеша складає 1К слів.

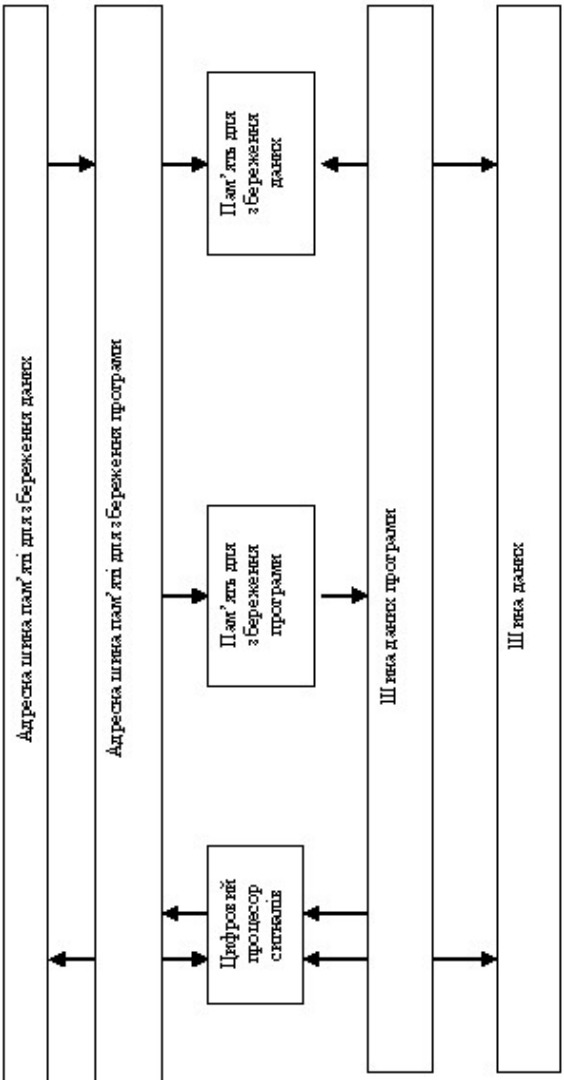


Рис. 3.8. Стандартна гарвардська архітектура

При виконанні будь-якої команди процесор звертається до пам'яті програми або до пам'яті даних. Спосіб визначення адреси операнда в команді або адреси передачі управління називається режимом адресації. У ЦПОС використовують режими таких видів адресації: прямої, безпосередньої та різних видів непрямої адресації [23].

Пряма адресація. Абсолютна адреса операнда міститься в кодовому слові команди. Наприклад, команда ЦПОС ADSP-21xx

AX0=DM(0800)

завантажує значення, розташоване за абсолютною адресою 0800 пам'яті даних до регістру AX0.

Безпосередня адресація. Значення операнда міститься в команді. Наприклад, команда ЦПОС ADSP-21xx

AX0=1357

Значення константи 1357 завантажується до регістру AX0. Якщо значення константи перевищує довжину одного кодового слова команди, воно розміщується в наступному кодовому слові. Це призводить до читання з пам'яті двох слів, що підвищує час виконання програми.

Непряма адресація. Використовує регістри-покажчики, вміст яких є реальною адресою розміщення даних у пам'яті. У цьому випадку в команді міститься тільки посилання на регістр-покажчик. Кількість регістрів для непрямої адресації в ЦПОС невелика, тому довжина команди непрямої адресації дорівнює одному слову. У ЦПОС використовуються такі види непрямої адресації: непряма регістрова адресація, непряма регістрова адресація з інкрементуванням, непряма регістрова адресація з модульною арифметикою, біт-інверсна адресація. Проілюструємо ці види адресації за допомогою операторів мови С++.

Непряма регістрова адресація. Відповідає наступній формі оператора присвоєння:

A=A+*R;

Відповідно до цього оператора значення, що зберігається за адресою пам'яті, яка міститься в регістрі R, додається до значення акумулятора A. Регістр R розглядається як покажчик.

Непряма регістрова адресація з інкрементуванням та декрементуванням. Відповідає наступним операторам:

A=A+(*R)++;

A=A+(*R)--;

Значення, що зберігається за адресою пам'яті, яка міститься в регістрі R, додається до значення акумулятора A. Після отримання значення з пам'яті вміст регістру-покажчика збільшується (++) або зменшується (--) на одиницю. Деякі ЦПОС дозволяють збільшувати або зменшувати вміст

регістра-покажчика не на одиницю, а на іншу величину, значення якої зберігається в додатковому регістрі.

Непряма регістрова адресація з модульною арифметикою. Використовується при організації в пам'яті кільцевого буфера. Буфер – послідовність однотипних елементів даних, що розташовані у суміжних комірках пам'яті. Якщо елементи даних обробляються в порядку їх запису у пам'ять, то буфер відповідає структурі даних, яка називається чергою. Для доступу до значень, що зберігаються в буфері, використовуються покажчики. Кожен раз після звертання до буфера значення покажчика змінюється з визначеним кроком. При досягненні кінця буфера значення покажчика змінюється на початкове. Цей механізм можна описати за допомогою умовного оператора

```
if(pointer+step<BkEnd);
  pointer=pointer+step;
else pointer=BkBegin;
```

Змінна BkBegin визначає початкову адресу, а BkEnd – кінцеву адресу області пам'яті, в якій розміщено буфер. Змінна pointer – покажчик, а змінна step відповідає кроку, з яким змінюється значення покажчика. Ця схема адресації використовується, наприклад, в ЦПОС TMS320C5x.

Можливе використання іншого механізму адресації кільцевого буфера, який застосовується в ЦПОС ADSP21xx, ADSP21xxx, TMS320C3x, TMS320C4x. У цих ЦПОС не задається кінцева адреса області пам'яті, відведеної під буфер, а в спеціальному регістрі фіксується довжина буфера. Значення, що відповідає довжині буфера, називається модулем. При збільшенні початкової адреси на величину модуля здійснюється перехід до початкової адреси, що і складає суть непрямої адресації з модульною арифметикою.

Біт-інверсна адресація. Дозволяє змінювати впорядкованість вхідних або вихідних даних, що потрібно, наприклад, в алгоритмах ШПФ. Біт-інверсний порядок задається шляхом «дзеркального» відображення двійкових розрядів вхідної чи вихідної послідовності. Приклад реалізації наведено в табл.. 3.1 [23].

Таблиця 3.1 – Біт-інверсний порядок

N	Двійковий номер	Біт-інверсія	Біт-інверсний номер
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

3.1.5. Архітектура й призначення спеціалізованих модулів ЦПОС: таймера, лічильника числа повторень, генераторів адреси

Призначення та роботу спеціалізованих модулів розглянемо на прикладі ЦПОС сімейства TMS320C2x [23].

У складі ЦПОС є 16-розрядний таймер (TIM) та регістр періоду таймера (PRD). Таймер управляється відповідним сигналом (CLKOUT1). При скиданні таймера в нього та регістр періоду передається максимальне число FFFFh. Після дії сигналу скидання вміст таймера зменшується на одиницю при кожному надходженні сигналу CLKOUT1. Початкове значення таймера визначається вмістом регістру періоду. Як тільки значення в таймері стане дорівнювати нулю, формується переривання від таймера (TINT) і він перезавантажує те значення, що знаходиться в регістрі періоду. Це відбувається під час першого машинного циклу після формування переривання TINT. Таким чином, переривання будуть відбуватися через проміжки часу, що дорівнюють $[(PRD)+1]*T_{CLKOUT1}$, де $T_{CLKOUT1}$ − період сигналу CLKOUT1.

Зазвичай переривання TINT використовується для зчитування вибірок вхідних даних, що обробляються процесором. Таймер та регістр періоду можуть бути проініціалізовані в будь-якому машинному циклі. Нульове значення регістру періоду не допускається.

Якщо переривання TINT не використовуються, то необхідно застосувати маску або заборонити переривання командою DINT. У цьому випадку регістр періоду може використовуватися як звичайна комірка пам'яті. Якщо знову виникне потреба у використанні переривання TINT, то регістр PRD та таймер необхідно попередньо ініціалізувати, а потім дозволити переривання TINT.

Лічильник числа повторень RPTC містить 8-розрядне число N, яке визначає повтори деякої команди. Кількість повторів дорівнює N+1. За допомогою команди RPT або RPTK лічильник числа повторень може бути завантажений значенням з діапазону від 0 до 255. Це дозволяє повторювати команду, що знаходиться за RPT або RPTK до 256 разів.

Лічильник числа повторень може використовуватися з командами: множення з накопиченням, пересилання блоків, вводу-виводу, читання-запису таблиць. Ті команди, що потребують для виконання декількох циклів, при запису їх після команди RPT або RPTK виконуються за один цикл.

Для формування адрес при звертанні до пам'яті в ЦПОС застосовуються генератори адреси, які можуть формувати одну чи декілька адрес даних за один цикл команди, не використовуючи для цього основного арифметичного пристрою, що займається обробкою даних. Це дозволяє

обчислювати необхідні адреси даних паралельно з виконанням арифметичних операцій, що підвищує продуктивність ЦПОС

3.1.6. Архітектурні особливості основних сімейств ЦПОС (Motorola, Texas Instruments, Analog Devices)

У числі найпоширеніших сигнальних процесорів можна відзначити вироби таких компаній: Motorola (56002,96002), Intel (i960), Texas Instruments (TM5320Cxx), Analog Devices (21xx, 210xx). Велика продуктивність, що вимагається при обробці сигналів у реальному часі, спонукала дві останні з перелічених компаній випустити трансп'ютерні сімейства мікропроцесорів TMS320C4x і ADSP2106x, що орієнтовані на використання в мультипроцесорних системах [1, 23].

Вибір того або іншого процесора для реалізації конкретного проекту – багатокритеріальна задача, але слід зазначити перевагу процесорів компанії Analog Devices для додатків, що вимагають виконання великих обсягів математичних обчислень (таких, як цифрова фільтрація сигналу, обчислення кореляційних функцій і т.ін.), оскільки їх продуктивність на подібних задачах вища, ніж у процесорів компаній Motorola і Texas Instruments. У той же час для задач, що вимагають виконання інтенсивного обміну із зовнішніми пристроями (багатопроцесорні системи, різного роду контроллери), перевагу можна віддати використанню мікропроцесорів компанії Texas Instruments, які обладнані високошвидкісними інтерфейсними підсистемами.

3.1.7. Інструментальні засоби розробки систем на основі ЦПОС

У число специфічних чинників, які слід розглянути при виборі процесора ЦОС для даного додатку, входять архітектурні особливості, швидкість виконання, тип арифметики і довжина слова [23].

Архітектурні особливості. Більшість доступних зараз ЦПОС має хорошу архітектуру. Ключовими характеристиками процесорів є розмір вбудованої пам'яті, наявність спеціальних команд і можливості вводу-виводу. Наявність вбудованої пам'яті – необхідна вимога в більшості додатків ЦОС реального часу, оскільки це означає швидкий доступ до даних і швидке виконання програми. Для додатків із суворими вимогами до пам'яті (цифрове аудіо, система Dolby, факс-модем, кодування-декодування MPEG) розмір внутрішньої пам'яті ОЗП може стати вирішальним чинником при виборі процесора. Якщо внутрішньої пам'яті недостатньо, її можна доповнити високошвидкісною зовнішньою пам'яттю, хоча це може призвести до збільшення вартості системи. Для додатків, що вимагають швидких і ефективних обчислень або обміну потоками даних із зовнішнім світом,

досить важливі такі засоби вводу-виводу, як інтерфейси АЦП і ЦАП, можливість прямого доступу до пам'яті і підтримка багатопроцесорної обробки. Залежно від додатку важливий багатий набір спеціальних команд підтримки операцій ЦОС, наприклад, можливість організації циклів з нульовими службовими витратами, спеціалізовані команди ЦОС і колова адресація.

Швидкість виконання. Оскільки більшість задач ЦОС вимагає термінового розв'язання, то важливою мірою продуктивності є швидкість процесорів ЦОС. Традиційно двома основними одиницями вимірювання цієї величини є тактова частота процесора в мегагерцах або гігагерцах і число команд, що виконуються, в мільйонах команд за секунду (Million Instructions Per Second - MIPS) або, якщо використовуються процесори ЦОС з плаваючою точкою, в мільйонах операцій з плаваючою точкою за секунду (million floating-point operations per second - MFLOPS). Втім подібні методи вимірювання можуть в деяких випадках не надавати об'єктивної картини через значні відмінності в принципах роботи різних ЦПОС, більшість з яких може виконувати декілька операцій в одній машинній команді. Різні процесори також відрізняються числом операцій, що виконуються в кожному такті. Отже, порівнювати швидкості роботи процесорів на основі зазначених вище методів не коректно. Альтернативна міра порівняння заснована на швидкості виконання контрольних алгоритмів, наприклад основних алгоритмів ЦОС, таких, як ШПФ, КІХ- та БІХ-фільтрація. Вивчаючи та порівнюючи чи ці дані, можна отримати уявлення про відносну продуктивність різних популярних ЦПОС.

Тип арифметики. Критерії вибору арифметики з фіксованою або з плаваючою точками розглядалися вище.

Довжина слова. Ще одним важливим параметром в ЦОС є довжина слова даних процесора, оскільки вона може істотно впливати на якість сигналу. Цей параметр визначає, наскільки точно можна представити параметри і результати операцій ЦОС. Взагалі чим довше слово даних, тим менша помилка при цифровій обробці сигналу. У аудіообробці з фіксованою точкою, наприклад, для підтримки CD-якості довжина слова процесора повинна бути не менше 24 біт, що дозволить підтримати найменший рівень сигналу, достатньо вищий за мінімальний рівень шуму, що генерується обробкою сигналу. У процесорах ЦОС з фіксованою точкою використовуються різноманітні довжини слів процесорів, залежно від додатку. У процесорах ЦОС з фіксованою точкою, націлених на ринок телекомунікацій, звичайно використовуються слова 16-бітової довжини (наприклад, TMS320C54x), тоді як у процесорах, націлених на аудіододатки високої якості, використовуються слова довжиною 24 біта (наприклад, DSP56300). Останніми роками автори відзначають тенденцію до

використання більшого числа бітів для АЦП і ЦАП (наприклад, 24-бітовий аудіокодек Cirrus, CS4228), оскільки вартість подібних пристроїв постійно знижується. Таким чином, автори передбачають підвищення попиту на процесори для аудіообробки з великими довжинами слів. У накопичувачах процесорів з фіксованою точкою також можуть бути потрібні захисні біти (звичайно 1 – 8 біт) для запобігання арифметичному переповненню в процесі операцій множення і накопичення підвищеної точності. Додаткові біти ефективно розширюють динамічний діапазон, доступний для процесорів ЦОС. У більшості процесорів ЦПОС під арифметикою звичної точності розуміють використання 32-бітових даних (24-бітова мантиса і 8-бітова експонента). Більшість процесорів ЦОС з плаваючою точкою також дозволяє виконувати операції з фіксованою точкою і часто підтримує арифметику з фіксованою точкою із змінним розміром даних.

На практиці при виборі процесора можуть враховуватися і такі чинники, як досвід роботи з конкретним сімейством процесорів ЦОС, легкість використання, термін присутності на ринку і вартість.

Контрольні питання

1. Наведіть приклади найбільш поширених методів ЦОС.
2. Які особливості апаратної архітектури ЦОС?
3. Від яких чинників залежить точність обчислень ЦОС?
4. З яких основних структурних елементів складається ЦПОС?
5. Які типи арифметики застосовуються в ЦПОС? У чому їх особливості?
6. Як апаратно реалізується операція множення з накопиченням (MAC)?
7. Що таке гарвардська архітектура?
8. Які основні принципи вибору ЦПОС для розробки конкретного додатку?
9. ЦПОС яких фірм набули найбільшого поширення?
10. За якими критеріями порівнюються ЦПОС щодо швидкодії?

3.2. Архітектура й особливості функціонування ЦПОС із фіксованою точкою

3.2.1. ЦПОС із фіксованою точкою фірми Analog Devices (ADSP 21xx)

Мікропроцесори компанії Analog Divices представлені двома сімействами: ADSP21xx і ADSP210xx [23].

Сімейство ADSP21xx – набір однокристальних 16-розрядних мікропроцесорів із загальною базовою архітектурою, що оптимізована для виконання алгоритмів цифрової обробки сигналів та інших додатків, що вимагають високошвидкісних обчислень з фіксованою точкою. Представники сімейства відрізняються один від одного, в основному, розташованими на кристалі периферійними пристроями, такими як кеш-пам'ять, таймери, порти і т. ін.

Інше сімейство мікропроцесорів ADSP210xx, яке об'єднує 32-розрядні мікропроцесори, орієнтовані на сигнальні алгоритми, що вимагають виконання обчислень з плаваючою точкою, буде розглянуте у відповідному розділі. Мікропроцесори сімейства ADSP21xx успішно конкурують з аналогічною продукцією інших компаній-виробників сигнальних процесорів завдяки порівнянній продуктивності при нижчій ціні, а також розвиненій системі технічних і програмних засобів розробки прикладних систем.

Висока продуктивність процесорів на сигнальних алгоритмах досягається завдяки багатофункціональній і гнучкій системі команд, апаратній реалізації більшості типових для даних додатків операцій, високому ступеню паралелізму процесів в мікропроцесорі, скороченню командного такту. Мікропроцесори ADSP21xx мають модифіковану Гарвардську архітектуру, в рамках якої передбачається можливість доступу в пам'ять команд, при її фізичному розділенні з пам'яттю даних. (Аналогічну архітектуру, що стала для DSP-процесорів стандартом де-факто, мають багато інших процесорів, наприклад TMS320xxx виробництва кампанії Texas Instruments). Узагальнена структура мікропроцесора ADSP21xx наведена на рис. 3.9.

Кожен мікропроцесор цього сімейства містить три незалежні повнофункціональні обчислювальні пристрої: АЛП, MAC-помножувач з накопиченням, пристрій барабанного зсуву. Кожен пристрій безпосередньо оперує з 16-розрядними даними і забезпечує апаратну підтримку обчислень з різною точністю.

Мікропроцесор містить генератор адрес команд (PS) і два генератори адрес даних (DAG), що забезпечують адресацію до даних і команд, розташованих як у внутрішній, так і в зовнішній пам'яті.

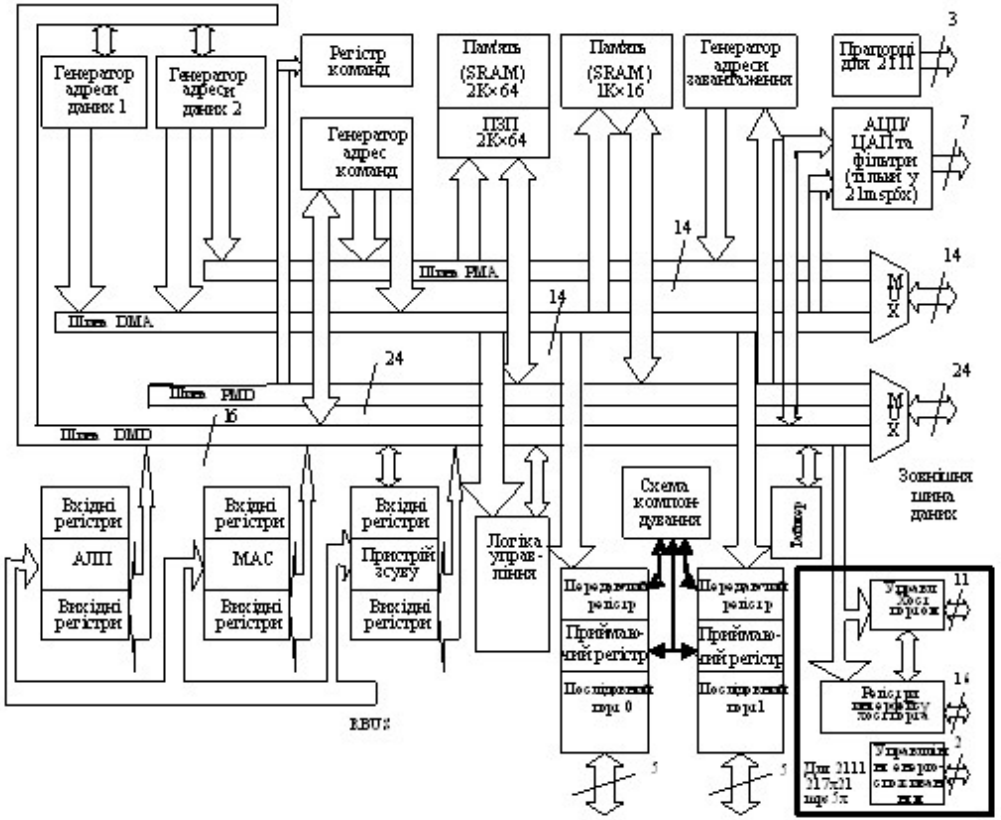


Рис. 3.9. Узагальнена структура мікропроцесора ADSP21xx

Паралельне функціонування генераторів скорочує тривалість виконання команди, дозволяючи за один такт вибирати з пам'яті команду і два операнди.

Таймер/лічильник мікропроцесора забезпечує періодичну генерацію переривань.

Послідовні порти (SPORTs) забезпечують послідовний інтерфейс з більшістю стандартних послідовних пристроїв, а також з апаратними засобами стиснення-відновлення даних, що використовують A- і μ-закони компандування.

Порт інтерфейсу з хост-процесором (HIP) дозволяє без додаткових інтерфейсних схем взаємодіяти з головним процесором системи, в якості якого може використовуватися як процесор даного сімейства, так і інший мікропроцесор, наприклад Motorola 68000 або Intel 8051.

Мікропроцесор ADSP-2181 містить внутрішній порт ПДП (IDMA) і байтовий порт ПДП (BDMA), що забезпечує швидкий обмін з внутрішньою пам'яттю. Порт IDMA підтримує асинхронний обмін з пам'яттю програм, а порт BDMA дозволяє записувати і читати як команди, так і дані.

Мікропроцесори компанії Analog Devices відрізняє високий ступінь паралелізму внутрішніх операцій. За один такт процесор може:

- генерувати адресу наступної команди;
- завантажити з пам'яті наступну команду;
- виконати 1 або 2 пересилання даних;
- відновити 1 або 2 покажчика на дані;
- виконати операцію.

Мікропроцесор, що має відповідний пристрій, може в цьому ж такті:

- прийняти та/або передати дані через послідовні порти;
- прийняти та/або передати дані хост-процесору;
- прийняти та/або передати дані через аналоговий інтерфейс.

Основні характеристики мікропроцесорів сімейства ADSP-21xx зведені в табл.. 3.2 [23].

Загальне для сімейства ADSP-21xx мікропроцесорне ядро зображене на рис. 3.10. Арифметико-логічний пристрій мікропроцесора виконує стандартний набір арифметичних і логічних операцій, включаючи розподіл. Пристрій MAC виконує операції множення із складанням (відніманням) за один такт. Пристрій зсуву здійснює арифметичні і логічні зсуви операндів, нормалізацію і експоненціальні операції. Функціональні пристрої мікропроцесора можуть обмінюватися результатами виконання операцій по шині результатів (R).

Внутрішні функціональні модулі зв'язані між собою за допомогою п'яти шин: шини адрес пам'яті даних (DMA), шини адрес пам'яті команд

Таблиця 3.2 – Основні характеристики мікропроцесорів сімейства 21xx

Можливості	2101	2103	2105	2115	2111	2171	2173	2181	2183	21mp58
АЛП	+	+	+	+	+	+	+	+	+	+
Блок МАС	+	+	+	+	+	+	+	+	+	+
Зсув	+	+	+	+	+	+	+	+	+	+
Генератор адрес даних	+	+	+	+	+	+	+	+	+	+
Генератор адрес команд	+	+	+	+	+	+	+	+	+	+
ОЗП даних	1K	1K	512	512	1K	2K	2K	16K	16K	2K
ОЗП команд	2K	2K	1K	1K	2K	2K	2K	16K	16K	2K
Таймер	+	+	+	+	+	+	+	+	+	+
Багатоканальний послідовний порт	+	+		+	+	+	+	+	+	+
Послідовний порт 1	+	+	+	+	+	+	+	+	+	+
Порт хост-інтерфейсу	-	-	-	-	+	+	+	-	-	+
Порт ПДП	-	-	-	-	-	-	-	+	+	-
Аналоговий інтерфейс	-	-	-	-	-	-	-	-	-	+
Напруга живлення, В	5	3.3	5	5	5	5	3.3	5	3.3	5
Продуктивність (MIPS)	20	10	13.8	20	20	33	20	33	33	26

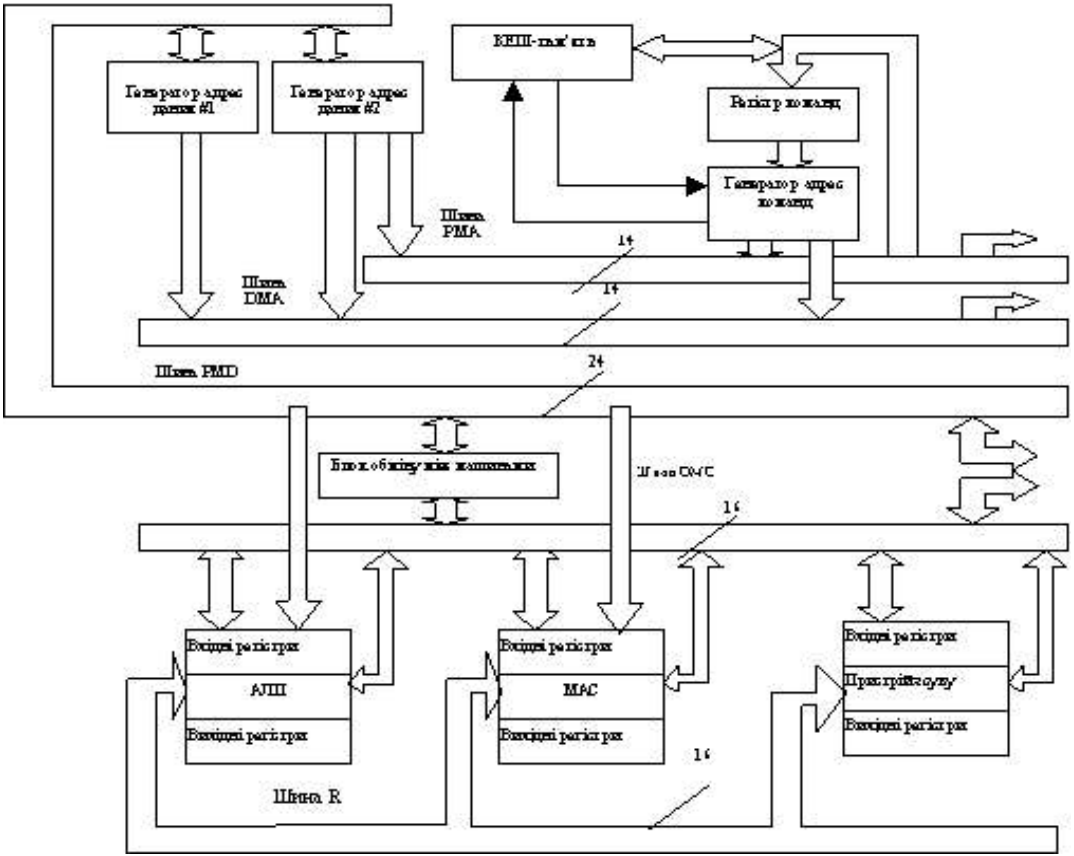


Рис 3.10. Структура мікропроцесорного ядра ADSP 21xx

(PMA), шини даних пам'яті даних (DMD), шини даних пам'яті команд (PMD) і шини внутрішніх результатів (R). Перші чотири шини мають мультиплексований зовнішній інтерфейс у вигляді шини адреси і шини даних (див. рис 3.10).

Система команд мікропроцесорів цього сімейства оптимізована для алгоритмів цифрової обробки сигналів. За системою команд всі мікропроцесори сумісні від верху до низу. Окремі представники сімейства – ADSP-2171, 2181, 21msp5x – мають додаткові і розширені команди. Кожна команда виконується за один такт. Багатофункціональні команди мікропроцесора об'єднують декілька пересилань даних з арифметико-логічною обробкою. Всі пристрої мікропроцесора є 16-розрядними і оперують з даними у форматі з фіксованою точкою. Числа подаються або як беззнакові, або в додатковому коді. Логічні операції виконуються над бітовими рядками.

Вдосконалення даного сімейства мікропроцесорів йде у напрямі підвищення тактової частоти, зниження енергоспоживання і розширення комунікаційних можливостей процесора.

3.2.2. ЦПОС із фіксованою точкою фірми Motorola (DSP 5600xx)

Сигнальні мікропроцесори компанії Motorola підрозділяються на сімейства 16- і 24-розрядних мікропроцесорів з фіксованою точкою – DSP 560xx, -561xx, -563xx, -566xx, -568xx і мікропроцесори з плаваючою точкою – DSP960xx.

Лінія 24-розрядних мікропроцесорів компанії Motorola включає два сімейства: DSP560xx і DSP563xx. Основні принципи, покладені в основу архітектури сигнальних мікропроцесорів Motoro1a, були розроблені і втілені в сімействі DSP560xx. Подальші роботи для вдосконалення сигнальних процесорів проводилися по трьох напрямах:

• нарощування продуктивності 24-розрядних процесорів за рахунок конвейєризації функціональних модулів і підвищення тактової частоти;

• створення дешевих 16-розрядних мікропроцесорів з розширеними засобами взаємодії з периферією;

• розробка високопродуктивних процесорів, що включають блок обчислень з плаваючою точкою.

Далі послідовно будуть розглянуті всі три напрями на прикладі найпопулярніших представників мікропроцесорних сімейств. Будуть вказані також найістотніші відмінності процесорів в рамках одного сімейства.

Мікропроцесори DSP56000/DSP56001 є першими представниками лінії сигнальних процесорів компанії Motoro1a. Архітектура мікропроцесорів орієнтована на максимізацію пропускної спроможності в додатках 08Р з

інтенсивним обміном даними. Це забезпечується завдяки розширеній архітектурі із складною вбудованою периферією і універсальній підсистемі вводу/виводу. Дані властивості, а також низьке енергоспоживання мінімізують складність, вартість і терміни розробки прикладних систем на базі мікропроцесорів DSP56000/DSP56001.

Мікропроцесори працюють на частотах до 33 Мгц і забезпечують продуктивність близько 16 MIPS, що дозволяє виконувати швидке перетворення Фур'е по 1024 відліках за 3,23 мс.

Відмінність між даними процесорами полягає в типі їх внутрішньої пам'яті. З метою мінімізації вартості прикладних систем мікропроцесор DSP56000 орієнтований на роботу під управлінням програми, що зберігається в НПЗП (ROM), ємністю 3,75 К слів. Існує також варіант процесора DSP56000, який володіє властивостями захисту від несанкціонованого доступу до програми, що зберігається у внутрішній пам'яті. DSP56001 містить на кристалі пам'ять довільного доступу (RAM), ємністю 512 слів, 32 слова пам'яті (ROM) програми початкового завантаження процесора із зовнішнього джерела, а також два модулі пам'яті, заздалегідь запрограмованих як таблиці функцій експандування за А- і μ-законами, і таблиці синусоїдального перетворення.

Структура мікропроцесорів DSP56000 і DSP56001 зображена на рис. 3.11 і 3.12 відповідно.

Призначення основних компонентів структури мікропроцесорів буде описане нижче, при розгляді загальної структури мікропроцесорного сімейства DSP560xx. Подальший розвиток сімейства мікропроцесорів DSP560xx здійснювався в рамках концепції процесорного ядра, загального для всіх представників сімейства, до складу якого входять 24-розрядні мікропроцесори з фіксованою точкою DSP 56002, 4, 7, 9, 11 [1, 23].

Процесори даного сімейства характеризуються високою пропускною спроможністю, розширеною розрядністю, що забезпечує високу точність обчислень, і широким динамічним діапазоном даних, що обробляються, підтримкою енергозберігаючого режиму роботи. Представники сімейства відрізняються один від одного конфігураціями пам'яті і периферійними пристроями.

Типова структура представника сімейства DSP560xx мікропроцесора наведена на рис. 3.13. Основними компонентами мікропроцесора є:

- шини даних;
- шини адрес;
- АЛП даних (ALU);
- пристрій генерації адрес (AGU);
- пристрій програмного управління (PCU);
- розширення пам'яті (порт А);

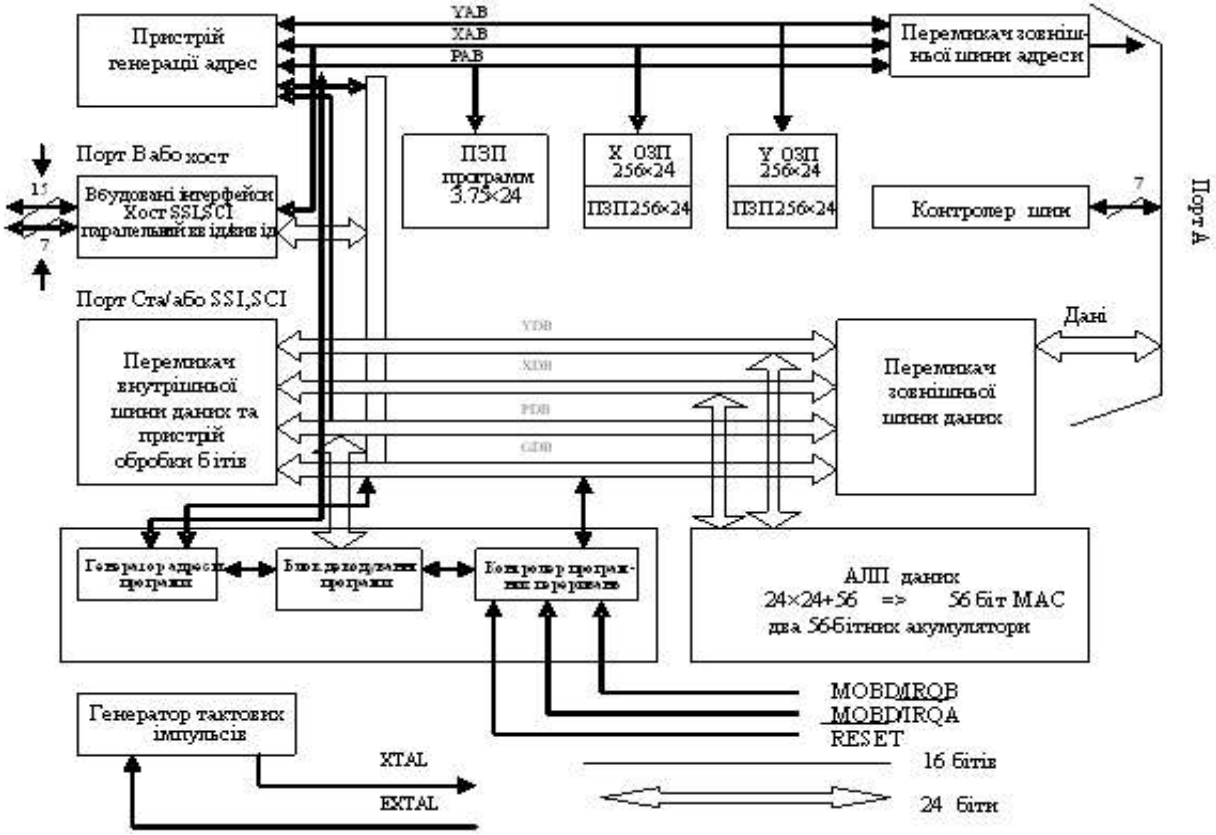


Рис. 3.11. Структура мікропроцесора DSP 56000

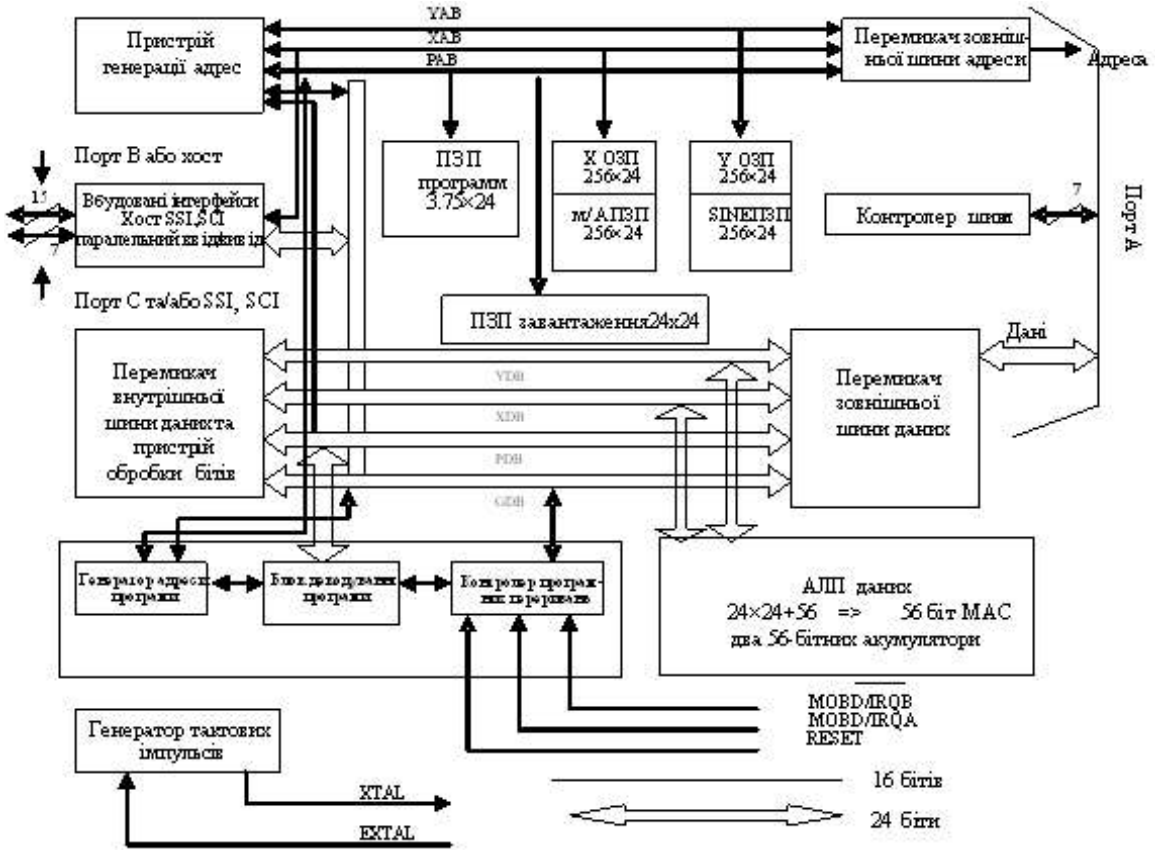


Рис. 3.12. Структура мікропроцесора DSP 56001

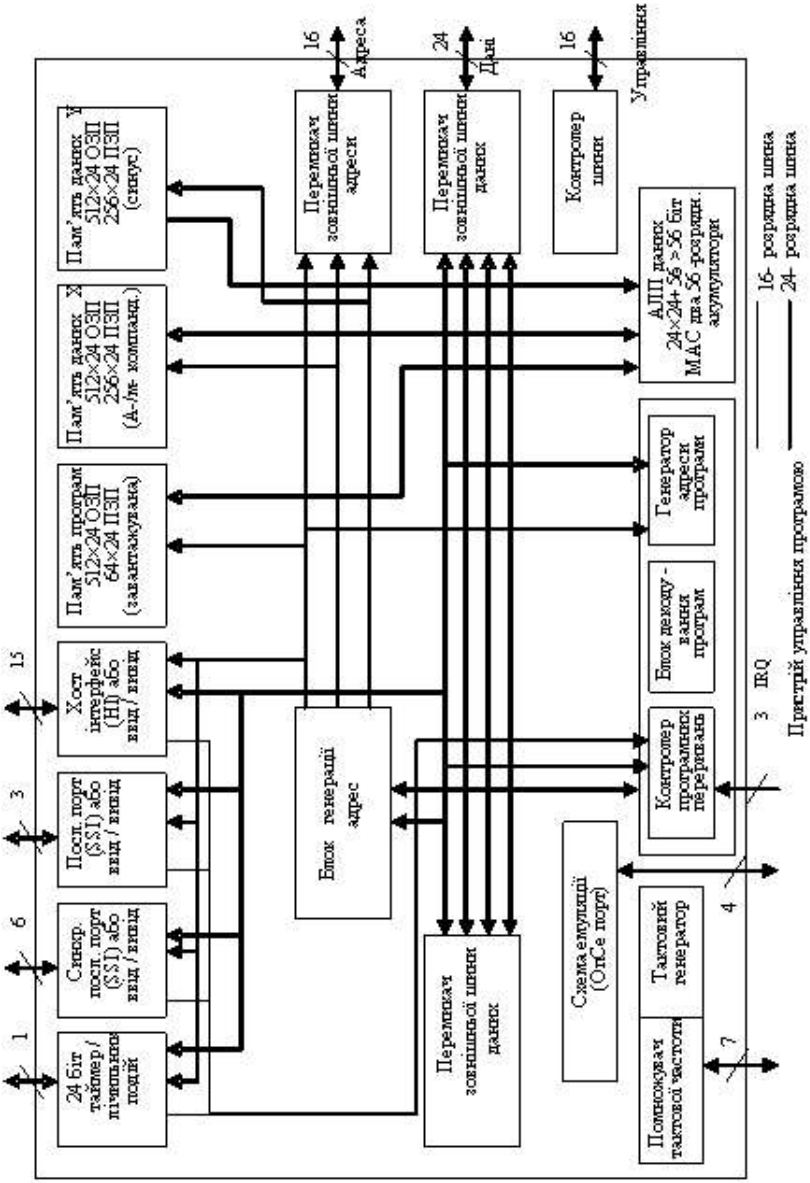


Рис. 3.13. Структура мікропроцесора сімейства DSP560xx

• внутрішньокристальна схема емуляції (OnCEдд);

• схема множення частоти.

Процесор містить три незалежні виконавчі пристрої: PCU, AGU і АЛП даних. Пересилання даних між регістрами виконавчих пристроїв здійснюється по двоспрямованих 24-розрядних шинах: шині даних X (XDB), шині даних У (YDB), програмній шині даних (PDB) і глобальній шині даних (GDB). Деякі команди використовують шини даних X і У як єдину 48-розрядну шину. Для підвищення швидкості вибірки операнди команди завантажуються в АЛП з модулів пам'яті X і У по незалежних шинах XDB і YDB, а команда – по програмній шині даних PDB. Обмін даних з периферійними пристроями здійснюється по шині GDB.

Шинна структура підтримує основні пересилання даних типу регістр-регістр, регістр-пам'ять, пам'ять-регістр. За один такт можуть бути передані два 24-бітових і одне 56-бітове слова. Обмін між шинами здійснюється через внутрішній комутатор – матрицю, що дозволяє з'єднати будь-які дві внутрішні шини без додавання тактів затримки. Адреси для внутрішніх X- і Y- пам'яті даних передаються по двоспрямованих 16-розрядних шинах XAB і YAB, а адреси пам'яті команд – по двоспрямованій програмній шині (PAB). Зовнішня пам'ять адресується за допомогою односпрямованої шини, що є виходом мультиплексора шин з трьома входами: XAB, YAB, PAB.

Пристрій бітових операцій фізично розташований в блоці комутатора, що забезпечує йому доступ до будь-якої області пам'яті і дозволяє виконувати бітові операції над даними в пам'яті, регістрах, вмістом адресних і управляючих регістрів.

АЛП даних мікропроцесора виконує над даними всі арифметичні і логічні операції і містить чотири 24-бітові регістри-джерела, два 48-бітові регістри-акумулятори, два 8-бітові регістри розширення акумуляторів, пристрій зсуву акумулятора, дві схеми зсуву/обмеження даних і паралельний (не конвеєризований) однотактовий пристрій множення з накопиченням (MAC).

Акумулятори A і B служать як буферні регістри шин XDB і YDB, а їх 8-бітові регістри розширення використовуються схемою зсуву/обмеження для фіксації і обробки ситуацій переповнення в результаті арифметичних операцій або зсуву.

АЛП даних дозволяє виконувати множення в режимі подвоєної точності (задається установленням відповідного біта в регістрі стану процесора). Результат множення двох 48-бітових операндів має 96 розрядів і міститься в 4-х 24-бітових регістрах.

Пристрій генерації адреси (AGU) працює паралельно з іншими компонентами процесора, забезпечуючи обчислення необхідних адрес даних в пам'яті за один такт за допомогою двох однакових 16-бітових

арифметичних пристроїв, кожен з яких може виконувати лінійні, модульні і циклічні арифметичні операції.

З кожним адресним АЛП зв'язані три набори з 4-х регістрів: адресних – R0-R3 і R4-R7, зсувів – N0 - N3, N4 - N7 і модифікаторів M0 -M3 і M4 - M7. Регістри використовуються трійками: RO:NO:MO, R1:N1:M1, R2:N2:M2, R3:N3:M3, R4:N4:M4, R5:N5:M5, R6:N6:M6 і R7:N7:M7. Адреса формується з вмісту адресного регістру і регістру зсуву з урахуванням типу арифметики, який визначається вмістом регістра модифікатора.

Пристрій програмного управління генерує адреси програми (попередня вибірка команд), декодує, апаратно обробляє команди циклічного переходу, внутрішні і зовнішні переривання або виняткові ситуації. Він містить 15-рівневий 32-бітовий системний стек (SS) і 6 регістрів, що безпосередньо адресуються: лічильник команд (PC), лічильник циклу (LC), регістр адреси циклу (LA), регістр стану (SR), регістр режиму (OMR) і покажчик стека (SP); 16-бітовий регістр PC може адресувати до 65536 команд. SS зберігає PC і SR при виклику процедур, обробці переривань і виконанні програмних циклів.

Команди процесора виконуються в триетапному (передвибірка, декодування, виконання) конвеєрі з подальшим аналізом 5 можливих станів процесора: "нормальний", "виключення", "скидання", "очікування" і "останов".

До складу PCU входять три блоки: блок декодування програми (PDC), генератор адреси програми (PGA) і програмний контролер переривань (PIC). PDC декодує команди, завантажені в командний буфер, і генерує всі необхідні для виконання команди управляючі сигнали. Вміст командного буфера дублюється для ефективнішого виконання команд повторення (REP) і переходу.

Основне призначення блока PGA – апаратне формування адрес циклів. При ініціалізації циклу адреса його початку поміщається в стек, значення змінної циклу міститься в регістрі LC, адреса кінця циклу – в LA. При завершенні чергової ітерації адреса переходу витягується із стека, тобто не формується програмно, що істотно підвищує швидкість обробки.

PIC одержує всі запити на переривання, класифікує їх і генерує адресу вектора переривань. Переривання можуть бути маскованими – що дорівнюють 0 (нижній рівень), 1, 2, і немаскованими – рівень 3 (вищий рівень).

Порт розширення пам'яті A забезпечує синхронний обмін даними з різними типами пам'яті і зовнішніми пристроями по 24-розрядній шині даних. Порт працює з високо- і низькошвидкісною пам'яттю, а також іншими універсальними і сигнальними процесорами в режимі master/slave.

Внутрішньокристальний емулятор – це схема, що дозволяє інтерактивно аналізувати стан регістрів, пам'яті, периферійних пристроїв і управляти процесом відлагодження програми – дозволяти відлагодження для розробника системи або забороняти іншим користувачам доступ до внутрішніх ресурсів процесора.

Помножувач частоти дозволяє процесору працювати на підвищеній внутрішній тактовій частоті, забезпечуючи синхронізацію внутрішніх і зовнішніх тактових імпульсів, а також зниження частоти в енергозберігаючому режимі.

Програмна модель мікропроцесора подається у вигляді трьох діючих паралельно функціональних пристроїв: ALU, AGU і PCU. Система команд орієнтована на ефективну підтримку мови С і організована так, щоб забезпечити зайнятість цих пристроїв протягом кожного такту, досягаючи при цьому максимальної швидкості виконання програми.

Команди мікропроцесора мають змінну довжину: 1 або 2 24-бітових слова. Типова команда мікропроцесора містить поле коду операції, що визначає відповідну дію ALU, AGU або PCU, поле операндів і два поля, що задають пересилання, які виконуються паралельно з основною операцією по шинах XDB і YDB. Приклад команди MAC наведено в табл. 3.3.

Таблиця 3.3 – Структура команди мікропроцесора DSP-560xx

Opcode	Operands	XDB	YDB
MAC	XO.YO.A	X:(RO)+,XO	У:(R4)+,YO

3.2.3. ЦПОС із фіксованою точкою фірми Texas Instruments (TMS 320 C2x)

Сигнальні процесори компанії Texas Instruments [1] поділяються на два класи: це процесори для обробки чисел з фіксованою та плаваючою точками (рис. 3.14.). Перший клас представлений трьома сімействами процесорів, базовими моделями яких є відповідно TMS320.10, .20, .50. Другий клас включає процесори TMS320.30, .40, TMS320C80, які підтримують операції з плаваючою точкою і які є мультипроцесорною системою, виконаною на одному кристалі, а сімейство TMS320C6x включає процесори як з фіксованою так і з плаваючою точками.

Процесори старших поколінь одного сімейства успадковують основні архітектурні особливості і є сумісними від низу до верху за системою команд (чого не можна сказати про процесори, що входять в різні сімейства).

Модифікована гарвардська архітектура TMS 320 C2x.

Перший процесор сімейства TMS320C10 був випущений в 1982 р. і завдяки ряду вдалих технічних рішень набув великого поширення [1].

Структура типового представника сімейства – мікропроцесора TMS320C15 наведена на рис. 3.15.

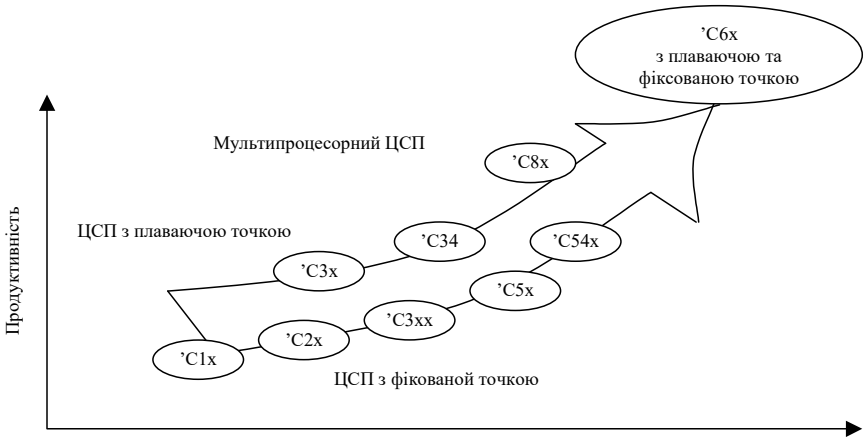


Рис 3.14. Сімейства мікропроцесорів компанії TI

У основі мікропроцесора лежить модифікована гарвардська архітектура, відмінністю якої від традиційної гарвардської архітектури є можливість обміну даними між пам'яттю програм і пам'яттю даних, що підвищує гнучкість пристрою.

TMS320C10 є 16-розрядним процесором. Його адресний простір складає 4К 16-розрядних слів пам'яті програм і 144 16-розрядних слів пам'яті даних. Тривалість командного такту процесора складає 160 – 200 нс.

Арифметичні функції в процесорі реалізовані апаратно. Він має апаратний помножувач (MULT), пристрій зсуву (SHIFTER), апаратурну підтримку автоінкремента/декремента адресних регістрів даних (AR0, AR1)

Із зовнішніми пристроями процесор взаємодіє через 8 16-розрядних портів вводу/виводу. Передбачена обробка зовнішнього переривання.

Інші мікропроцесори даного сімейства (С14-С17) мають аналогічну архітектуру і відрізняються тривалістю командного такту, конфігурацією пам'яті, наявністю (або відсутністю) додаткових периферійних пристроїв (наприклад, в С17-кодек даних за μ-/А-законом, перетворювач логарифмічної імпульсно-кодової модуляції (ІКМ) в лінійну ІКМ).

Мікропроцесори сімейства TMS320C2x аналогічні за архітектурою, але мають підвищену продуктивність і ширші функціональні можливості [1]. Всі процесори цього сімейства можуть використовувати по 64К слів пам'яті

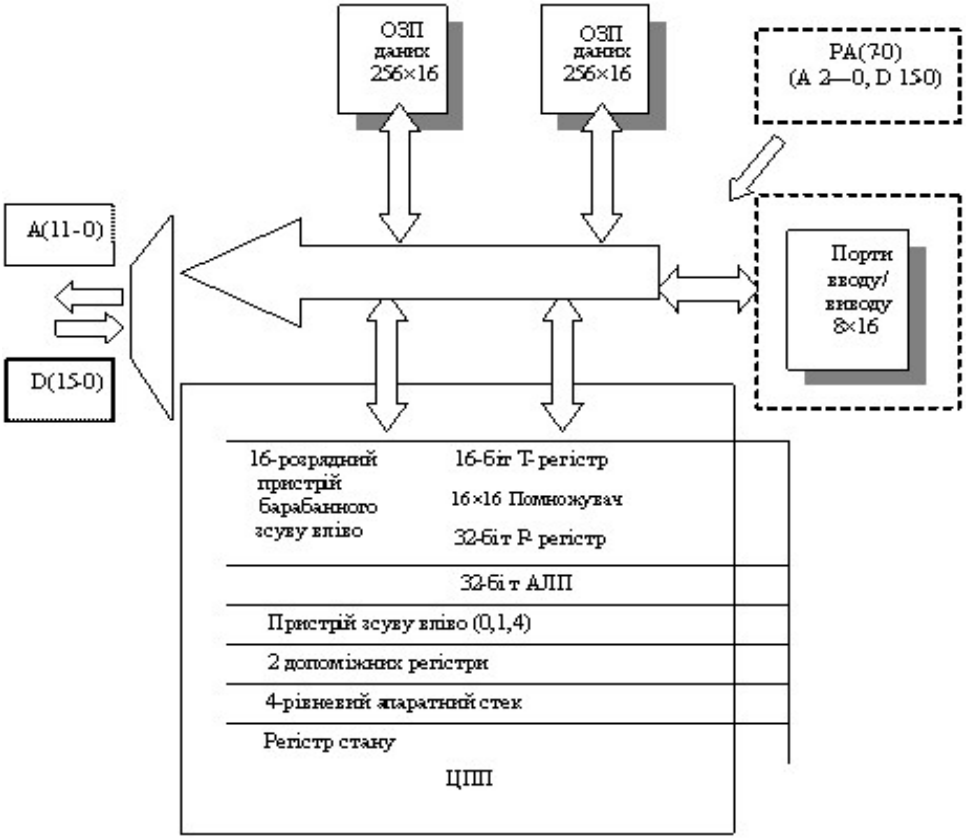


Рис. 3.15. Структура мікропроцесора сімейства TMS320C1x

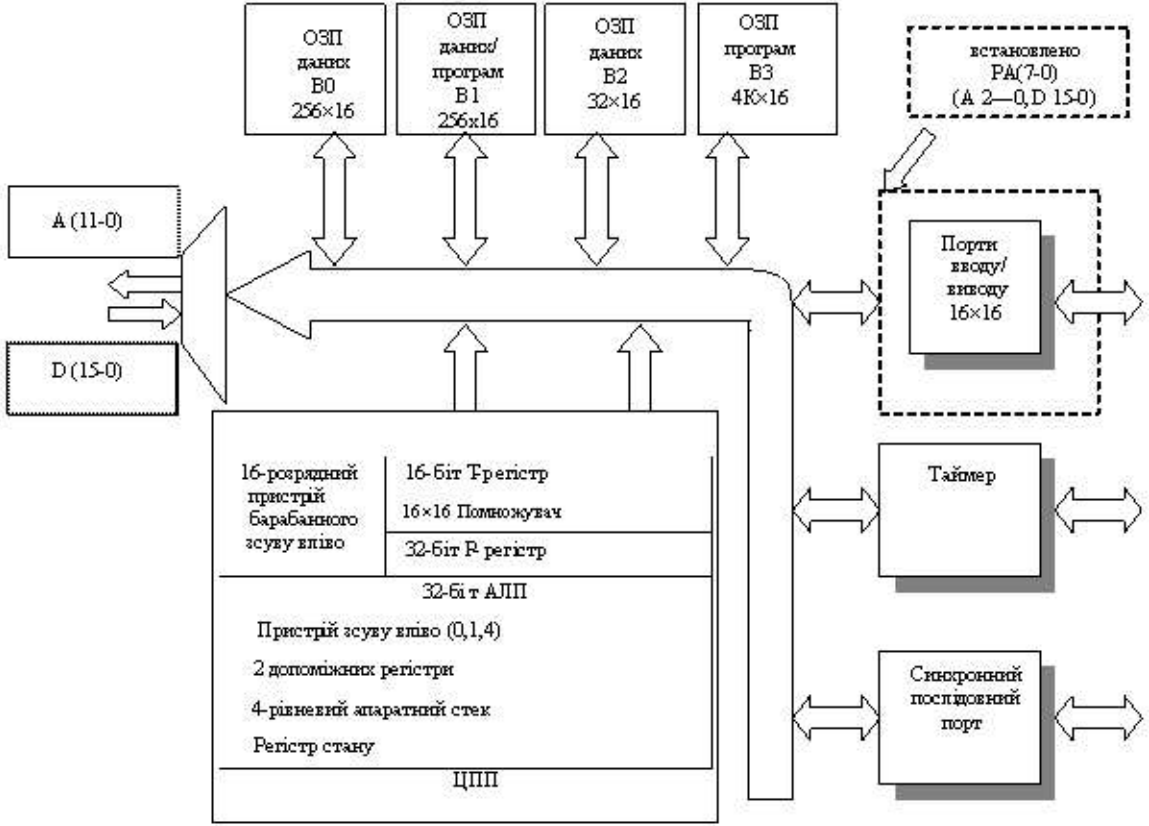


Рис.3.16. Структура мікропроцесора TMS320C2x

Таблиця 3.4 – Основні характеристики процесорів сімейства TMS320C2x

Мікропроцесор	Технологія	Командний цикл	Внутрішня пам'ять			Зовнішня пам'ять		Ввід/вивід		
		(нс)	ОЗП	ПЗП	НПЗП	Дані	Програма	Послідовний порт	Паралельний порт	Прямий доступ до пам'яті (ПДП)
TMS32020	NMOS	200	544			64K	64K	1	16x16	Так
TMS320C25	CMOS	100	544	4K		64K	64K	1	16x16	Так
TMS320C25-50	CMOS	80	544	4K		64K	64K	1	16x16	Так
TMS320C25	CMOS	100	544		4K	64K	64K	1	16x16	Так
TMS320C26	CMOS	100	1568	256		64K	64K	1	16x16	Так

програм і даних, мають по 16 16-розрядних портів вводу/виводу і послідовний порт.

Структура мікропроцесора TMS320C2x наведена на рис. 3.16.

Процесори сімейства TMS320C2x мають можливість використовувати зовнішній контролер ПДП. Помножувач мікропроцесорів, крім операцій множення, дозволяє виконувати за один такт зведення в квадрат. У процесори включена апаратна підтримка кратного виконання команди, реалізований режим двійкової непрямої адресації, призначений для ефективної реалізації ШПФ.

Основні технічні характеристики процесорів 2-го покоління наведено в табл.. 3.4.

Основні відмінності архітектури процесорів TMS3202x від TMS3201x полягають в такому:

• виконання множення і збереження результатів в TMS3202x здійснюється за один командний цикл;

• набір команд підтримує обчислення з плаваючою точкою;

• є внутрішній маскований ПЗП програм (ROM), розміром 4К слів для TMS320C25 або ПЗП з ультрафіолетовим стиранням (EPROM) 4К слів для TMS320E25;

• виконання програм здійснюється з пам'яті програм RAM, розташованої на кристалі. Обсяг пам'яті програм RAM становить 544 слова, з яких 256 можуть бути використані як пам'ять даних;

• розширена зовнішня пам'ять може мати обсяг 128К слів (64К слів – пам'ять програм, 64К – пам'ять даних);

• TMS3202x містить зовнішній інтерфейс для організації багатопроцесорних зв'язків і засобу синхронізації для доступу до пам'яті, що розділяється;

• можливість обміну пам'яті даних і програм блоками;

• можливість організації циклів очікування при доступі до повільної зовнішньої пам'яті або до повільних периферійних пристроїв;

• TMS3202X містить на кристалі таймер і послідовний порт;

• наявність п'яти (TM532020) або восьми (TM5320C25) допоміжних регістрів і спеціального арифметичного пристрою для них;

• наявність апаратного стека розміром 4 слова для TM832020 або 8 слів для TM5320C25 і можливості програмного розширення стека в пам'яті даних;

• наявність команд обробки бітових даних;

• наявність трьох маскованих користувачем переривань;

• наявність режиму прямого доступу до пам'яті (ПДП) (тільки для TMS320C25).

Архітектура й призначення основних модулів TMS320C2x - ЦАЛП, ДАП, лічильника команд.

Центральний арифметико-логічний пристрій (ЦАЛП) містить 16-розрядний регістр із масштабуванням та зсувом, паралельний помножувач 16x16, 32-розрядний арифметико-логічний пристрій (ALU), 32-розрядний акумулятор (ACC) та додаткові зсувні пристрої на виходах помножувача та акумулятора.

Виконання команди в ЦАЛП звичайно складається з таких дій:

• отримання даних з пам'яті через шину даних;

• обробка даних в регістрі з масштабуванням та зсувом, помножувачі, ALU;

• збереження результатів в акумуляторі.

Один з виходів ALU з'єднаний з виходом акумулятора ACC, інший може бути підключений до виходу регістру із масштабуванням та зсувом, або до виходу помножувача. Регістр із масштабуванням та зсувом зв'язує шину даних з ALU та забезпечує зсув даних вліво на 16 розрядів (SFL(0-16). Молодші біти слова, що зсувається, заповнюються нулями або бітом знака (SX), якщо встановлено біт режиму розширення знака регістру стану ST1.

ЦАЛП містить додатковий пристрій зсуву на виході помножувача, який дозволяє виконати зсув вправо на шість розрядів (SFR(6) з урахуванням біта знака SX, або вліво на 1 або 4 розряди (SFL(1,4). Крім цього вміст акумулятора може бути зсунуто вліво на 7 розрядів при виводі даних.

ALU забезпечує виконання великого набору арифметичних та порозрядних логічних операцій за один командний цикл, оперуючи при цьому 16-розрядними словами, що надходять з пам'яті даних чи безпосередньо з команди.

ALU підтримує операції з плаваючою точкою. До системи команд процесора включено команду NORM, яка дозволяє нормалізувати число з фіксованою точкою, що міститься в акумуляторі. Це досягається шляхом вилучення зайвих знакових бітів. Команда NORM реалізується за допомогою зсуву вліво. Команда LAST дозволяє виконати зворотну операцію – денормалізацію числа з плаваючою точкою за рахунок зсуву мантиси. В цьому випадку кількість зсувів задається чотирма молодшими бітами Т-регістру. Операція зсуву виконується в регістрі із масштабуванням та зсувом. Вміст Т-регістра визначає значення експоненти.

Програмування режиму переповнення акумулятора може виконуватися командами SOVM та ROVM, які встановлюють або скидають відповідно біт режиму переповнення OVM регістру стану ST1. Якщо OVM=1 та відбувається переповнення, то акумулятор звантажується максимальним додатним числом (7FFF FFFFh) або мінімальним від'ємним числом (8000 0000h) залежно від напрямку переповнення. При OVM=0

результати в акумуляторі не модифікуються. Логічні операції не скидають прапор переповнення.

ALU та акумулятор дозволяють виконувати широкий набір команд безумовного та умовного переходів, наприклад команду переходу при виникненні переповнення BV, команду BZ, яка перевіряє, чи дорівнює нулю вміст акумулятора, команду переходу за адресою, що міститься в акумуляторі, та ін.

Акумулятор процесора складається з двох 16-розрядних регістрів: ACCH (старше слово акумулятора) і ACCL (молодше слово акумулятора). Пристрій зсуву на виході акумулятора забезпечує зсув на 7 розрядів вліво. Зсув виконується під час передачі даних до пам'яті і тому не впливає на вміст акумулятора.

Акумулятор містить біт переносу C, який дозволяє більш ефективно виконувати арифметичні операції з підвищеною точністю. Для цього можуть використовуватися команди ADDC (складання з урахуванням переносу) та SUBB (віднімання з урахуванням позики). Біт переносу може змінювати значення під впливом багатьох арифметичних команд та команд зсуву. На значення біта переносу не впливають логічні команди, команди завантаження акумулятора, команди множення MPY, MPYK, MPIU. Значення біта переносу C аналізується в командах умовного переходу BC та BNC. Біт переносу C бере участь в операціях зсуву акумулятора SFL, SFR, ROL, ROR.

У ЦПОС використано апаратний помножувач 16x16 біт, який дозволяє отримувати 32-розрядний результат з урахуванням чи без урахування знака операндів команди множення. Помножувач може обчислювати добуток двох операндів, що надійшли з пам'яті даних та пам'яті програм або тільки з пам'яті даних. Результат виконання операції множення поміщається в P-регістр. При передачі вмісту P-регістру в АЛП чи пам'ять даних може виконуватися операція зсуву добутку вліво на 1 або 4 біта та вправо на 6 біт. Конкретна кількість розрядів, на яку зсувається добуток, залежить від значення 2-розрядного поля PM регістру стану ST1. Зсув вправо на 6 біт дозволяє реалізувати 128 команд множення з накопиченням результатів в акумуляторі, що усуває небезпеку виникнення переповнення.

Для управління помножувачем використовуються команди: LT – завантаження T-регістру вмістом завданої комірки пам'яті даних; MPY – множення операндів з T-регістру та пам'яті даних; MPYK – обчислення добутку вмісту T-регістру та константи, що задана в команді. Крім цього до системи команд ЦПОС входять команди MAC, MACD, MPYA, MPYS, які управляють як помножувачем, так і АЛП з акумулятором. Ці команди дозволяють виконати множення з одночасним накопиченням результатів в акумуляторі. При виконанні команд MAC та MACD операнди отримуються з пам'яті програм за один цикл. Це дозволяє командам MAC та MACD

виконуватися також за один цикл, якщо вони використовуються сумісно з командами RPT або RPTK.

ЦПОС також містить додатковий арифметичний пристрій (ДАП), до якого входять: допоміжні регістри AR0 – AR7, покажчик допоміжного регістру ARP, буфер допоміжного регістру ARB, арифметичний пристрій допоміжних регістрів (ARAU), покажчик сторінок DP, мультиплексори MUX.

Дев'ятирозрядний регістр сторінок DP використовується при прямій адресації пам'яті даних. Для цього сім молодших розрядів слова команди об'єднуються із вмістом DP.

Допоміжні регістри AR0 – AR7 використовуються при непрямій адресації пам'яті даних. При цьому вміст допоміжних регістрів розглядається як 16-розрядна адреса пам'яті даних. Для вибору одного з допоміжних регістрів застосовується 3-розрядний регістр-покажчик ARP, до якого завантажується значення від 0 до 7. Процесор має спеціальні команди для первинного завантаження регістрів AR0 – AR7 та ARP. Покажчик ARP може бути завантажений значенням або з пам'яті даних (через шину даних), або з трьох молодших розрядів команди (через шину команд). Кожного разу, коли ARP завантажується новим значенням, старе значення зберігається в буфері ARB (за виключенням команди LST).

Пристрій ARAU дозволяє після первинного завантаження виконувати індексацію допоміжного регістру, на який вказує покажчик ARP. Індексація виконується шляхом збільшення (зменшення) вмісту ARn на одиницю, або шляхом складання (віднімання) вмісту AR0 з вмістом ARn. Індексація здійснюється паралельно з операціями, що виконуються в ЦАЛП.

Хоча пристрій ARAU розроблено для підтримки маніпуляцій з адресами, він може також використовуватися як арифметичний пристрій загального призначення. ARAU оперує 16-розрядними аргументами без знака, а також забезпечує реалізацію команди передачі управління (BANZ), що заснована на порівнянні вмісту AR0 з допоміжним регістром, на який вказує ARP. Це дозволяє реалізовувати за допомогою ARAU циклічні алгоритми. У цьому випадку один з допоміжних регістрів застосовується як лічильник циклів, а AR0 містить його кінцеве значення.

Управління обчислювальним процесом в ЦПОС здійснюється за допомогою лічильника команд (PC), апаратного стека, сигналів переривань та скидань, регістрів станів, таймера лічильника числа повторень.

ЦПОС містить 16-розрядний лічильник команд (PC) та 8-рівневий апаратний стек. Лічильник команд адресує пам'ять програм. Стек використовується під час обробки переривань та виклику підпрограм.

Лічильник команд формує адресу чергової команди та передає її на шину адреси команд. Команди з пам'яті програм завантажуються до регістру

команди (IR). В момент завантаження регістра IR лічильник команд містить адресу наступної команди. Лічильник команд може також адресувати пам'ять даних. Це відбувається під час виконання команди BLKD, яка виконує пересилання блоків даних до пам'яті даних.

Лічильник команд зв'язаний з шиною даних через мультиплексор (MUX) та може бути завантажений вмістом акумулятора. Це використовується в командах передачі управління за адресою, що знаходиться в акумуляторі (BACC, CALA). На початку нового циклу вибірки команди вміст лічильника команд збільшується на одиницю, або в нього завантажується адреса, за якою слід передати управління.

Якщо відбувається виклик підпрограм або обробка переривань, то вміст лічильника команд зберігається у стеку. Для роботи із стеком використовуються команди PUSH (занести до стека) та POP (вилучити із стека). Коли вміст лічильника команд заноситься у верхівку стека, старе значення верхівки переміщується на рівень нижче; значення, що знаходилося внизу стека, втрачається. Старе значення верхівки стека може бути втрачене, якщо до вилучення його із стека команда PUSH буде виконана 8 разів. При вилученні значень із стека може статися зворотне переповнення стека, якщо виконати команду POP більш ніж 7 разів. В цьому випадку на всіх рівнях стека буде знаходитися те саме значення.

ЦПОС містить дві додаткові команди (PUSHD та POPD), які дозволяють зберегти вміст верхівки стека у пам'яті даних та вилучати його з пам'яті даних. Ці команди забезпечують організацію стека довільної глибини у пам'яті даних.

У ЦПОС TMS320C25 кожна команда виконується впродовж трьох машинних циклів: попередньої вибірки, декодування та виконання. Лічильник попередньої вибірки (PFC) містить адресу наступної команди, яка буде обрана. Ця команда завантажується до регістру команд (IR). Якщо він зайнятий командою, що виконується, то попередньо обрана команда зберігається в регістрі черги команд (QIR). Далі інкрементується вміст PFC, і після виконання поточної команди вміст QIR завантажується в IR. Таким чином, лічильник команд, хоча і містить адресу наступної команди, але безпосередньо в операціях вибірки не використовується, а по суті є покажчиком поточної команди програми. Вміст PC інкрементується після того, як завершиться виконання команди.

Наявність 3-етапного конвеєра дозволяє в кожному машинному циклі обробляти три команди, що знаходяться на етапах виконання. Конвеєр стає 2-етапним, якщо програма, що виконується, зчитується із внутрішнього ОЗП. При звертанні до внутрішнього ОЗП вибірка та декодування команди виконуються в тому ж самому машинному циклі.

Організація пам'яті TMS320C2x і розподіл адресного простору. Процесори сімейства TMS320C2x забезпечують роботу з трьома адресними просторами: пам'яті даних, пам'яті програм та вводу-виводу. Процесор може виконувати дії над вмістом внутрішньої (розташованої на кристалі) пам'яті або над вмістом зовнішньої пам'яті (рис. 3.16).

Внутрішня пам'яті даних представлена трьома блоками ОЗП (B0, B1, B2) та шістьма регістрами (DRR, DXR, TIM, PRD, IMR, GREG). Ємність ОЗП складає 544 16-розрядних слова. З них 256 слів (блок B0) можуть використовуватися або як пам'ять даних, або як пам'ять програм. Інші 288 слів (блоки B1, B2) завжди використовуються як пам'ять даних. Загальна ємність пам'яті даних складає 64К 16-розрядних слова. Адреси, що відповідають внутрішній пам'яті даних, не перевищують 1024 (0400h). Адреси регістрів, що відображаються на пам'ять даних наведено в табл.. 3.5.

Таблиця 3.5 – Адреси регістрів

Регістр	Адреса	Призначення
DRR(16)	0	Регістр прийому послідовного порту
DXR(16)	1	Регістр передачі послідовного порту
TIM(16)	2	Таймер
PRD(16)	3	Регістр періоду таймера
IMR(6)	4	Регістр маски переривань
GREG(8)	5	Регістр розподілу глобальної пам'яті

Внутрішня пам'ять програм представлена ПЗП ємністю 4Кх16 та блоком B0, якщо його розподілено на пам'ять програм командою CNFP. Внутрішня пам'ять програм дозволяє процесору функціонувати з максимальною швидкодією, при цьому зовнішня шина даних D15 – D0 залишається вільною для звертання до пам'яті даних. Після надходження сигналу $\overline{RS}$ блок B0 відображається на пам'ять даних.

Розподіл перших 4К слів пам'яті програм задається сигналом $MP/\overline{MS}$. Якщо сигнал $MP/\overline{MS}$=1, то зазначений адресний простір відводиться під зовнішню пам'ять програм. Якщо $MP/\overline{MS}$=0, то адреси від 32 до 4015 відводяться під ПЗП програм. Звертання до зовнішньої пам'яті програм/даних здійснюється при активних сигналах $\overline{PS}/\overline{DS}$, $\overline{STRB}$. Під час звертання до внутрішньої пам'яті ці сигнали неактивні.

Дії, що виконуються за тією чи іншою командою, суттєво залежать від пам'яті, що використовується. При цьому можливі чотири комбінації видів пам'яті: PI/DI, PI/DE, PE/DI, PE/DE. З урахуванням того, що внутрішня пам'ять може бути представлена ПЗП (PI) чи блоком B0 (PR), отримуємо шість різних комбінацій. Ці обставини необхідно мати на увазі при визначенні кількості циклів, що потрібні для виконання команди.

Реалізація конвеєра команд і апаратної адресації в TMS320C2x. У ЦПОС TMS320C2x з метою скорочення тривалості командного циклу та підвищення продуктивності використано модифіковану гарвардську архітектуру, яка дозволяє здійснювати обмін даними між пам'яттю програм і пам'яттю даних та конвеєрний режим роботи, що дає можливість обробляти одночасно декілька команд.

В архітектурі процесорів цього сімейства використано 3-етапний конвеєр. Процесор може одночасно обробляти 3 команди, які знаходяться на різних етапах виконання: попередня вибірка, декодування, виконання. Наприклад, коли здійснюється вибірка *n*-ї команди, то попередня команда (*n*-1) знаходиться на етапі декодування, команда (*n*-2) – на етапі виконання. Конвеєрний режим роботи процесора залишається для користувача непомітним, за виключенням команд передачі управління, коли необхідне перезавантаження конвеєра.

При звертанні до зовнішньої пам’яті програм чи даних адресація або зчитування даних виконується за допомогою однієї зовнішньої шини адреси (A15 – A0) та однієї зовнішньої шини даних (D15 – D0) (рис. 3.16). Тому звертання до зовнішньої пам’яті програм та даних можуть виконуватися тільки послідовно. З метою підвищення швидкодії процесора та можливості паралельного доступу до пам’яті програм та пам’яті даних до складу процесора включено внутрішньокристалічні ОЗП даних/програм та ПЗП програм.

Адреси внутрішньокристалічного ОЗП, що відповідають блоку (B0), можуть бути розподілені на пам’ять даних, або на пам’ять програм. Для цього відповідно використовуються команди CNFD/CNFP. Програми, що зберігаються у блоці B0, виконуються з максимально можливою швидкістю. Блоки B1 та B2 завжди використовуються як пам’ять даних. До внутрішньокристалічного ОЗП даних відносять також шість регістрів, розподілених на адресний простір пам’яті даних, регістри послідовного порту (DRR та DXR), таймер (TIM), регістр періоду таймера (PRD), регістр маски переривань (IMR) та регістр розподілу глобальної пам’яті (GREG).

Контрольні питання

1. Що таке модифікована гарвардська архітектура?
2. Перелічити основних виробників ЦПОС з фіксованою точкою та основні сімейства цих процесорів.
3. У чому полягають основні відмінності між ЦПОС сімейств ADSP21xx та ADSP210xx?
4. Склад та призначення основних модулів ЦПОС сімейства ADSP21xx.

5. Склад та призначення основних модулів мікропроцесорного ядра ЦПОС сімейства ADSP21xx.

6. Склад та призначення основних модулів ЦПОС сімейств DSP56000 та DSP56001.

7. Склад та взаємодія між основними модулями ЦПОС сімейства DSP560xx.

8. Архітектура та призначення центрального арифметичного пристрою ЦПОС TMS320C2x.

9. Склад та призначення додаткового арифметичного пристрою ЦПОС TMS320C2x.

10. Організація пам'яті ЦПОС TMS320C2x і розподіл адресного простору.

11. Призначення та принцип дії конвеєра команд ЦПОС TMS320C2x.

12. Організація апаратної адресації в ЦПОС TMS320C2x.

3.3. Архітектура ЦПОС із плаваючою точкою ADSP-2106x

3.3.1. Супергарвардська архітектура (Super Harvard Architekture) ADSP-2106x

Представники сімейства ADSP 21xxx – ADSP 21060 і ADSP-21062 – високопродуктивні сигнальні процесори, що працюють на частоті 40 Мгц і що мають швидкодію до 80 MIPS і 120 MFLOPS [23]. Будучи схожими з раніше розглянутими ADSP-210xx за структурою мікропроцесорного ядра і сумісні знизу-вгору за системою команд, ці процесори мають деякі дуже істотні доповнення, орієнтовані перш за все на розширення комунікаційних можливостей.

Мікропроцесори сімейства ADSP-2106x мають другу назву – SHARC, пов'язану з особливостями їх архітектури (Super Harvard Architecture Computer), яка задає новий стандарт інтеграції сигнальних процесорів в мультипроцесорну систему. У SHARC-мікропроцесорі об'єднані високоефективне процесорне ядро, що виконує обробку даних у форматі з плаваючою точкою, інтерфейс з хост-процесором, контролер ПДП, послідовні порти і шина, що розділяється. Узагальнена архітектура ADSP-2106x наведена на рис. 3.17.

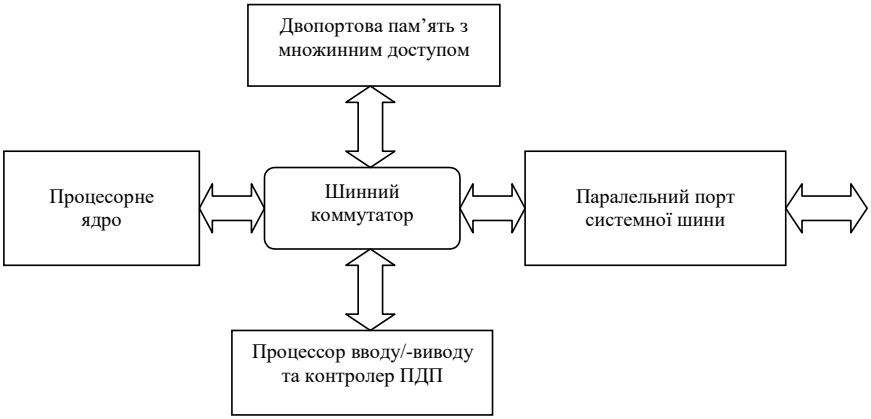


Рис. 3.17. Узагальнена архітектура ADSP-2106x (SHARC-архітектура)

Архітектуру SHARC створено на основі ядра цифрового сигнального процесора ADSP -21000. Для формування однокристальної мікро-ЕОМ до нього додаються двопортова пам’ять (статичне ОЗП), що знаходиться на кристалі, процесор вводу/виводу, контролер ПДП, паралельний порт системної шини та ряд інших пристроїв.

SHARC-архітектура є прикладом гармонійного поєднання принципів побудови розподілених і зв'язаних систем, що об'єднують в собі простоту і ефективність масштабування розподілених систем із зручністю програмування систем з пам'яттю, що розділяється.

Обчислювальні модулі на базі мікропроцесорів ADSP-2106x випускаються у вигляді мікропроцесорних кластерів, що містять від 3 до 8 вузлів у вигляді плат з шинним інтерфейсом ISA, PCI або VME, а також у вигляді модулів SHARCPAC і TRANSPAC – мезонінної плати, що встановлюється в спеціальні роз'єми материнських плат сигнальних процесорів.

Розроблено два ADSP SHARC-процесори: ADSP-21060 та ADSP-21062. Перший містить 4 Мегабіти ОЗП на кристалі, другий – 4 Мегабіти. У всьому іншому ці процесори однакові і є функціонально сумісними з ADSP -21020. Розширена структурна схема мікропроцесора ADSP-21060 SHARC наведена на рис. 3.18. До його складу входять: цифровий сигнальний процесор (ядро), двопортовий статичний ОЗП, процесор вводу/виводу та інтерфейсний процесор (зовнішній порт).

3.3.2. Архітектура мікропроцесорного ядра в ADSP-2106x

Мікропроцесорне ядро ADSP-2106x складається з таких блоків:

• 32-розрядного обчислювального пристрою для виконання операцій з плаваючою точкою відповідно до стандарту IEEE, який складається з помножувача, АЛП та багаторегістрового пристрою циклічного зсуву;

• блока регістрів загального призначення (файла даних);

• двох генераторів адреси: ГА-1 та ГА-2;

• контролера мікропрограм;

• кеш-пам'яті команд;

• таймера;

• комутатора шин.

ADSP-2106x SHARC містить три незалежних пристрої, що здійснюють обчислювальні операції: АЛП, помножувач з акумулятором та пристій зсуву. Обчислювальні пристрої обробляють дані у трьох форматах: 32-бітні слова з фіксованою точкою, 32-бітні слова та 40-бітні слова з плаваючою точкою. Операції над числами з плаваючою точкою є IEEE-сумісними; 32-розрядний формат з плаваючою точкою є стандартним IEEE-форматом; 40-розрядний формат з плаваючою точкою є IEEE-форматом з розширеною точністю, який має вісім додаткових бітів мантиси для забезпечення більшої точності обчислень.

АЛП виконує стандартний набір арифметичних та логічних операцій як у форматі з фіксованою точкою, так і у форматі з плаваючою точкою. Помножувач здійснює множення з плаваючою та фіксованою точками, а також сумісне множення/додавання та множення/віднімання з фіксованою точккою. Пристрій зсуву здійснює арифметичні та логічні зсуви, маніпуляції з бітами, копіювання полів, операції виділення (маскування) та нормалізацію 32-бітних операндів. Обчислювальні пристрої виконують операції за один цикл без використання конвеєра. Вихід якого-небудь обчислювального пристрою може бути входом будь-якого іншого в наступному циклі. При множенні АЛП та помножувач виконують незалежні операції одночасно.

Файл регістрів даних загального призначення використовується для пересилання даних між обчислювальними пристроями та шинами даних і для збереження проміжних результатів. Файл регістрів даних має два набори регістрів (основний та додатковий), до 16 регістрів у кожному, для використання швидкого контекстного переключення при застосуванні мультизадачного режиму. Всі регістри 40-розрядні.

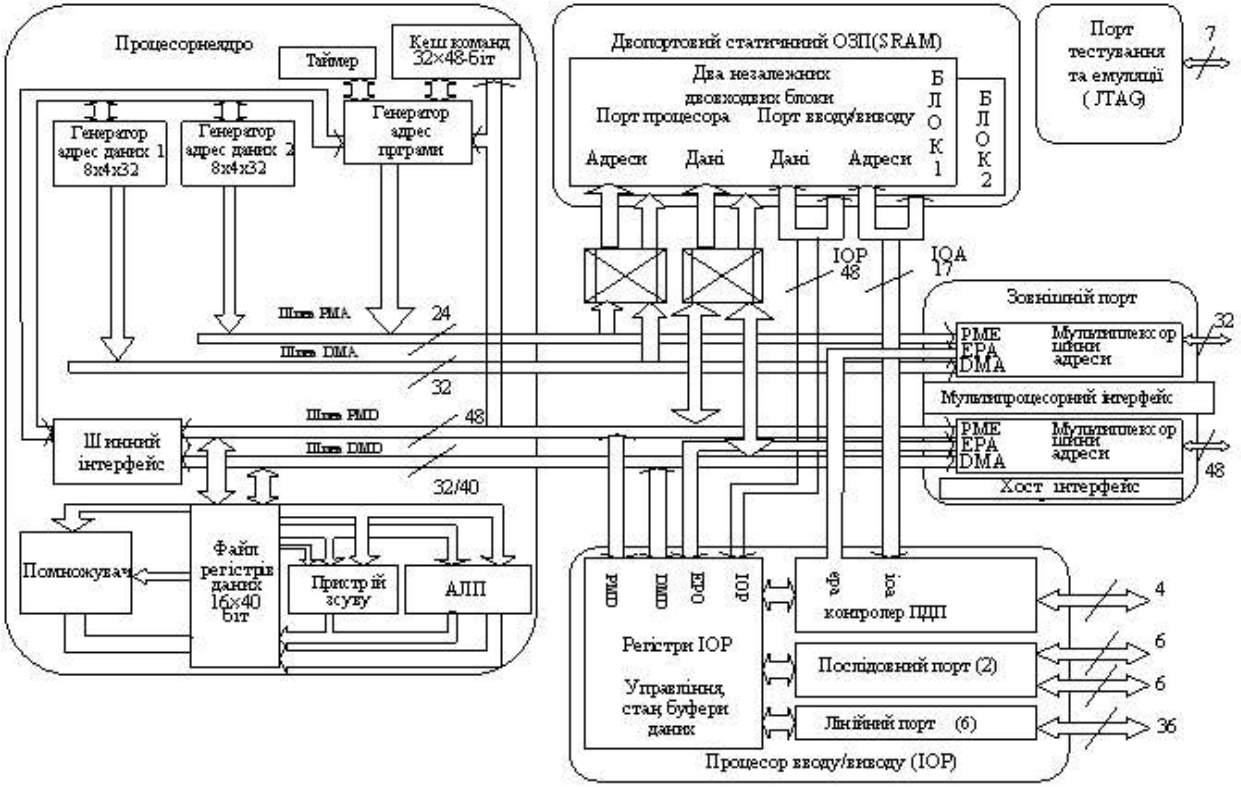


Рис. 3.18. Розширена структурна схема ADSP-21060 SHARC

3.3.3. Організація управління в ADSP-2106x. Призначення й архітектура контролера мікропрограм, генератора адреси й кеша команд

Два генератора адреси (ГА-1, ГА-2) (рис. 3.18) та контролер мікропрограм (КМП) генерують адресу для доступу до пам'яті. Разом КМП та генератори адреси даних дозволяють обчислювальним операціям виконуватися з найбільшою ефективністю, тому що пристрої обчислення використовуються виключно для обробки даних. Маючи у своєму складі кеш команд, ADSP-2106x може одночасно вибирати інструкцію з кеш-пам'яті та 2 операнди з пам'яті. Генератори адреси даних апаратно реалізують циклічні буфери даних. КМП передає адресу команди до програмної пам'яті. Він управляє ітераціями та оцінює результати виконання умовних команд. Завдяки внутрішньому лічильнику циклів та стеку циклів, ADSP-2106x виконує циклічні операції з нульовими непродуктивними затратами. При цьому не потребується явних інструкцій переходу для зациклювання або декрементації та перевірки лічильника.

Висока швидкодія ADSP-2106x досягається за рахунок конвеєрного режиму роботи, який передбачає вибірку, декодування та виконання команди. Генератори адреси забезпечують формування адреси пам'яті при передачі даних між пам'яттю та регістрами. Пара генераторів адрес даних дозволяє процесору отримувати адреси для одночасного виконання двох операцій: читання або запису операндів. ГА-1 посилає 32-розрядну адресу до пам'яті даних, а ГА-2 – 24-розрядну адресу до пам'яті програм для доступу до даних програмної пам'яті. Кожен з генераторів адрес забезпечує управління 8-ма покажчиками адрес, 8-ма модифікаторами адрес та 8-ма регістрами значень довжини. Покажчик, що використовується для непрямої адресації, при необхідності модифікується значенням, розташованим у спеціальному регістрі. Модифікація може виконуватися до або після здійснення доступу до пам'яті. З кожним покажчиком може бути пов'язаний визначений обсяг даних, що адресуються. Наприклад, щоб виконати автоматичну адресацію кільцевих буферів, кожен з регістрів обох генераторів адрес має дублюючий регістр, який активується при здійсненні швидкого контекстного переключення. Кільцеві буфери звичайно використовуються в цифрових фільтрах та перетвореннях Фур'є.

Контролер мікропрограм містить кеш команд на 32 слова. До кеш-пам'яті надсилаються тільки ті інструкції, які конфліктують при доступі до даних програмної пам'яті. Це дозволяє ядру функціонувати з повною швидкістю при циклічному виконанні багатьох операцій.

В ADSP-2106x використовуються чотири зовнішніх переривання. Три з них – загального призначення (IRQ2-0) та спеціальне переривання для скидання. Процесор також має внутрішні переривання таймера,

арифметичних виключень, мультипроцесорні векторні переривання, та переривання, що визначаються програмним забезпеченням користувача. При зовнішніх перериваннях в стеку автоматично зберігаються арифметичні стани та вміст регістру режиму (MODE1). Допускаються переривання з чотирма рівнями вкладеності.

Таймер, що програмується, забезпечує періодичну генерацію переривань. Якщо переривання дозволені, таймер декрементує 32-розрядний регістр підрахунку на кожному тактовому циклі. При переході цього регістру в нульовий стан ADSP-2106x генерує переривання, активізуючи свій вихід TIMEXP. Регістр підрахунку автоматично перезавантажується з 32-розрядного регістру періоду, і підрахування поновлюється.

3.3.4. Організація ОЗП в ADSP-2106x

На кристалі ADSP-21060 розташований ОЗП ємністю 4 Мбіт, що організований у вигляді двох незалежних блоків по 2 Мбіт кожний, які можуть бути конфігуровані як для збереження даних, так і кодів. ADSP-21062 містить статичний ОЗП ємністю 2 Мбіт, розподілений на блоки по 1 Мбіт.

Кожен блок пам'яті є двопортовим (рис.3.18). Доступ до ОЗП здійснюється за один машинний цикл як для ядра процесора та процесора вводу/виводу, так і для контролера ПДП. Двопортова пам'ять та розподілені внутрішньокристалічні шини дозволяють на протязі одного машинного циклу одночасно пересилати дані з ядра та одного з портів вводу/виводу. В ADSP-21060 пам'ять може мати дві конфігурації: максимум 128К 32-розрядних слів даних або 80К 48-розрядних слів інструкцій (або 40-розрядних даних) загальним обсягом до 4 Мбіт. Вміст пам'яті може бути обрано словами по 16, 32 або 48 розрядів. Крім цього може бути забезпечений 16-розрядний формат збереження з плаваючою точкою, при котрому подвоюється кількість даних, які можуть зберігатися на кристалі. Перетворення між 32-розрядним форматом з плаваючою точкою та 16-розрядним з плаваючою точкою здійснюється за допомогою однієї команди. Незважаючи на те, що кожен блок може зберігати комбінації кодів та даних, доступ до пам'яті найбільш ефективний тільки у випадку, коли один блок зберігає дані, використовуючи для пересилання шину даних (ШД) DMD, а інший – інструкції та дані, передаючи їх по шині даних PMD. Тільки при такому використанні шин, коли одна шина виділена одному, а інша – другому блоку, гарантується виконання двох пересилань впродовж одного машинного циклу. В цьому випадку інструкція, що виконується, повинна знаходитися у кеш-пам'яті. Виконання передачі даних протягом одного машинного циклу зберігається також в тому випадку, коли один з операндів

передається у внутрішню пам’яті із зовнішнього пристрою через зовнішній порт ADSP-2106x або у зворотному напрямку.

3.3.5. Організація зовнішніх зв'язків в ADSP-2106x. Архітектура процесора вводу-виводу й портів зв'язку

Зовнішні порти забезпечують зв'язок процесора ADSP-2106x з пам’яттю, що розташована поза кристалом, та зовнішніми пристроями. При цьому в об’єднаний адресний простір може бути включено до 4Г слів адресного простору позакристалічної пам’яті. Роздільні шини всередині мікросхеми для адрес пам’яті програм і пам’яті даних (PMA та DMA) та даних пам’яті програм і пам’яті даних (PMD та DMD), адреса вводу/виводу та даних вводу/виводу мультиплексуються в інтерфейсному процесорі для створення зовнішньої системної шини з однією 32-розрядною шиною адреси (ША) та однієї 48-розрядної шини даних (ШД). Зовнішній статичний ОЗП може бути 16, 32, або 48-розрядним. При цьому інтегрований в ЦПОС контролер ПДП автоматично упаковує зовнішні дані в слово прийнятної довжини (48 розрядів для інструкцій або 32 розряди для даних).

Адресація зовнішніх пристроїв пам’яті спрощується за рахунок декодування всередині кристала старших розрядів ліній адреси та генерації сигналу вибору банка пам’яті. Для спрощення організації сторінкової адресації є можливість активізації окремих управляючих ліній процесора. ADSP-2106x дозволяє програмувати стан очікування пам’яті та контроль підтвердження зовнішньої пам’яті та зовнішніх пристроїв.

Інтерфейс головного (host) процесора дозволяє здійснювати простий зв'язок із стандартними 16- або 32-розрядними процесорами. Для цього потребується невелике додаткове програмне забезпечення. При цьому підтримується асинхронна передача із швидкістю, що відповідає максимальній частоті тактових імпульсів ЦПОС. Інтерфейс головного процесора доступний через зовнішній порт ADSP-2106x, що відображається на пам’ять в об’єднаному адресному просторі. Чотири канали ПДП підтримують пересилання кодів та даних з мінімальними програмними затратами.

Мікросхема містить засоби для роботи в багатопроцесорних системах ЦОС. Об’єднаний адресний простір дозволяє здійснювати прямий доступ до пам’яті кожного з ADSP-2106x, які включені до багатопроцесорної системи. В мікросхему інтегрована логіка шинного арбітражу, яка забезпечує управління системою, що складається з одного провідного та до шести підпорядкованих ADSP-2106x. Зміна провідного процесора потребує тільки одного циклу. Шинний арбітраж може функціонувати на основі фіксованого або циклічного пріоритету. Дозволяється захоплення шин процесора для

здійснення послідовностей читання-зміна-запис, а також забезпечується виконання векторних переривань для міжпроцесорних команд. Максимальна швидкість пересилань даних через порти зв'язку або зовнішній порт складає 240 Мбіт/с.

Процесор вводу/виводу містить два послідовних порти, шість 4-бітових лінійних портів зв'язку та контролер ПДП. Послідовні порти ADSP-2106x є синхронними та забезпечують інтерфейс з різними типами зовнішніх пристроїв. Послідовні порти функціонують з тактовою частотою процесора, що забезпечує максимальну швидкість передачі даних 40 Мбіт/с. Тактування портів може бути внутрішнім або зовнішнім. Незалежність процесів прийому та передачі забезпечує велику гнучкість послідовного зв'язку. Обмін через послідовні порти також може здійснюватися в режимі ПДП.

ADSP-2106x має шість 4-розрядних портів зв'язку, які забезпечують додаткові можливості вводу/виводу. Їх можна використовувати в мультипроцесорних системах для міжпроцесорної взаємодії за схемою двох точок (point-to-point).

Порти зв'язку можуть функціонувати паралельно з максимальною швидкістю 240 Мбіт/с. Дані портів зв'язку можуть бути упаковані у 32- або 48-розрядні слова та безпосередньо зчитані ядром процесора або передані до внутрішньокристалічної пам'яті через ПДП. Кожен порт зв'язку має власні вхідні та вихідні регістри синхронізації.

3.3.6. Система команд і програмно доступні регістри в ADSP-2106x

Набір команд ЦПОС сімейства ADSP-2106x забезпечує широкий вибір можливостей програмування. Багатофункціональні команди дозволяють здійснювати обчислення одночасно з передачею або модифікацією даних, а також одночасні операції у помножувачі та АЛП. Кожна команда виконується впродовж одного машинного циклу. Для полегшення кодування та розуміння програм асемблер процесора використовує алгебраїчний синтаксис.

Команди ЦПОС SHARC можна розподілити на чотири групи:

• команди обчислення та передачі модифікованих даних, які специфікують обчислювальну операцію з паралельною передачею одного чи двох полів даних або модифікацією індексного регістру;

• команди, що управляють виконанням програми, які специфікують різні типи розгалужень, циклів, викликів підпрограм та повернень;

• команди безпосереднього пересилання даних, в яких поля інструкцій використовуються як операнди, або безпосередньо для адресації;

• інші команди, зокрема, перевірка та модифікація окремих бітових полів, відсутність операції та ін.

В багатьох командах містяться поля специфічних операцій обчислення, які використовуються в АЛП, помножувачі або в пристрої зсуву.

Процесор містить велику групу програмно доступних регістрів. Вони є універсальними, тобто дозволяють як читати, так і модифікувати їх вміст.

Усі регістри ЦПОС ADSP-2106x можна розподілити на 6 груп [23]:

• регістри загального призначення (табл. 3. 6);
• регістри управління послідовністю команд (табл. 3. 7);
• адресні регістри (табл. 3. 8);
• регістри управління шинами (табл. 3. 9);
• регістри таймера (табл. 3. 10);
• системні регістри (табл. 3. 11).

Таблиця 3.6 – Регістри загального призначення

Позначення	Назва та призначення
R15 – R0	Регістри для операцій з фіксованою точкою
F15 – F0	Регістри для операцій з плаваючою точкою

Таблиця 3.7 – Регістри управління послідовністю команд

Позначення	Назва та призначення
PC	Програмний лічильник
PCSTK	Верхівка стека
PCSTKP	Покажчик стека
FADDR	Регістр адреси вибірки
DADDR	Адреса, що декодується
LADDR	Адреса закінчення циклу
CURLCNTR	Лічильник поточного циклу
LCNTR	Лічильник циклів для вкладених циклів

Таблиця 3.8 – Адресні регістри

Позначення	Назва та призначення
I7 – I0	Індексні регістри ГА-1
M7 – M0	Регістри модифікації ГА-1
L7 – L0	Регістри довжини ГА-1
B7 – B0	Базові регістри ГА-1
I15– I8	Індексні регістри ГА-2
M15– M8	Регістри модифікації ГА-2
L15– L8	Регістри довжини ГА-2
B15– B8	Базові регістри ГА-2

Таблиця 3.9 – Регістри управління шинами

Позначення	Назва та призначення
PX1	Обмін шин 1. Шина даних пам'яті програм - Шина даних пам'яті даних (16 біт)
PX2	Обмін шин 2. Шина даних пам'яті програм - Шина даних пам'яті даних (32 біта)
PX	48-бітова комбінація PX1 та PX2

Таблиця 3.10 – Регістри таймера

Позначення	Назва та призначення
TPERIOD	Період таймера
TCOUNT	Лічильник таймера

Таблиця 3.11 – Системні регістри

Позначення	Назва та призначення
MODE1	Управління режимом у стані 1
MODE2	Управління режимом у стані 2
IRPTL	Фіксатор переривань
IMASK	Маскування переривань
IMASP	Покажчик маски переривань (для вкладеності)
ASTAT	Прапори арифметичного стану
STKY	Прапори стану помилок
USTAT1	Регістр 1 стану користувача
USTAT2	Регістр 1 стану користувача

Для локального збереження даних використовуються регістри загального призначення (РЗП), які, за термінологією розробника, мають назву «файл регістрів». Файл регістрів (ФР) складається з 16 40-бітних основних та 16 альтернативних регістрів. Номери регістрів позначаються символом R при обчисленні з фіксованою точкою, та символом F в операціях з плаваючою точкою. АЛП може обирати два операнди (x та y) з будь-якого РЗП. Результат операції вміщується в будь-якому з регістрів файла регістрів. Залежно від результату операції встановлюються відповідні прапори регістра стану ОЗП – ASTAT.

Команди АЛП на мові асемблеру ADSP-2106x записуються таким чином:

```
Rn=Rx+Ry;                /* Сума вмісту регістрів у форматі з фіксованою
                            точкою*/
Fn=Fx+Fy;                /* Сума вмісту регістрів у форматі з плаваючою
                            точкою*/
Rn=(Rx+Ry)/2;            /*Визначення середнього значення*/
Rn=MIN(Rx,Ry);           /*Визначення мінімального операнда*/
Fn=RSQRTS Fx;            /*Обчислення зворотного значення квадратного
                            кореня із вмісту Fx  у форматі з плаваючою точкою
                            */
Ra=Rx+Ry, Rs=Rx-Ry;      /*Одночасне складання та віднімання */
                         /* Rx, Fx, Ry, Fy – будь-які РЗП для операцій з */
Fa=Fx+Fy, Fs=Fx-Fy;      /*фіксованою або плаваючою точами */
```

Для виконання операцій множення з фіксованою та плаваючою точками, а також операцій множення з накопиченням застосовується апаратний помножувач, який помножує два вхідних операнди, позначених X

та Y, з файла регістрів. Акумулювання результату відбувається в двох 80-розрядних регістрах (MRB та MRF), які допускають контекстне перемикання. Обидва регістри мають однакову структуру, що складається з трьох частин: MR0, MR1, MR2. Кожен з регістрів може бути незалежно зчитаний чи записаний у файл регістрів. Регістр MR2 має тільки 16 розрядів, але при читанні його вміст розширюється до 32 розрядів. Залежно від результату множення встановлюються чотири прапори регістра ASTAT та чотири прапори регістра стану помилки STKY.

На додаток до обчислень, що виконуються кожним обчислювальним пристроєм окремо, ADSP-2106x дозволяє здійснювати багатофункціональні обчислення, при яких має місце паралельне функціонування помножувача та АЛП. Дві операції виконуються таким чином, як вони виконувалися б у відповідних однофункціональних обчисленнях. Кожен з чотирьох вхідних операндів багатофункціональних команд обмежується визначеним місцем розташування у файлі регістрів. У всіх інших операціях вхідними операндами можуть бути будь-які регістри файла регістрів.

На мові асемблеру ADSP-2106x команди множення та багатофункціональні команди записуються таким чином:

Rn=Rx*Ry;
MRF= Rx*Ry;
MRB= Rx*Ry;
Fn=Fx*Fy;
Rn=MRF+ Rx*Ry;
Rn=MRB-Rx*Ry;
MRF=MRF-Rx*Ry;

Синтаксис команд є простим, тому не потребує додаткових пояснень. При виконанні однофункціональних команд в якості регістрів Rn, Rx та Ry може бути використаний будь-який регістр файла регістрів, а при багатофункціональних – тільки регістри відповідно до схеми, що зображена на рис. 3.19 [23].

Регістри управління послідовністю команд розташовані в контролері мікропрограм та є універсальними, тобто доступ до них здійснюється так як і для інших універсальних регістрів. ADSP-2106x виконує команди на протязі трьох тактів: вибірки, декодування, виконання. За рахунок конвеєризації, поки здійснюється вибірка поточної команди, виконується декодування інструкції, обраної у попередньому циклі, та виконання команди, обраної два такти тому. Тобто, продуктивність процесора складає одну команду за один цикл. Будь-яке розгалуження програми знижує продуктивність.

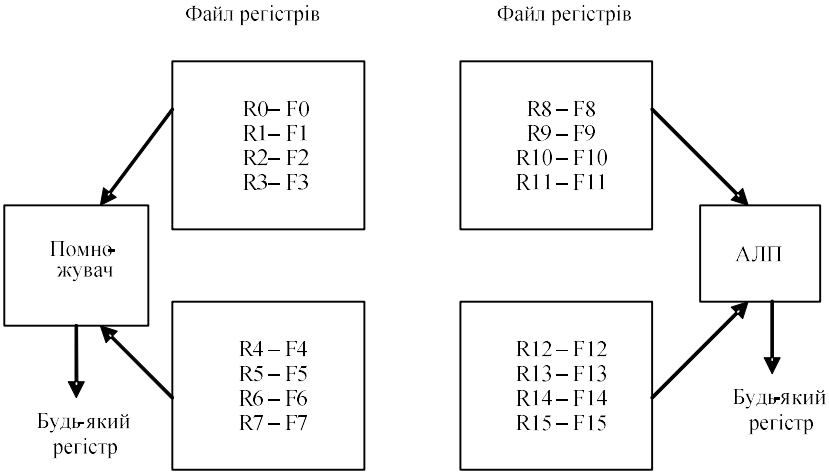


Рис. 3.19. Схема закріплення регістрів при багатофункціональних операціях

Регістр адреси вибірки FADDR, регістр адреси декодування DADDR та програмний лічильник PC містять відповідно адреси команди, що обирається, та команди, що декодується та виконується в даний момент часу. Покажчик стека програмного лічильника PCSTKP являє собою регістр, доступний для читання та запису, і містить адресу верхівки стека. При пустому стеку маємо PCSTKP=0, а PCSTKP=1, 2, …,30 за наявності в стеку даних. У випадку PCSTKP=31 виникає переповнення стеку.

Контролер мікропрограм оперує з двома окремими лічильниками циклів: лічильником поточного циклу (CURLCNTR), який відстежує ітерації поточного циклу, та зі звичайним лічильником циклів (LCNTR), який задає кількість повторів. Наявність двох лічильників необхідна для підтримки вкладених циклів. Регістр LADDR зберігає адресу верхівки стека адрес циклів. Він доступний для читання/запису через шину даних пам’яті даних. У випадку, коли стек адрес циклів пустий, він набуває значення 0xFFFFFFFF.

ADSP-2106x підтримує програмні цикли з інструкцією DO UNTIL. Ця команда здійснює повтор послідовностей інструкцій доти, поки умови, що перевіряються, не стануть істинними (true). Контролер мікропрограм аналізує для визначення моменту припинення виконання циклу шляхом перевірки бітів регістрів стану ASTAT, управління режимом MODE1, лічильника циклів CURLCNTR та вхідних прапорів. Кожна умова, яку оцінює ADSP-2106x, має мнемоніку мови асемблеру та унікальний код, який використовується в коді операції. Для більшості умов контролер може

перевіряти обидва стани: true та false. Умова LCE (закінчення підрахунку лічильника циклів) найчастіше використовується в інструкціях DO UNTIL.

Інструкція DO UNTIL забезпечує ефективний цикл без непродуктивних втрат (додаткових команд переходу, перевірки стану або зменшення лічильника). Наприклад:

```
        LCNTR=30,       DO label UNTIL LCE;
        R0=DM(10,M0), F2=PM(18,M8);
        R1=R0-R15;
label:  F4=F2+F3;
```

У процесі виконання команди DO UNTIL контролер мікропрограм заносить до стека адреси циклу адресу останньої команди та умову завершення для виходу з циклу. Він також заносить до стека програмного лічильника адресу верхівки циклу, яка являє собою адресу інструкції, що безпосередньо йде за рядком DO UNTIL.

Адресні регістри розташовані в ГА-1 та ГА-2. Обидва генератори адреси мають чотири групи регістрів по вісім однотипних регістрів кожної з груп: індексні (I), модифікації (M), бази (B) та довжини (L). У 32-розрядному ГА-1 регістри нумеруються 0 – 7, а 24-розрядному ГА-2 мають номери 8 – 15. I – регістр виконує роль покажчика на пам'ять, M – регістр містить значення приросту покажчика. Регістри B та L використовуються для адресації кільцевих буферів даних. Зокрема, в регістрі B розташована начальна (базова) адреса кільцевого буфера, а в регістрі L – довжина буфера. Кожен з регістрів ГА має альтернативний набір регістрів, який використовується при контекстному переключенні.

Пристрій зсуву працює з 32-розрядними операндами з фіксованою точкою та виконує такі функції:

- зсуви та циклічні зсуви;
- операції маніпулювання бітами;
- маніпулювання полями бітів, включаючи виділення та копіювання до зовнішньої пам'яті;
- підтримка операцій сумісності процесорів сімейства ADSP-2100 (фіксована/плаваюча точка).

Пристрій зсуву приймає від одного до трьох операндів: X – операнд, над яким проводиться дія; Y – операнд, що визначає величину зсуву s, довжини полів бітів або положення бітів; Z – операнд, над яким проводиться дія та корекція. Пристрій зсуву повертає значення результату в один з регістрів ФР.

Вхідні операнди обираються із старших 32 розрядів регістра ФР (біти 39 – 8), або як безпосереднє значення з поля інструкції. Операнди пересилаються протягом першої половини циклу. Результат передається у старші 32 розряди регістра протягм другої половини циклу, при цьому

молодші вісім бітів обнуляються. Таким чином, пристій зсуву може здійснювати читання та запис того самого місця пам'яті за один машинний цикл. X та Z – це завжди 32-розрядні числа з фіксованою точкою. Вхід Y – 32- розрядне значення з фіксованою точкою або 8-бітове поле.

Наприклад, при виконанні команди FDEP, яка дозволяє маніпулювати групами бітів в межах 32-розрядного слова фіксованої довжини, вхід Y визначає два 6-розрядних значення (bit6 та len6), розташованих в Ry. При цьому bit6 – початкова позиція для виділення або депонування, а len6 – довжина поля бітів, яка визначає кількість бітів, які виділяються або депонуються. Так, при виконанні команди R0=FDEP R1 BY R2 (R1=0x000000FF00 R2=0x000002100) в регістр R0 буде завантажено фрагмент вмісту регістру R1, що складається з 8 бітів, починаючи від початку цього регістру (len6=8), та що відстоїть від початку регістра результату R0 на 16 розрядів (bit6=16). Оскільки в даному випадку обробка 32-розрядних слів здійснюється у 40-розрядних регістрах, то відлік бітів в регістрах починається з 8-го розряду (біти з 0 по 7 ігноруються та обнуляються). У результаті виконання такої операції вміст регістру R0 буде 0x00FF000000.

Для забезпечення можливостей передачі інформації між 48-розрядною шиною даних пам'яті програм та 32-розрядною шиною пам'яті даних використовується PX регістр, який складається з двох частин: 16-розрядного PX1 та 32-розрядного PX2. Вони можуть використовуватися незалежно або розглядатися як складові частини регістру PX. Регістр PX дозволяє здійснити обмін даними по шині даних пам'яті програм або по шині даних пам'яті даних з пам'яттю або регістрами ФР. Наприклад, щоб записати 48-розрядне слово до комірки пам'яті з ім'ям Port1 по шині даних пам'яті програм можна використати такі команди:

```
R0=0x9A00;/*Завантаження до R0 16-ти молодших бітів*/
R1-0x12345678;/*Завантаження до R1 32-х старших бітів*/
PX1=R0;
PX2=R1;
PM(Port1)=PX;
```

Для управління таймером застосовуються два регістри: TPERIOD та TCOUNT. Обидва регістри доступні як для читання, так і для запису. До першого з них заноситься кількість періодів підрахунку, яка після запуску таймера переноситься до регістру підрахунку TCOUNT. На кожному періоді підрахунку таймер зменшує вміст TCOUNT на одиницю. Коли вміст регістру стає таким, що дорівнює нулю, таймер генерує переривання та заносить на вихід TIMEXP логічну одиницю тривалістю 4 цикли тактових імпульсів (якщо робота таймера не заборонена встановленням відповідного прапора регістру режимів MODE2). Далі відбувається автозавантаження числа періодів підрахунку з TPERIOD в TCOUNT.

Системні регістри призначені для задавання різноманітних режимів роботи процесора, управління операціями переривання, індикації результатів логічних та арифметичних операцій. Для управління режимом та станом процесора використовуються два регістри режиму MODE1 та MODE2. Шляхом встановлення та скидання окремих бітів цих регістрів можна змінювати режими адресації, перемикати ФР на альтернативні регістри, управляти перериваннями, дозволяти роботу таймера, визначати тип сигнального процесора та ін. [23].

Назва та призначення бітів MODE1 наведено в табл.. 3.12. Регістр MODE2 використовується для управління таймером, для заборони виходу на зовнішню шину, він блокує кеш, містить інформацію про тип процесора, що використовується, та ін.

Таблиця 3.12 – Призначення бітів регістру MODE1

Номер біта	Назва	Призначення
0	BR8	Інвертування розрядів регістру I8 (ГА-2)
1	BR0	Інвертування розрядів регістру I0 (ГА-1)
2	SRCU	Вибір альтернативного регістру обчислювача
3	SRD1H	Вибір альтернативних регістрів ГА-1 (7 - 4)
4	SRD1L	Вибір альтернативних регістрів ГА-1 (3 - 0)
5	SRD2H	Вибір альтернативних регістрів ГА-2 (15 - 12)
6	SRD2H	Вибір альтернативних регістрів ГА-2 (11 - 8)
7	SRRFH	Вибір альтернативних файлових регістрів (R15 – R8)
8-9	-	Зарезервовані
10	SRREL	Вибір альтернативних файлових регістрів (R7 – R0)
11	NESTM	Дозвіл вкладеності переривань
12	IRPTEN	Дозвіл глобальних переривань
13	ALUSAT	Дозвіл насиченості АЛП
14	SSE	Дозвіл знакового розширення короткого слова
15	TRUNC	1 – усікання з плаваючою точкою, 0 – округлення
16	RND32	1 – округлення з плаваючою точкою до 32 біт, 0 – до 40 біт
17-18	CSEL	00 – вибір основної шини
19-31	-	Зарезервовані

Регістр арифметичного стану ASTAT відображає результат виконання операції не тільки в АЛП, але й у помножувачі та пристрої зсуву.

Регістр стану помилки STKY показує додаткові ознаки стану виконання операцій або уточнює причину встановлення одного чи декількох прапорів регістру ASTAT. Прапори в регістрі STKY перевіряються після закінчення серії операцій. Якщо будь-який прапор встановлено, то деякі з результатів є помилковими. Призначення бітів регістрів ASTAT та STKY наведено в табл.. 3.13 та 3.14 відповідно.

Таблиця 3.13 – Призначення бітів регістру ASTAT

Номер біта	Назва	Індикація стану (регістр ASTAT)
0	AZ	Вміст АЛП = 0, або втрата значності при операціях з плав. точкою
1	AV	Переповнення АЛП
2	AN	Вміст АЛП менше 0
3	AC	Без переносу АЛП з фіксованою точкою
4	AS	Знак входу X АЛП
5	AI	Неможлива операція в режимі з плаваючою точкою
6	MN	Знак множення негативний (від'ємний)
7	MV	Переповнення помножувача
8	MU	Втрата значності при операціях с плаваючою точкою
9	MI	Неможлива операція помножувача при операціях з плав. точкою
10	AF	Індикація операції АЛП з плаваючою точкою
11	SV	Переповнення пристрою зсуву
12	SZ	Вміст пристрою зсуву дорівнює 0
13	SS	Знак операнда на вході пристрою зсуву
14-17	-	Зарезервовані
18	BTF	Прапор тестування біта
19	FLG0	Значення прапора 0
20	FLG1	Значення прапора 1
21	FLG2	Значення прапора 2
22	FLG3	Значення прапора 3
23	-	Зарезервовані
24-31	CACC	Порівняння бітів акумулятора

Таблиця 3.14 – Призначення бітів регістру STKY

Номер біту	Назва	Індикація стану (регістр STKY)
0	AUS	Втрата значності АЛП з плаваючою точкою
1	AVS	Переповнення АЛП з плаваючою точкою
2	AOS	Переповнення АЛП з фіксованою точкою
3-4	-	Зарезервовані
5	AIS	Невірна операція в АЛП з плаваючою точкою
6	MOS	Переповнення помножувача з фіксованою точкою
7	MVS	Переповнення помножувача з плаваючою точкою
8	MUS	Втрата значності помножувача з плаваючою точкою
9	MIS	Невірна операція в помножувачі з плаваючою точкою
10-16	-	Зарезервовані
17	CB7S	Переповнення циркулярного буфера 7 ГА-1
18	CB15S	Переповнення циркулярного буфера 15 ГА-1
19-20	-	Зарезервовані
21	PCFL	Переповнення покажчика стека
22	PCEM	Покажчик стека пустий
23	SSOV	Переповнення стека стану (MODE1 та ASTAT)
24	SSEM	Регістр стека пустий
25	LSOV	Переповнення стека циклів
26	LSEM	Стек циклу пустий
27-31	-	Зарезервовані

3.3.7. Розподіл адресного простору пам'яті в ADSP-2106x

Весь простір пам'яті розподіляється на три секції: простір внутрішньої пам'яті, простір мультипроцесорної пам'яті та простір зовнішньої пам'яті. Простір внутрішньої пам'яті ADSP-2106x складається з внутрішньокристалічної пам'яті та власних ресурсів. Мультипроцесорний простір пам'яті відповідає внутрішньокристалічній пам'яті та ресурсам іншого ADSP-2106x у мультипроцесорній системі. Простір зовнішньої пам'яті відповідає зовнішній пам'яті та пристроям вводу/виводу, що відображаються на пам'ять. Для звертання до пам'яті в процесорі генеруються адреси, формат яких наведено на рис. 3. 20 [23].

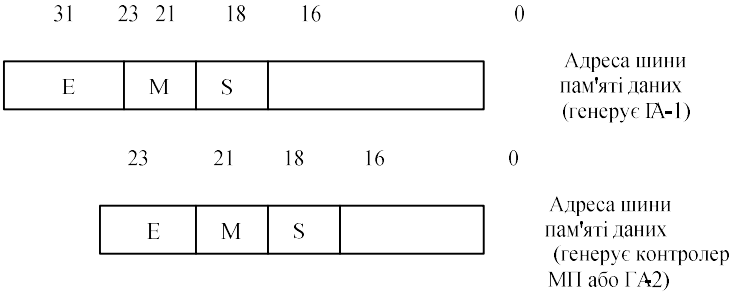


Рис. 3.20. Формат адрес процесора ADSP-2106x

Частина розрядів адресного слова виділяється для ідентифікації секції пам'яті, що використовується: E – зовнішня, M – мультипроцесорна та S – внутрішня пам'ять. Якщо поле E містить всі нулі, то поля M та S стають активними та використовуються для адресації власної внутрішньої пам'яті (M=000), або для пересилання у внутрішню пам'ять іншого процесора, що входить до складу мультипроцесорної системи. В останньому випадку поле M ідентифікує номер іншого процесора. Секції пам'яті обмежені такими адресами:

- внутрішня пам'ять від 0x00000000 до 0x0007FFF;
- мультипроцесорна пам'ять від 0x00080000 до 0x003FFFF;
- зовнішня пам'ять від 0x00400000 до 0xFFFFFFF.

Внутрішня пам'ять ADSP-2106x має три області адрес:

- регістри процесора вводу/виводу від 0x00000000 до 0x00000FF;
- адреси нормальних слів від 0x00020000 до 0x000FFFF;
  (таблиця векторів переривань від 0x00020000 до 0x0002007F);
- адреси коротких слів від 0x00040000 до 0x0007FFFF.

Регістри процесора вводу/виводу являють собою 256 регістрів, що відображаються на пам'ять і контролюють конфігурації системи ADSP-2106x, а також різноманітні операції вводу/виводу. Адресний простір між регістрами вводу/виводу та адресами нормальних (за довжиною) слів (комірки від 0x00000100 до 0x0001FFF) зарезервовано, та інформація до них записуватися не повинна.

Блок пам'яті 0 починається з початку простору нормальних слів (з адреси 0x002000), а блок 1 – з середини простору нормальних слів (з адреси 0x00030000). Таким чином, для різних областей внутрішньої пам'яті використовуються такі діапазони адрес:

• 0x00000000 – 0x000000FF – регістри вводу/виводу (IOP);

• 0x00010000 – 0x0001FFFF – резервні адреси;

• 0x00020000 – 0x0002FFFF – блок 0 (адресація 32- та 48-розряднх слів);

• 0x00030000 – 0x0003FFFF – блок 1 (адресація 32- та 48-розряднх слів);

• 0x00040000 – 0x0005FFFF – блок 0 (адресація 16-розрядних слів);

• 0x00060000 – 0x0007FFFF – блок 1 (адресація 16-розрядних слів).

Простір адрес нормальних (32 та 48 розрядів) та коротких (16 розрядів) слів являє собою ту саму фізичну пам'ять. Наприклад, вибірка слова з адресою 0x00020000 – це ті ж самі комірки, що і доступ до коротких слів з адресами 0x00040000 та 0x00040001 (для 32-розрядних даних у просторі нормальних слів). Адресація коротких слів збільшує кількість 16-розрядних даних, які можуть зберігатися у внутрішній пам'яті. Вона широко застосовується, зокрема, у системах, що здійснюють операції з матрицями.

Простір мультипроцесорної пам'яті у мультипроцесорній системі відображається на внутрішню пам'ять іншого ADSP-2106x. Це дозволяє кожному з ADSP-2106x звертатися до внутрішньої пам'яті та до регістрів вводу/виводу, що відображаються на пам'ять іншого процесора. Значення поля M адресної частини визначає ідентифікаційний номер (ID2-0) зовнішнього ADSP-2106x процесора, до якого здійснюється доступ, і тому тільки цей процесор буде реагувати на цикл читання/запис. Якщо поле M=111, то відбувається циркулярний запис у всі процесори мультипроцесорної системи.

Звертання до зовнішньої пам'яті може відбуватися через зовнішній порт по шинах пам'яті програм, пам'яті даних та вводу/виводу. Цими шинами управляє ГА-1, контролер мікропрограм (ГА-2) або процесор вводу/виводу. ГА-1 та процесор вводу/виводу видають 32-розрядні адреси на шину адреси пам'яті програм та на шину адреси вводу/виводу. При цьому можливо адресувати 4Г слів пам'яті. Контролер мікропрограм та ГА-2 генерують

24-розрядні адреси по шині адреси пам'яті програм, обмежуючи адресацію молодшими 12М словами пам'яті (0x00400000 – 0x00FFFFFF).

Кожен блок внутрішньої пам'яті може бути пристосований для збереження 32-розрядних даних з одинарною точністю або для збереження 40-розрядних даних з підвищеною точністю. Ці дії ініціюються встановленням чи скиданням бітів IMDW0 та IMDW1 в регістрі системної конфігурації SYSCON.

На додаток до розгляду внутрішньокристалічної пам'яті слід зазначити, що ADSP-2106x забезпечує адресацію 4Г слів зовнішньої пам'яті через зовнішній порт. Ця пам'ять може зберігати як інструкції, так і дані. Зовнішній адресний простір включає до себе простір мультипроцесорної пам'яті, внутрішню пам'ять всіх інших ADSP-2106x, що об'єднані у мультипроцесорну систему, а також простір зовнішньої пам'яті та область для стандартної позакристалічної пам'яті.

У сигнальному процесорі використовуються безпосередня, відносна та непряма регістрова адресації. У першому випадку адреса комірки пам'яті вказується безпосередньо в полі операнда команди, наприклад

dm(0x000047E0)=astat

при виконанні цієї команди значення astat заноситься до комірки пам'яті даних з адресою 000047E0h.

При відносній адресації адреса обчислюється як сума програмного лічильника PC та зміщення, що задається в полі команди. Наприклад,

call(pc,20)

При непрямій регістровій адресації використовуються регістри ГА-1 та ГА-2. У команді вказується регістр, в якому міститься адреса необхідної комірки пам'яті. При цьому можливі два режими роботи:

• попередня модифікація (pre-modify). Ефективна адреса визначається як сума вмісту індексного регістру I та регістру M або безпосереднього операнда. При цьому вміст регістру I не змінюється;

• формування адреси з наступною модифікацією індексного регістру (post-modify). ГА видає на шину адреси вміст індексного регістру I, а потім модифікує його шляхом додавання вмісту регістру M або безпосереднього значення модифікатора.

В асемблері ADSP-2106x ці операції різняться позиціями індексу та модифікатора (M-регістру або безпосереднього значення) в інструкції. Розташування першим в полі команди регістра I означає режим post-modify. Наприклад,

R6=PM(I15,M12)

Ця інструкція здійснює звертання до комірки програмної пам'яті з адресою, що знаходиться в I15, а потім I15+M12 записується до I15.

Якщо модифікатор записаний першим, то це свідчить про використання попередньої модифікації без операції оновлення вмісту регістру I. При виконанні команди

R6=PM(M12, I15)

обирається комірка програмної пам'яті з адресою I15+ M12, але при цьому вміст I15 не змінюється.

Зміна будь-якого регістру I здійснюється в межах одного ГА. Так, команда

DM=(M0,I2)=TPERIOD

записана правильно і дозволяє обрати комірку пам'яті даних < M0+I2>, але команда

DM=(M0,I4)=TPERIOD

помилкова, тому що M та I належать різним генераторам адреси.

Максимальне безпосереднє значення, що модифікує регістр I, залежить від типу інструкції та від того, де розташований регістр I: в ГА-1 чи ГА-2. При використанні ГА-1 максимальне безпосереднє значення може дорівнювати 2^{32}, а при використанні ГА-1 – 2^{24}.

3.3.8. Реалізація мультипроцесорних систем на базі ADSP-2106x. Архітектура й технічні засоби

Різноманітні операції ЦОС потребують дуже високої швидкості обчислень, які не можуть реалізуватися на одному процесорі навіть з максимально можливою швидкодією. У той же час багату кількість алгоритмів можливо розподілити на окремі задачі, які будуть виконуватися паралельно різними процесорами.

При розробці процесорів сімейства ADSP-2106x до його схеми були введені функціональні особливості, які дозволяють використовувати ЦПОС у багатопроцесорних системах. Ці особливості включають наявність вбудованого арбітражу доступу до провідної шини та можливість звертання до ОЗП та до регістрів процесора вводу/виводу інших ADSP-2106x.

У багатопроцесорних системах з декількома ADSP-2106x, що функціонують в режимі розподілу зовнішньої шини, будь-який процесор може стати провідним та управляти системною шиною яка, складається з 40-бітової ШД, 32-розрядної ША та групи управляючих ліній. При цьому ОЗП та регістри процесора вводу/виводу (ПВВ) всіх ADSP-2106x, що входять до системи, створюють простір пам'яті багатопроцесорної системи. Адреса внутрішнього ОЗП та регістри ПВВ кожного процесора відображаються на цей простір. Як тільки один з ЦПОС стає провідним, він може безпосередньо працювати з ОЗП та регістрами ПВВ будь-якого підлеглого ADSP-2106x, включаючи порт FIFO-буферів даних. Провідний процесор може записувати

дані до регістрів ПВВ для встановлення режиму ПДП або для генерації векторного переривання.

У багатопроцесорних системах зазвичай використовують дві схеми зв'язків між процесорами. У першій застосовують виділений канал для обміну даними між ЦПОС, у другій – єдину глобальну пам'ять, доступ до якої здійснюється через глобальну шину. У випадку виділення каналу обміну процесор ADSP-2106x може бути з'єднаний з іншими процесорами через власні лінійні порти, або він може підключатися до паралельної шини, що розподіляється (так зване кластерне з'єднання).

Ефективність багатопроцесорної системи суттєво залежить від схеми обміну інформацією між процесорами, яка визначає втрати на зв'язки між процесорами та швидкість передачі даних. Наявність у ADSP-2106x вбудованих апаратних засобів дозволяє будувати багатопроцесорні системи на основі однієї з трьох топологій: потокової, кластерної або матричної [23]. Структурна схема потокової багатопроцесорної системи, де процесори з'єднані через лінійні порти, наведена на рис. 3. 21.

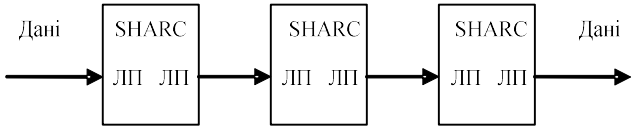


Рис. 3.21. Потокова мікропроцесорна система

Багатопроцесорна система, побудована за потоковою топологією (Data Flow Multiprocessing), складається з декількох процесорів SHARC, що об'єднані за допомогою лінійних портів. Ця топологія найбільш придатна для обробки великих міжпроцесорних потоків. У випадку використання такої схеми програміст розподіляє алгоритм обробки на декілька процесів та пропускає дані послідовно по конвеєру. ADSP-2106x SHARC ідеально підходить для систем такого типу, тому що не має необхідності у зовнішній пам'яті та буферах FIFO для зв'язків між процесорами. Ємність вбудованої пам'яті процесора достатня для розміщення кодів команд та даних більшості прикладних задач. Крім того, схема обробки проста у реалізації та має невелику вартість. Недоліком цієї топології є обмеження гнучкості системи.

Кластерна топологія (Cluster Multiprocessing) найбільш придатна для систем, які потребують максимальної гнучкості, зокрема, при підтримці декількох задач, частина з котрих може конкурувати. Кластерні багатопроцесорні системи складаються з декількох процесорів SHARC, що з'єднуються паралельною шиною, яка дозволяє здійснювати міжпроцесорний доступ як до вбудованого у кристал ОЗП, так і до глобальної пам'яті, що розподіляється. Структурна схема цієї топології наведена на рис. 3. 22.

У типовому кластері (блоці) на процесорах ADSP-2106x може вміщуватися до шести процесорів та одного головного (хост-процесора), що здійснює арбітраж шини. Вбудована у кристал логіка дозволяє розподіляти загальну шину без додаткової апаратної частини. Запит шини генерується автоматично кожного разу, коли процесор звертається до зовнішньої адреси. Як тільки ADSP-2106x стає провідним, він може отримати доступ не тільки до зовнішньої, але і до внутрішньої пам'яті та до регістрів інших процесорів. Провідний процесор напряму переміщує дані в інший процесор або використовує для цього канал ПДП. Кластерна конфігурація дозволяє SHARC-процесорам працювати на максимальній швидкості міжпроцесорного обміну.

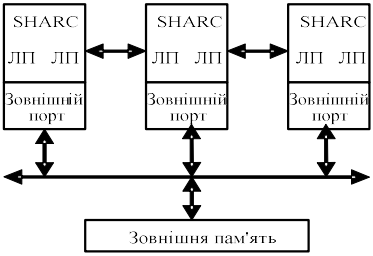


Рис. 3.22. Кластерна мікропроцесорна система

Матрична топологія – це декілька процесорів, що з’єднані, як показано на рис. 3. 23. Така топологія може бути доволі ефективною у ряді задач. Наприклад, у системах відображення даних у радіо- та гідролокації.

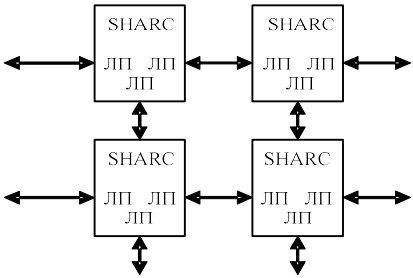


Рис. 3.23. Матрична топологія мікропроцесорної системи

На рис. 3. 23 зображена двовимірна матриця ЦПОС. Лінійні порти забезпечують зв’язок з сусідніми процесорами та маршрутизацію даних. Один провідний процесор забезпечує потік команд, який виконує масив процесорів.

Декілька сигнальних процесорів ADSP-2106x можуть сумісно використовувати зовнішню шину без додаткової схемної частини для арбітражу шин. Логіка арбітражу, вбудована до мікросхеми, дозволяє з'єднати до шести ADSP-2106x та один головний (host) комп'ютер.

Кожен з процесорів багатопроцесорної системи шляхом аналізу відповідних бітів регістру SYSTAT, може визначити, який ЦПОС є провідним у системі. Ці біти показують номер провідного процесора.

Використовуючи команди перевірки виконання умови (If condition computer), можна програмно визначити, чи є даний процесор провідним (Bus master), чи підлеглим. Мнемоніка асемблеру з визначення провідного процесора – BM (або NBM – не є провідним).

У більшості багатопроцесорних систем бажано обмежити час, протягом якого один процесор може займати системну шину. Для цього програмним способом завантажують до спеціального регістра BMAX максимальну кількість циклів мінус два, після яких процесор втрачає статус провідного. Коли в результати декремента це число стане таким, що дорівнює нулю, поточний провідний процесор дозволяє передачу управління системною шиною іншому процесору. Цикл, у якому провідний процесор передає управління шиною іншому ЦПОС, називається циклом переходу шини. Будь-який ADSP-2106x у процесі роботи у складі багатопроцесорної системи може визначити, коли настає цикл переходу шини та який процесор стає провідним після його завершення.

У багатопроцесорних систем провідний процесор може здійснювати безпосередній доступ до внутрішньої пам'яті та регістрів вводу/виводу інших процесорів. Він здійснює читання/запис комірок пам'яті, що входять до простору пам'яті мультипроцесорної системи. Такий режим доступу отримав назву "пряме читання" або "прямий запис". Коли відбувається прямий запис у підлеглий ADSP-2106x, то дані та адреси зберігаються у 4-рівневому FIFO-буфері процесора вводу/виводу. У випадку переповнення буфера підлеглий процесор знімає сигнал підтвердження на запис, поки буфер не звільниться. Такий доступ є невидимим для підлеглих процесорів тому, що він відбувається через зовнішній порт та внутрішньокристалічну шину процесора вводу/виводу. Це дозволяє підлеглому процесору продовжувати виконання програми без перерв. Процес прямого читання не конвеєризується, тому цей режим не є найбільш ефективним способом переміщення даних з підлеглого процесора.

Для одночасного переміщення даних у всі процесори використовується широкомовний запис. Провідний процесор може здійснювати широкомовний запис в однакові області пам'яті або регістри процесора вводу/виводу у всі підлеглі процесори і в себе самого. Такий вид

запису використовується для одночасного завантаження даних та програм до декількох процесорів.

Найбільш ефективним способом переміщення даних між провідним та підлеглими процесорами є прямий доступ до пам'яті (ПДП). Провідний процесор може ініціювати операції ПДП шляхом запису управляючих слів до регістрів параметрів та управління ПДП, а також встановити зовнішній порт каналів ПДП для пересилання блоків даних як із так і до внутрішньої пам'яті підлеглого процесора. Після установлення каналу ПДП провідний ЦПОС може читати/писати із відповідного буфера підлеглого процесора або може запрограмувати для цього свій власний контролер ПДП. Провідний процесор може встановити зовнішній порт ПДП для переміщення даних до зовнішньої пам'яті, вирисовуючи лінії запиту та підтвердження ПДП.

Для розподілу обчислювальних ресурсів та синхронізації у багатопроцесорній системі використовуються семафори. Обидві пам'яті, зовнішня і вбудована в ADSP-2106x, доступні з боку будь-якого процесора, тому семафори можуть бути розташовані практично де завгодно. На практиці для семафорів частіше всього використовуються регістри загального призначення процесора вводу/виводу MSGR0 – MSGR7. Використовуючи можливість блокування шини, ADSP-2106x може читати та модифікувати семафор на протязі однієї команди.

Для передачі управління у багатопроцесорній системі ADSP-2106x використовує векторні переривання. Провідний процесор видає вектор переривання підлеглому процесору шляхом запису адреси процедури обслуговування переривання до регістру VIRT підлеглого процесора. Це обумовлює переривання з високим пріоритетом, яке переключає підлеглий процесор на виконання процедури, що потрібна. Молодші 24 біта вектора містять адресу підпрограми, а старші 8 біт можуть бути додатково задіяні як дані для читання процедурою обслуговування переривання.

Контрольні питання

1. Основні принципи SHARC-архітектури.
2. Склад та призначення основних модулів мікропроцесорного ядра ADSP-2106x.
3. Призначення основних блоків управління: генераторів адрес (ГА-1 та ГА-2), контролера мікропрограм, кеша команд.
4. Організація ОЗП в ADSP-2106x
5. Організація зовнішніх зв'язків в ADSP-2106x. Призначення процесора вводу/виводу, портів зв'язку та зовнішньої системної шини.
6. Класифікація команд ЦПОС ADSP-2106x. Склад та призначення регістрів кожної групи команд.

7. Класифікація програмно доступних регістрів ЦПОС ADSP-2106x. Склад та призначення регістрів кожної групи регістрів.

8. Реалізація розподілу адресного простору пам'яті в ADSP-2106x.

9. Призначення полів E, M та S у форматі адресації ADSP-2106x.

10. Основні способи об'єднання ADSP-2106x у багатопроцесорні системи.

3.4. Система команд та особливості програмування

Системи команд різних сімейств ЦПОС мають свої особливості, але основні принципи та мнемоніка основних команд залишаються спільною. Даний розділ розглядає питання адресації та програмування ПОС на прикладі ЦПОС TMS320C2x. При виконанні будь-якої команди процесор звертається до пам'яті програм та пам'яті даних. Пам'ять програм містить команди, що виконуються процесором, а пам'ять даних – операнди, над якими здійснюються необхідні дії, що задаються полем «код операції» (КОП) команди. Спосіб визначення адреси операнда чи адреси передачі управління називають режимом адресації. TMS320C2x підтримує режими трьох видів адресації: прямої, непрямої, безпосередньої [23].

3.4.1. Режими адресації та формати команд

У режимі прямої адресації сім молодших розрядів слова команди об'єднуються з дев'ятьма розрядами регістру покажчика сторінок (DP) для отримання повної 16-розрядної адреси пам'яті даних (dma). Нульове значення сьомого біта слова команди визначає режим прямої адресації (рис. 3. 24).

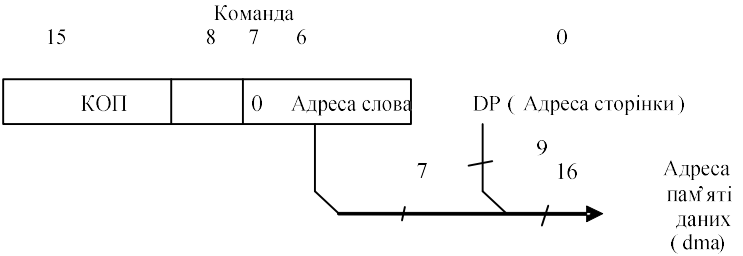


Рис. 3.24. Пряма адресація

Покажчик сторінок дозволяє обрати одну з 512 сторінок пам'яті даних. Кожна сторінка містить 128 слів. Адреса слова на сторінці визначається молодшими розрядами команди. Завантаження покажчика сторінок DP виконується командами LDP (завантажити покажчик сторінок DP) та LDPK

(завантажити покажчик сторінок безпосередньо). Покажчик сторінок не ініціалізується при скиданні процесора.

У випадку непрямої адресації адреса пам'яті даних визначається вмістом одного з допоміжних регістрів AR0 – AR7. Вибір необхідного регістру здійснюється за допомогою 3-розрядного покажчика допоміжного регістру ARP, до якого вміщується число, що визначається трьома молодшими бітами (NARP) команди (рис. 3. 25). При описі команд для позначення поточного допоміжного регістру, на який посилається покажчик ARP, використовується запис AR(ARP).

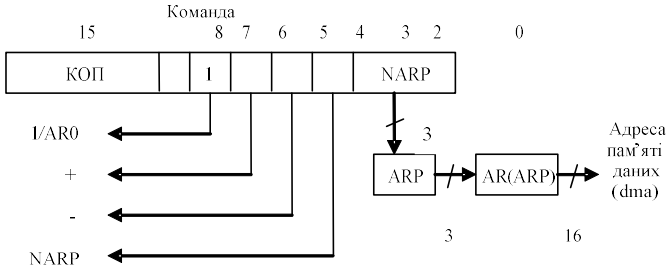


Рис. 3.25. Непряма адресація

Над вмістом регістрів AR0 – AR7 можна виконувати арифметичні операції за допомогою арифметичного пристрою допоміжних регістрів (ARAU). Зазначений пристрій виконує операції над вмістом регістрів AR0 – AR7 паралельно з операціями, що відповідають виконанню поточної команди у ЦАЛП. Наявність ARAU дозволяє організувати сім різних варіантів непрямої адресації. При запису команд з непрямою адресацією на мові асемблеру використовують спеціальні позначення, наведені в табл. 3. 15.

Таблиця 3. 15 – Режими непрямої адресації

Позначення	Назва режиму адресації
*[,NARP]	Адресація без зміни (AR)
*-[,NARP]	Адресація із зменшенням (AR) на 1
*+[,NARP]	Адресація із збільшенням (AR) на 1
*0-[,NARP]	Адресація із зменшенням (AR) на (AR0)
*0+[,NARP]	Адресація із збільшенням (AR) на (AR0)
*BR0-[,NARP]	Адресація із зменшенням (AR) на (AR0) та зворотним розповсюдженням переносу
*BR0+[,NARP]	Адресація із збільшенням (AR) на (AR0) та зворотним розповсюдженням переносу

У цій таблиці операнад NARP відповідає новому значенню покажчика ARP, що буде встановлене після виконання команди. Операнд, який міститься у квадратних дужках, не є обов’язковим. Запис (AR) означає вміст

допоміжного регістру. Зменшення або збільшення значення допоміжного регістру відбувається після його використання поточною командою.

Режими адресації, засновані на зворотному розповсюдженні переносу, називають біт-інверсною адресацією та використовують при упорядкуванні даних в програмах ШПФ. Як показано на рис. 3. 25, одиниця в сьомому розряді команди означає непряму адресацію. Розряди 4, 5, 6 визначають один із різновидів непрямої адресації. Розряд 3 визначає використання поля NARP. Якщо значення цього поля дорівнює нулю, то поле NARP ігнорується та значення ARP залишається незмінним.

При безпосередній адресації операнд міститься прямо в команді. Розрізняють короткі (одне слово) (рис. 3. 26, *а*) та довгі (два слова) (рис. 3. 26, *б*) команди з безпосереднім операндом. У довгих командах у першому слові міститься код операції, а в другому – безпосередній операнд.

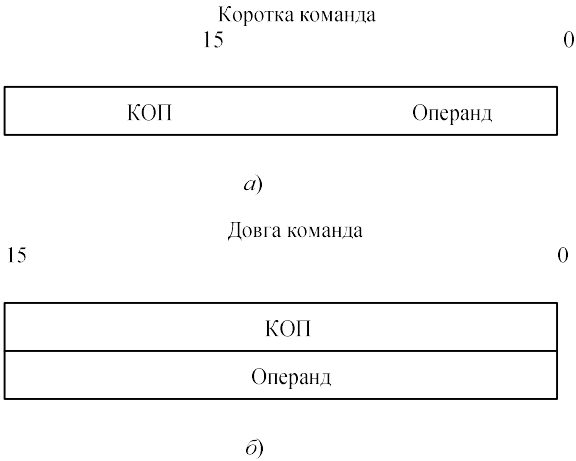


Рис. 3.26. Безпосередня адресація

Для запису команди будемо використовувати розширені формули Бекуса–Наура. Формат асемблерних команд процесора залежить від режиму адресації:

- <команда з прямою адресацією>::=
  [<мітка>] <мнемоніка> <ас> [,<зсув>]

- <команда з непрямою адресацією>::=
  [<мітка>] <мнемоніка> <ка> [,<зсув> [,<NARP>]]
- <команда з безпосередньою адресацією>::=
  [<мітка>] <мнемоніка> [<константа>]

Мітка – визначене користувачем ім’я. Значенням мітки є поточне значення лічильника команд. Мітки використовуються у програмі для передачі управління та не є обов’язковими.

Мнемоніка являє собою заздалегідь визначене ім’я, яке ідентифікує код машинної операції (КОП). Як мнемоніки використовують скорочені англійські слова або абревіатури англійських назв команд, що передають зміст основної функції команди. Наприклад, ADDH (add to high accumulator) – складання із старшими розрядами акумулятора, LAR (load auxiliary register) – завантажити допоміжний регістр.

Після мнемоніки записується операнд команди. Значеннями операндів <ас>, <зсув>, <NARP>, <константа> є цілі числа. При цьому <ас> задає адресу слова на поточній сторінці пам'яті даних. Значення <ас> лежить у діапазоні 0 – 127. Необов’язковий операнд <NARP> визначає нове значення покажчика допоміжного регістру ARP та змінюється у діапазоні 0 – 7.

Операнд <константа> – ціле 8-розрядне або 16-розрядне число. Операнд <зсув> указує кількість двійкових розрядів, на яку необхідно зсунути дане число, що бере участь у операції. Значення <зсув> залежить від команди, що виконується. Зазвичай значення поля <зсув> лежить у межах від 0 до 15.

Для команд з непрямою адресацією операнд <ка> визначається таким чином:

<ка>::=*|*+|*0+|*BR0+|*BR0-

При запису асемблерних команд операнди відокремлюються один від одного комою. Нижче наведено приклади запису команд відповідно до сформульованих правил:

```
ADD DAT1,3
ADD *,3
ADDK 5h
```

У цих прикладах ADD та ADDK є мнемоніками команд. Операнд DAT1 відповідає полю <ас> формату команди з прямою адресацією. Число 3 визначає зсув операнда, що адресується на 3 розряди вліво. Операнд 5h – 16-річна константа.

3.4.2. Класифікація команд

Систему команд ЦПОС можна розподілити на такі групи команд:

• команди акумулятора, призначені для виконання арифметичних та логічних операцій, виконання завантаження та збереження вмісту акумулятора;

• команди допоміжних регістрів, що виконують арифметичні операції над вмістом допоміжних регістрів, та операції порівняння допоміжних регістрів;

• команди T- та P-регістрів та команди множення, що призначені для обміну даними між пам'яттю та регістрами помножувача T та P. Деякі команди цієї групи розповсюджують свої дії не тільки на T- та P-регістри, але й на акумулятор, що дозволяє разом з передачею даних виконувати операції складання та віднімання акумулятора з регістрами T та P;

• команди вводу/виводу та пересилань даних, до яких входять команди пересилань блоків даних як всередині пам'яті даних, так і між пам'яттю програм та пам'яттю даних. До цієї групи команд також входять команди роботи з послідовним та паралельним портами вводу/виводу;

• команди управління, що включають команди управління стеком, кількістю повторень циклів, перериваннями. До цієї групи також входять команди завантаження та збереження регістрів стану та команди управління окремими бітами: C, OVM, HM, TC [23].

Команди акумулятора. Перелік арифметичних команд акумулятора наведено в табл. 3. 16.

Таблиця 3. 16 – Арифметичні команди акумулятора

Команда	Опис
ABS	Абсолютна величина акумулятора
ADD	Складання з акумулятором із зсувом
ADDC	Складання з акумулятором з переносом
ADDH	Складання із старшим словом акумулятора
ADDK	Складання коротке безпосереднє з акумулятором
ADDS	Складання з акумулятором без розширення знака
ADDT	Складання з акумулятором із зсувом, що визначає T-регістр
ADLK	Складання з акумулятором безпосередньої довгої константи із зсувом
SBLK	Віднімання з акумулятора безпосередньої довгої константи із зсувом
SUB	Віднімання з акумулятора із зсувом
SUBB	Віднімання з акумулятора із зсувом
SUBB	Віднімання з акумулятора з переносом
SUBC	Умовне віднімання
SUBH	Віднімання із старшого слова акумулятора
SUBK	Коротке безпосереднє віднімання з акумулятора
SUBS	Віднімання з молодшого слова акумулятора без розширення знака
SUBT	Віднімання з акумулятора із зсувом, що визначає T-регістр
NORM	Нормалізація вмісту акумулятора

Команди цієї групи виконують арифметичні дії над вмістом акумулятора та вмістом заданої комірки пам'яті даних, або над 8- чи 16-розрядною константою. При цьому може враховуватися біт переносу та

виконуватися зсув вмісту заданої комірки пам'яті. Зсув вказується безпосередньо у команді, або визначається вмістом Т-регістру.

Для кожної команди вказується формат та операція, що виконується, яка умовно описується у вигляді деякого виразу, а також у текстовій формі. Нижче наводяться формати деяких команд та приклади їх використання.

Тут і далі, якщо адреса чи ім'я вказані у круглих дужках (), це означає вміст області пам'яті, на яку посилається ім'я чи адреса. Наприклад, запис (AR(ARP)) означає вміст пам'яті, на яку посилається поточний допоміжний регістр, а запис (ACC) – вміст акумулятора. У прикладах лівий стовпчик відображає стан регістрів та пам'яті процесора до виконання команди, а правий – після виконання.

Команда ADD (add to accumulator with shift) – виконує складання з акумулятором із зсувом.

Формат команди:

ADD <ас> [,<зсув>]

ADD <ка> [,<зсув> [,<NARP>]]

Операція реалізує дії (ACC)+(dma)*$2^{зсув}$ → ACC. Вміст пам'яті даних, що визначається адресою в команді, зсувається вліво на кількість розрядів, що відповідає полю <зсув>, а потім додається до вмісту акумулятора. Результат поміщається до акумулятора.

Приклад:

(AR(ARP)=1025		(1025)=4h
(1025)=4h	ADD *,3	(ACC)=28h
(ACC)=8h		(C)=0

Вміст комірки пам'яті за адресою 1025 (4h) зсувається на 3 розряди (дорівнює 20h) та складається із вмістом акумулятора (20h+8h=28h).

Команда ADDC – (add to accumulator with carry) – виконує складання з акумулятором із урахуванням переносу.

Формат команди:

ADDC <ас>

ADDC <ка> [,<NARP>]

Операція реалізує дії (ACC)+(dma)+(C) → ACC. Команда використовується при виконанні арифметичних операцій з підвищеною розрядністю.

Приклад:

(AR(ARP)=1025	ADDC *,3	(1025)=4h
(1025)=4h		(ACC)=29h
(C)=1		(C)=0

Команда ADDH – (add to high accumulator) – виконує складання із старшим словом акумулятора.

Формат команди:

ADDH <ас>

ADDH <ка> [,<NARP>]

Операція реалізує дії (ACC)+(dma)*2^{16} → ACC. Вміст пам'яті складається із старшим словом акумулятора (розряди 31..16). Значення молодшого слова не змінюється. Біт С може бути тільки встановлений даною командою, але не скинутий.

Приклад:

(AR(ARP)=1025 ADDH *	(1025)=4h
(1025)=4h	(ACC)=40008h
(ACC)=8h	(C)=1
(C)=1	

Команда ADDK – (add to accumulator short immediate) – виконує коротке безпосереднє складання з акумулятором.

Формат команди:

ADDK <константа>

Операція реалізує дії (ACC)+8-розрядна константа → ACC.

Команда ADDS – (add to accumulator with sign-extension suppressed) – виконує складання з акумулятором без розширення знаку.

Формат команди:

ADDS <ас>

ADDS <ка> [,<NARP>]

Операція реалізує дії (ACC)+(dma) → ACC, де (dma) – 16-розрядне число без знака. Акумулятор розглядається як число із знаком, вміст за адресою dma – як число без знака.

Команда ADDT – (add to accumulator with shift specified by T-register) – виконує складання з акумулятором із зсувом, що визначається Т-регістром.

Формат команди:

ADDT <ас>

ADDT <ка> [,<NARP>]

Операція реалізує дії (ACC)+(dma)*$2^{Treg(3..0)}$ →ACC. Вміст пам'яті за адресою dma зсувається вліво на кількість розрядів, що визначається чотирма молодшими розрядами Т-регістру, а потім додається до вмісту акумулятора. Команда аналогічна команді ADD.

Команда ADLK – (add to accumulator long immediate with shift) – виконує довге безпосереднє складання з акумулятором із зсувом.

Формат команди:

ADLK <константа> [,<зсув>]

Тут <константа> – це 16-розрядне число. При виконанні команди константа зсувається вліво а потім додається до вмісту акумулятора. Довжина команди 2 слова.

Приклад:

(ACC)=13AFh ADLK 7,8 (ACC)=22FAh
C=0

До групи арифметичних команд входять операції віднімання (табл. 3. 16), що аналогічні операціям додавання. Їх відмінність від розглянутих вище команд полягає в тому, що замість операції додавання виконується операція віднімання.

Приклади використання цих команд:

(AR(ARP)=1041 SUB * (1041)=21h
(1041)=21h (ACC)=24h
(ACC)=45h (C)=1

(ACC)=48h SUBK 11h (ACC)=37h
(C)=1

Одна з команд віднімання не має аналогів серед команд додавання та називається умовним відніманням:

SUBC (conditional subtract)

SUBC <ac>

SUBC <ка> [,<NARP>]

Команда SUBC виконує умовне віднімання, що використовується при виконанні операцій ділення. 16-розрядне ділиме поміщається до молодшого слова акумулятора, а старше слово обнуляється. Дільником є вміст визначеної комірки пам'яті даних. Команда SUBC виконується 16 разів при 16-розрядному діленні. Після виконання команди SUBC результат ділення (ціла частина) буде знаходитися у молодшому слові акумулятора, а залишок – у старшому. Команда дозволяє отримати коректні результати, якщо числа, що діляться, є додатними. Якщо ділиме вміщує менш ніж 16 розрядів, то його можна попередньо зсунути вліво. При цьому кількість повторів команди SUBC скорочується на кількість зсувів.

Приклад:

(AR(ARP)=1025 PRTK 15 (1025)=6h
(1025)=6h SUBC * (ACC)=2000Eh
(ACC)=56h (C)=1

У цьому прикладі команда PRTK забезпечує повторне виконання команди PRTK 16 разів.

Команда ABS обчислює абсолютну величину вмісту акумулятора. Виконується операція |(ACC)| → ACC. Якщо вміст акумулятора більше або дорівнює 0, то його значення залишається незмінним. Якщо вміст акумулятора менше 0, то до акумулятора поміщається його двійкове доповнення.

Приклади:

(ACC)=2341h ABS (ACC)=2341h

C=0

(ACC)=AFFF FFFFh ABS (ACC)=5000 0001h

C=0

Команда NORM (normalize contents of accumulator) нормалізує вміст акумулятора.

Формат команди:

NORM <ка>

Нормалізація числа з фіксованою точкою потребує виділення в ньому мантиси та порядку. Для визначення порядку необхідно визначити кількість бітів розширення знака. Ці дії всередині команди NORM реалізуються за допомогою операції (ACC(31)XOR(ACC(30)). Якщо розряди акумулятора 30 та 31 однакові, то обидва вони є знаковими, У цьому випадку вміст акумулятора зсувається вліво для того, щоб вилучити зайвий знаковий біт. Потім модифікується значення допоміжного регістру відповідно до обраного режиму непрямої адресації для того, щоб отримати в ньому значення порядку. Зазвичай воно інкрементується.

Для повної нормалізації 32-розрядного значення може знадобиться багатократне виконання команди NORM.

Перелік логічних операцій акумулятора наведено в табл.. 3. 17. Ця група команд реалізує порозрядні операції булевої алгебри над вмістом акумулятора та вмістом комірки пам'яті даних, або константою, що задана безпосередньо в команді.

Таблиця 3. 17 – Логічні команди акумулятора

Команда	Опис
AND	Логічне "І" з акумулятором
ANDK	Логічне безпосереднє "І" з акумулятором із зсувом
NEG	Логічне "НІ" акумулятора
OR	Логічне "АБО" з акумулятором
ORK	Логічне безпосереднє "АБО" з акумулятором із зсувом
XOR	Виключне "АБО" з акумулятором
XORK	Виключне "АБО" акумулятора і безпосередньої константи. із зсувом
CMPL	Доповнення (інверсія) акумулятора
ROL	Циклічний зсув акумулятора вліво
ROR	Циклічний зсув акумулятора вправо
SFL	Зсув акумулятора вліво
SFR	Зсув акумулятора вправо

Команда NEG (negative accumulator) заміщає значення акумулятора на його двійкове доповнення. Виконується операція (ACC) → ACC.

Приклад:

(ACC)=AFFF FF34h NEG (ACC)=5000 00CCh

C=0

Команда AND (and with accumulator) – виконує логічне «І» з акумулятором.

Формат команди:

AND <ас>

AND <ка> [,<NARP>]

Команда реалізує дії (ACC(15..0))AND(dma) → ACC.

Приклад:

DAT1=16	AND DAT1	(DAT1)=FFh
(DP)=4		(ACC)=0000 0078h
(DAT1)=FFh		(C)=1
(ACC)=5432 1678h		

Команда ANDK (and immediate with accumulator with shift) – виконує безпосереднє логічне «І» з акумулятором із зсувом.

Формат команди:

ANDK <константа> [,<зсув>]

Команда реалізує дії (ACC(30..0))AND константа*$2^{зсув}$) → ACC(30..0), 0→ ACC(31); 16-розрядна константа, що безпосередньо задана у команді, зсувається вліво на кількість розрядів, що визначена полем команди <зсув>. Над константою та вмістом акумулятора виконується логічна операція «І». Біти, розташовані поза межами, що визначаються даною константою, вважаються нульовими і тому відповідні розряди після виконання команди стають такими, що дорівнюють нулю. Старший розряд акумулятора дорівнює нулю завжди. Довжина команди два слова.

Приклад:

(ACC)=5423 1678h ANDK FFFFh,8 (ACC)=0032 1600h

Команди OR та ORK реалізують операції логічного «АБО» з акумулятором та безпосереднього логічного «АБО» з акумулятором із зсувом відповідно. Команди XOR та XORK реалізують операції виключного «АБО» з акумулятором та безпосереднього виключного «АБО» з акумулятором із зсувом відповідно. Формати команд аналогічні форматам команд AND та ANDK, де в тексті команди замість мнемоніки AND використовується мнемоніка OR чи XOR, а замість ANDK – мнемоніка ORK чи XORK відповідно.

Команда CMPL (complement accumulator) замінює вміст акумулятора його порозрядною логічною інверсією.

Команди ROL (rotate accumulator left) та ROR (rotate accumulator right) виконують циклічний зсув вмісту акумулятора на один біт вліво та вправо відповідно. При цьому виконуються такі операції:

• при зсуві вліво – операції ACC(31) → C; (ACC(30..0)) → ACC(31..1); (C) → ACC(0);

• при зсуві вправо – операції ACC(0) → C; (ACC(31..1)) → ACC(30..0); (C) → ACC(31).

Команди SFL (shift accumulator left) та SFR (shift accumulator right) виконують зсув вмісту акумулятора на один біт вліво та вправо відповідно. При цьому виконуються такі операції:

• при зсуві вліво – операції ACC(30) → C; (ACC(30..0)) → ACC(31..1); 0 → ACC(0);

• при зсуві вправо – операції ACC(0) → C; (ACC(31..1)) → ACC(30..0). Якщо біт режиму SXM=0, то 0 → ACC(31), інакше – (ACC(31)) → ACC(31).

У табл.. 3. 18 наведено перелік команд, що використовуються для завантаження та збереження вмісту акумулятора. Команди цієї групи заносять до акумулятора значення, що вилучається з пам'яті даних, або задається безпосередньо у команді, а також зберігають значення акумулятора у пам'яті даних.

Таблиця 3. 18 – Команди завантаження та збереження вмісту акумулятора

Команда	Опис
LAC	Завантажити акумулятор із зсувом
LACK	Завантажити акумулятор безпосередньо (коротка константа)
LALK	Завантажити акумулятор безпосередньо (довга константа)
LACT	Завантажити акумулятор із зсувом, що визначається Т-регістром
SACH	Зберегти старше слово акумулятора із зсувом
SACL	Зберегти молодше слово акумулятора із зсувом
ZAC	Обнулити акумулятор
ZALH	Обнулити молодше слово акумулятора і завантажити число в старше слово акумулятора
ZALR	Обнулити молодше слово акумулятора і завантажити число в старше слово акумулятора з обертанням
ZALS	Обнулити акумулятор, завантажити молодше слово акумулятора без розширення знака

Команда LAC (load accumulator with shift) дозволяє завантажити акумулятор із зсувом.

Формат команди:

LAC <ас> [,<зсув>]

LAC <ка> [,<зсув> [,<NARP>]]

Виконується операція (dma)*$2^{зсув}$ → ACC. Вміст пам'яті, що адресується, зсувається вліво та вміщується до акумулятора. При зсуві молодші розряди заповнюються нулями, а старші – знаковим бітом (якщо біт режиму SXM=1) або нулями якщо (SXM=0).

Приклад:

(AR(ARP))=1027	LAC *,4,5	(1027)=3h
(1027)=3h		(ACC)=30h
(ACC)=5432 1678h		(ARP)=5

Команда LACK (load accumulator immediate short) поміщає коротку (8-розрядну) константу до акумулятора. Старші 24 розряди акумулятора обнуляються. Виконується операція 8-розрядна константа → ACC.

Формат команди:

LACK <константа>

Команда LACT (load accumulator with shift specified by T register) забезпечує завантаження акумулятора із зсувом, що визначається Т-регістром.

Формат команди:

LACT <ас>

LACT <ка> [,<NARP>]

Виконується операція (dma)*$2^{Treg(3..0)}$ → ACC. Акумулятор завантажується значенням, зчитаним з пам'яті даних та зсунутим вліво на кількість розрядів, що визначаються чотирма молодшими розрядами Е-регістру. Якщо біт режиму SXM=1, то значення, що завантажується до акумулятора, доповнюється зліва до необхідної кількості розрядів знаковим бітом.

Приклад:

(AR(ARP))=1300 LACT *	(1300)=276h
(1300)=276h	(ACC)=2760h
(ACC)=8432 1AB5h	(T)=2014h
(T)=2014h	

Команда LALK (load accumulator long immediate with shift) виконує безпосереднє завантаження акумулятора довгою константою (16 розрядів) із зсувом.

Формат команди:

LALK <константа> [,<зсув>]

Зсунуте вліво значення 16-розрядної константи поміщається до акумулятора. Кількість розрядів, на яку зсувається константа, визначається полем <зсув>. Виконується операція (16-розрядна константа)*$2^{зсув}$ → ACC. Якщо біт SXM=1, то значення, що завантажується до акумулятора, доповнюється зліва до необхідної кількості розрядів знаковим бітом. Значення старшого біта акумулятора може бути встановлене у 1 тільки у режимі SXM=1. Тоді до акумулятора може бути завантажено значення з діапазону -32768..32767. Якщо SXM=0, то це значення належить діапазону 0..65535. Довжина команди два слова.

Приклади:

SXM=1	LALK F634h,4	(ACC)=FFFF 6340h
(ACC)=5432 6781h		
SXM=0	LALK F634h,4	(ACC)=000F 6340h
(ACC)=5432 6781h		

Команда ZAC (zero accumulator) обнуляє акумулятор. Виконується операція 0 → ACC. Дана команда відповідає команді LACK 0.

Команда ZALH (zero low accumulator and load high accumulator) дозволяє обнулити молодше слово акумулятора та завантажити старше слово значенням, яке вилучається з пам'яті даних, що адресується.

Приклад:

(AR(ARP))=4033 ZALH * (4033)=2A01h
(4033)=2A01h (ACC)=2A01 0000h
(ACC)=FFFF FFFFh

Команда ZALR (zero low accumulator, load high accumulator with rounding) дозволяє обнулити молодше слово акумулятора та завантажити старше слово з округленням.

Формат команди:

ZALR <ac>
ZALR <ка> [,<NARP>]

Виконуються операції: 8000h → ACC; (dma) → ACC(31..16).

Приклад:

(AR(ARP))=4033 ZALR * (4033)=2A01h
(4033)=2A01h (ACC)= 2A01 8000h
(ACC)=AFAF AFAFh

Команда ZALS (zero accumulator, load low accumulator with sign extension suppressed) дозволяє обнулити акумулятор та завантажити молодше слово без розширення знака.

Формат команди:

ZALS <ac>
ZALS <ка> [,<NARP>]

Виконуються операції 0 → ACC(31..16); (dma) → ACC(15..0). Значення з пам'яті даних завантажується до молодшого слова акумулятора. Старше слова акумулятора обнуляється.

Приклад:

(AR(ARP))=4033 ZALS * (4033)=2A01h
(4033)=2A01h (ACC)=0000 2A01h
(ACC)=FAFA FAFAh

Команди додаткових (допоміжних) регістрів. Ця група команд виконує арифметичні операції над вмістом допоміжних регістрів та операції порівняння допоміжних регістрів. Перелік команд цієї групи наведено у табл. 3.19. Команди можна розподілити на 2 підгрупи: арифметичні та завантаження і збереження вмісту регістрів.

Таблиця 3. 19 – Команди додаткових (допоміжних) регістрів

Команда	Опис
ADRK	Складання коротке безпосереднє із допоміжним регістром
CMPR	Порівняння допоміжного регістра з допоміжним регістром ARO
LAR	Завантаження допоміжного регістру
LARK	Завантаження допоміжного регістру безпосереднє, коротке
LARP	Завантаження покажчика допоміжного регістру
LDP	Завантаження покажчика сторінок пам'яті даних
LDPK	Завантаження покажчика сторінок пам'яті даних безпосереднє
LRLK	Завантаження допоміжного регістру безпосереднє, довге
MAR	Модифікація допоміжного регістру
SAR	Збереження допоміжного регістру
SBRK	Віднімання коротке безпосереднє із допоміжного регістра
BLKD	Переміщення блока з пам'яті даних в пам'ять даних

Команда ADRK (add to auxiliary register short immediate) здійснює коротке безпосереднє складання допоміжного регістру.

Формат команди:

ADRK <константа>

Виконується операція (AR(ARP)) + 8-розрядна константа → (AR(ARP)). Вміст поточного допоміжного регістру складається з 8-розрядною константою. Результат поміщається до поточного допоміжного регістру.

Приклад:

(AR(ARP))=2341h ADRK 40h (AR(ARP))=2381h

Команда SBRK (subtract from auxiliary register short immediate) здійснює коротке безпосереднє віднімання допоміжного регістру.

Формат команди:

SBRK <константа>

Виконується операція (AR(ARP)) - 8-розрядна константа → (AR(ARP)). З вмісту поточного допоміжного регістру віднімається 8-розрядна константа. Результат поміщається до поточного допоміжного регістру.

Приклад:

(AR(ARP))=0h SBRK 03Fh (AR(ARP))=FFC1h

Команда MAR (modify auxiliary register) виконує модифікацію вмісту допоміжного регістру.

Формат команди:

MAR <ас>

MAR <ка> [,<NARP>]

У випадку прямої адресації команда відповідає пустій операції. У випадку непрямої адресації здійснюється модифікація допоміжного регістру та покажчика ARP. Ніяких операцій з пам'яттю не виконується. Якщо у команді задано значення NARP, то старий вміст ARP копіюється до поля

ARB регістру стану ST1. Слід відзначити, що дії, які виконуються командою MAR, можна також виконати за допомогою будь-якої команди з безпосередньою адресацією.

Приклад:

(ARP)=1h MAR *+,4 (AR1)=38h
(AR1)=37h (ARP)=4

Команда CMPR (Compare auxiliary register with auxiliary register AR0) порівнює допоміжний регістр з допоміжним регістром AR0.

Формат команди:

CMPR <константа>

Виконується операція if ARn <константа> AR0 then 1 → TC else 0 → TC. Команда виконує порівняння вмісту поточного допоміжного регістру з вмістом допоміжного регістру AR0. Вид операції порівняння залежить від константи, яка може набувати значення з діапазону 0..3. При цьому має місце наступна відповідність між значенням константи та видом операції:

• 0 – (ARn=AR0);
• 1 – (ARn<AR0);
• 2 – (ARn>AR0);
• 3 – (ARn≠AR0).

Якщо результат порівняння буде мати значення «істина», то встановлюється біт TC регістру стану ST1.

Приклад:

(AR0)=00F7h CMPR 1 (AR0)=00F7h
(AR(ARP))=00A1h (AR(ARP))=00A1h
(TC)=0h (TC)=1h

Команди T- та P-регістрів та команди множення. Перелік команд T- та P-регістрів наведено у табл.. 3. 20.

Таблиця 3. 20 – Команди T- та P-регістрів

Команда	Опис
APAC	Додати P-регістр до акумулятора
LPH	Завантаження старших розрядів P-регістру
LT	Завантаження T-регістру
LTA	Завантаження T-регістру і складання попереднього добутку
LTD	Завантаження T-регістру, складання попереднього добутку, пересилання даних
LTP	Завантаження T-регістру і збереження P-регістру в акумуляторі
LTS	Завантаження T-регістрау, віднімання попереднього добутку
PAC	Завантаження P-регістру в акумулятор
SPAC	Віднімання P-регістру з акумулятора
SPH	Збереження старшого слово P-регістру
SPL	Збереження молодшого слово P-регістру
SPM	Завдання режиму зсуву на виході P-регістру

Команди цієї групи призначені для обміну даними між пам'яттю та регістрами помножувача T та P. Деякі з команд даної групи розповсюджують свою дію не тільки на регістри T та P, але й на акумулятор. Це дозволяє поряд з передачею даних виконувати операції складання та віднімання акумулятора з регістрами T та P. Перелік команд наведено у табл.. 3. 20.

Команда LPH (load high P register) – завантажує старше слово P-регістра.

Формат команди:

LPH <ac>

LPH <ка> [,<NARP>]

Виконується операція (dma) → Preg(31..16). Вміст пам'яті даних за адресою dma завантажується до старшого слова P-регістру.

Приклад:

(AR(ARP))=1025 LPH * (1025)=3A64h

(1025)=3A64h (P)=3A64 5214h

(P)=3041 5214h

Команда LT (load T register) завантажує T-регістр.

Формат команди:

LT <ac>

LT <ка> [,<NARP>]

Виконується операція (dma) → Treg. Вміст пам'яті даних за адресою dma завантажується до T-регістру.

Команда LTA (load T register and accumulate previous product) завантажує T-регістр та додає попередній добуток.

Формат команди:

LTA <ac>

LTA <ка> [,<NARP>]

Виконуються операції (dma) → Treg; (ACC)+(Preg)*$2^{\text{зсув}}$ → (ACC). Вміст пам'яті даних за адресою dma завантажується до T-регістру. Вміст регістру P зсувається відповідно до значення бітів PM регістру стану ST1 та додається до вмісту акумулятора. Якщо PM=00, то зсув не відбувається. Якщо PM=01 або PM=10, то вміст регістру P зсувається вліво відповідно на 1 або на 4 розряди, при цьому молодші біти заповнюються нулями. У випадку коли PM=11, виконується зсув вправо на шість розрядів, старші біти заповнюються значенням знакового біта.

Зсув відбувається у процесі передачі вмісту P-регістру до акумулятора, тому операції зсуву не впливають на вміст P-регістру.

Зсув вліво використовується для вирівнювання значень добутків в операціях з дробовими числами. Зсув вправо на шість розрядів дозволяє виконати 128 операцій типу «множення з накопиченням» без небезпеки переповнення.

Команда LTA впливає на біти C та OV. Останній встановлюється, якщо задано режим OVM:

(AR(ARP))=768 LTA *	(768)=30h
(768)=30h	(T)=30h
(T)=4h	(P)=AFh
(P)=AFh	(ACC)=B2h
(ACC)=3h	(C)=0
PM=00	

Команда LTD (load T register, accumulate previous product and move data) завантажує T-регістр, додає попередній добуток, переслає дані.

Формат команди:

LTD <ac>

LTD <ка> [,<NARP>]

Виконуються операції (dma) → Treg; (ACC)+(Preg)*$2^{зсув}$ → (ACC); (dma) → (dma)+1. Дії, що виконуються командою, аналогічні діям команди LTA. Додатково виконується пересилання даних з комірки з адресою (dma) до комірки з адресою (dma)+1. Команда може використовуватися при роботі з блоками пам'яті B0, B1, B2.

Команда LTP (load T register and store P register in accumulator) завантажує T-регістр та зберігає вміст P-регістру в акумуляторі.

Формат команди:

LTP <ac>

LTP <ка> [,<NARP>]

Виконуються операції (dma) → Treg; (Preg)*$2^{зсув}$ → (ACC). Вміст пам'яті (dma) завантажується до T-регістру, вміст регістру P пересилається до акумулятора. При цьому виконується зсув P-регістру на кількість розрядів, що визначається полем PM.

Команда LTS (load T register, subtract previous product) завантажує T-регістр, віднімає попередній добуток.

Формат команди:

LTS <ac>

LTS <ка> [,<NARP>]

Виконуються операції (dma) → Treg; (ACC)-(Preg)*$2^{зсув}$ → (ACC). Команда аналогічна команді LTA. Відмінність полягає в тому, що замість складання виконується віднімання.

Команда PAC (load accumulator with P register) завантажує до акумулятора вміст регістру P. Виконується операція (Preg)*$2^{зсув}$ → (ACC). Вміст регістру P зсувається на 1, 4 або 6 розрядів відповідно до значення бітів PM та поміщається до акумулятора.

Команда SPM (set P register output shift mode) встановлює режим зсуву регістру P.

Формат команди:

SPM <константа>

Виконується операція 2-розрядна константа → PM. Константа, що задається в команді, копіюється у поле PM регістру стану ST1. Поле PM визначає зсув вмісту регістру при передачі його до акумулятора.

Команди множення забезпечують виконання множення, множення з накопиченням результатів в акумуляторі, множення з накопиченням та пересиланням даних. Зазвичай помножуються аргументи, що знаходяться у пам'яті даних, або у регістрі T та пам'яті даних. Перелік команд цієї групи наведено у табл.. 3.21.

Таблиця 3. 21 – Команди множення

Команда	Опис
MAC	Множення та додавання
MACD	Множення та додавання з рухом даних
MPY	Множення
MPYA	Множення і складання з акумулятором попереднього добутку
MPYK	Множення безпосереднє
MPYS	Множення і віднімання з акумулятора попереднього добутку
MPYU	Множення без урахування знака
SQRA	Піднесення до квадрата і складання результату попереднього множення з акумулятором
SQRS	Піднесення до квадрата і віднімання результату попереднього множення з акумулятора

Команда MAC (multiply and accumulate) виконує множення з накопиченням.

Формат команди:

MAC <pma>, <ac>

MAC < pma>, <ка> [,<NARP>]

Виконуються операції (ACC)+(Preg)*$2^{зсув}$ → ACC; (dma) → Treg; (pma)*(dma) → Preg. За командою MAC обчислюється добуток вмісту пам'яті даних за адресою dma та пам'яті програм за адресою pma. Команда також забезпечує накопичення попереднього добутку в акумуляторі. При цьому зсув вмісту регістру P виконується відповідно до вмісту поля PM регістру стану ST1. Адреса пам'яті програм pma, що задається у команді, лежить у діапазоні 0..65535.

Команда MAC може використовуватися сумісно з командами циклу RPT та RPTK. У цьому випадку після кожного поточного виконання MAC-операції модифікується вміст допоміжного регістру відповідно до значення поля <ка>, а також інкрементується вміст лічильника команд: (PFC)+1 → PFC. Це дозволяє адресувати ланцюжок операндів з пам'яті програм.

Зазвичай команда MAC займає два слова та для її виконання потрібні два командних цикли. Але у випадку використання команд RPT та RPTK вона виконується за один цикл.

Нижче наведено фрагмент програми сумісного використання команд MAC та RPTK:

```
SPM 3              ; встановити зсув вправо на 6 розрядів
CNFP               ; B0 – блок пам'яті програм
LARP 3             ; поточний регістр AR3
LRLK 3,300h        ; 300h – початкова адреса B1
RPTK 255           ; повторити 256 разів
MAC FF00h,*+       ; підсумовування добутків
```

У цьому прикладі блок B0 – пам'ять програм. Як пам'ять даних використано блок B1, який адресується за допомогою регістру AR3. Виконується підсумовування добутків, що вилучаються з блоків B0 та B1. Завантаження поля PM регістру ST1 константою 3 (команда SPM 3) дозволяє виключити переповнення акумулятора, яке може виникнути в результаті операції додавання.

Команда MACD (multiply and accumulate with data move) здійснює множення з накопиченням та передачею даних.

Формат команди:

MACD <pma>, <ac>

MACD < pma>, <ка> [,<NARP>]

Виконуються операції (ACC)+(Preg)*$2^{зсув}$ → ACC; (pma)*(dma) → Preg; (dma) → dma+1. Команда MACD аналогічна команді MAC та додатково забезпечує пересилання даних: (dma) → dma+1. Пересилання даних буде виконуватися, якщо як пам'ять даних використовуються блоки B0, B1, B2. В іншому випадку команда виконує ті ж дії, що і MAC.

Команда MPY (multiply) здійснює множення.

Формат команди:

MPY <ac>

MPY <ка> [,<NARP>]

Виконується операція Treg*(dma) → Preg. Вміст регістру T помножується на вміст пам'яті даних за адресою dma. Результат спрямовується до регістру P.

Команда MPYA (multiply and accumulator previous product) здійснює множення та накопичення попереднього добутку.

Формат команди:

MPYA <ac>

MPYA <ка> [,<NARP>]

Виконуються операції (ACC)+(Preg)*$2^{зсув}$ → ACC; (Treg)*(dma) → Preg. Попередній добуток зсувається та додається до вмісту акумулятора.

Вміст регістру T помножується на вміст пам'яті даних за адресою dma. Результат спрямовується до регістру P.

Приклад:

(AR(ARP))=737 MPYA *	(737)=6h
(737)=6h	(T)=7h
(T)=7h	(P)=2Ah
(P)=34h	(ACC)=8Bh
(ACC)=57h	(C)=0
PM=0	

Команда MPYK (multiply immediate) здійснює безпосереднє множення.

Формат команди:

MPYK <константа>

Виконується операція (Treg)*13-розрядна константа → Preg. Вміст регістру T помножується на 13-розрядну константу, значення якої лежить в діапазоні -4096..4096. Перед множенням константа вирівнюється по правій границі, старші біти заповнюються бітом знака.

Приклад:

(T)=9h	MPYK -9	(T)=9h
(P)=3Ch		(P)=FFFF FFAFh

Команда MPYS (multiply and subtract previous product) здійснює множення та віднімання попереднього добутку.

Формат команди:

MPYS <ас>

MPYS <ка> [,<NARP>]

Виконуються операції (ACC)-(Preg)*$2^{\text{зсув}}$ → ACC; (Treg)*(dma) → Preg. Команда аналогічна команді MPYA, тільки замість операції додавання виконується операція віднімання.

Команда MPYU (multiply unsigned) здійснює множення без урахування знака.

Формат команди:

MPYU <ас>

MPYU <ка> [,<NARP>]

Виконується операція: Usgn(Treg)*Usgn(dma) → Preg. Множення вмісту регістру T та вмісту пам'яті даних за адресою dma відбувається без урахування знаків співмножників. Слід додати, що помножувач перемножує 17-бітові аргументи. У випадку команди MPYU старші біти кожного із співмножників, що подаються на вхід помножувача, вважаються нульовими. З командою MPYU не слід використовувати режим PM=3, оскільки при цьому відбувається розширення знака. Команда MPYU може використовуватися для обчислення добутку 32-розрядних аргументів.

Команда SQRA (square and accumulate previous product) підносить до квадрата та накопичує попередній добуток.

Формат команди:

SQRA <ас>

SQRA <ка> [,<NARP>]

Виконуються операції (ACC)+(Preg)*$2^{зсув}$ → ACC; (dma)*(dma) → Preg. Вміст Р-регістру зсувається відповідно до вмісту РМ регістру ST1 та додається до акумулятора. Потім значення (dma) завантажується до Т- регістру, підноситься до квадрата та заноситься до Р-регістру.

Команда SQRS (square and subtract previous product) підносить до квадрата та віднімає попередній добуток.

Формат команди:

SQRS <ас>

SQRS <ка> [,<NARP>]

Виконуються операції (ACC)-(Preg)*$2^{зсув}$ → ACC; (dma)*(dma) → Preg. Команда аналогічна SQRA, тільки замість операції додавання виконується операція віднімання.

Команди вводу/виводу, переходів, управління. Перелік команд вводу/виводу та пересилань даних наведено у табл.. 3. 22.

Таблиця 3. 22 – Команди вводу/виводу та пересилання даних

Команда	Опис
IN	Ввід даних з порту
OUT	Вивід даних у порт
RFSM	Скинути біт режиму кадрової синхронізації послідовного порту
RTXM	Скинути біт режиму передачі послідовного порту
RXF	Скинути зовнішній прапор
SFSM	Встановити біт режиму кадрової синхронізації послідовного порту
STXM	Встановити біт режиму передачі послідовного порту
SXF	Встановити зовнішній прапор
FORT	Формат регістрів послідовного порту
BLKD	Переміщення блока з пам'яті даних в пам'ять даних
BLKP	Переміщення блока з пам'яті програм в пам'ять даних
DMOV	Переміщення даних в пам'яті даних
TBLR	Читати таблицю
TBLW	Записати таблицю

Команда IN (input from port) вводить даних з порту.

Формат команди:

IN <ас>, <PA>

IN <ка>, <PA> [,<NARP>]

Виконуються дії PA → (A3-A0); 0 → (A15-A4); (D15-D0) →dma. Команда копіює 16-розрядне значення, що встановлене на шині даних

D15-D0 зовнішнім пристроєм, до пам'яті даних за адресою dma. Значення поля <PA> лежить у діапазоні 0..15 і являє собою адресу порту.

Приклад:

(AR(ARP))=721 IN *+,PA15 (721)=31h

(721)=0h

(A15-A0)=Fh

(D15-D0)=31h

Команда OUT (output data to port) виводить дані у порт.

Формат команди:

OUT <ac>, <PA>

OUT <ка>, <PA> [,<NARP>]

Виконуються дії PA → (A3-A0); 0 → (A15-A4); (dma) → (D15-D0). Команда встановлює на шині адреси адресу порту PA та виводить на шину даних D15-D0 вміст пам'яті даних за адресою dma.

Команда RFSM (reset serial port frame synchronization mode) зжійснює скидання режиму кадрової синхронізації послідовного порту.

Формат команди:

RFSM

Виконується операція 0 → FSM. Команда встановлює біт FSM регістру стану ST1 в нуль. У цьому режимі не потребуються зовнішні імпульси FSR для ініціалізації операції прийому кожного слова: достатньо тільки одного імпульсу для встановлення «безперервного режиму».

Команда RTXM (reset serial port transmit mode) здійснює скидання режиму передачі послідовного порту.

Формат команди:

RTXM

Виконується операція 0 → TXM. Команда встановлює біт TXM регістру стану ST1 в нуль. У цьому випадку передача запускається, коли приходить імпульс на вхід FXM.

Команда RXF (reset external flag) здійснює скидання зовнішнього прапора.

Формат команди:

RXF

Виконується операція 0 → XF. Команда встановлює вивід XF та біт XF регістру стану ST1 в нуль.

Команда SFSM (set serial port frame synchronization mode) встановлює режим кадрової синхронізації послідовного порту.

Формат команди:

SFSM

Виконується операція 1 → FSM. Команда встановлює біт FSM регістру стану ST1 в нуль. У цьому режимі для прийому даних потребується подача

зовнішніх імпульсів FSR. Якщо TXM=0, то потребується подача імпульсів на вивід FSX. Якщо TXM=1, то імпульси FSX генеруються кожного разу, коли завантажується регістр DXR. Передача ініціюється заднім фронтом цього імпульсу.

Команда SXF (set external flag) встановлює зовнішній прапор.

Формат команди: SXF

Виконується операція 1 → XF. Команда встановлює вивід XF та біт XF регістру стану ST1 в одиницю.

Команда FORT (format serial port registers) здійснює формат регістрів послідовного порту.

Формат команди: FORT <константа>

Виконується операція 1-бітна константа → FO. Біт стану FO завантажується 1-бітною константою. Біт FO використовується для управління форматом регістрів зсуву приймача та передавача послідовного порту. Якщо FO=0, то регістри приймача та передавача є 16-розрядними. Якщо FO=1, то регістри є 8-розрядними. Біт FO скидається в нуль при подачі сигналу скидання.

У табл. 3. 23 наведено перелік команд переходів та виклику підпрограм. Ці команди забезпечують виконання програм з розгалуженою та циклічною структурами, там де на відміну від лінійних програм, необхідно виконувати не наступну команду, а команду, що знаходиться в іншій області пам'яті.

Таблиця 3. 23 – Команди переходів та виклику підпрограм

Команда	Опис
B	Безумовний перехід
BACC	Перехід за адресою, що задана акумулятором
BBNZ	Перехід, якщо (TC)=1
BBZ	Перехід, якщо (TC)=0
BC	Перехід, якщо біт C=1
BNC	Перехід, якщо біт C=0
BGEZ	Перехід, якщо (ACC)>=0
BGZ	Перехід, якщо (ACC)>0
BLEZ	Перехід, якщо (ACC)<=0
BLZ	Перехід, якщо (ACC)<0
BNZ	Перехід, якщо (ACC)≠0
BZ	Перехід, якщо (ACC)=0
BNV	Перехід, якщо біт OV=0
BV	Перехід, якщо біт OV=1
BIOZ	Перехід, якщо $\overline{(BIO)}$ =0
CALA	Непрямий виклик підпрограми
CALL	Виклик підпрограми
RET	Повернення з підпрограми

Для цього у лічильник команд завантажується адреса нової комірки пам'яті програм, яка називається адресою переходу. Команди, що реалізують зазначену операцію, називаються командами передачі управління. Більшість команд даної групи займають у пам'яті два слова.

Команда B (branch unconditionally) здійснює безумовний перехід.

Формат команди:

B < pma>, [<ка> [,<NARP>]]

Виконується операція: pma → PC. Поточний допоміжний регістр ARn та покажчик ARB модифікуються так, як вказано у команді. Управління передається команді, що знаходиться за адресою pma. Значення цієї адреси лежить у діапазоні 0..65535. Команда займає у пам'яті два слова.

Команда BACC (branch on address specified be accumulator) здійснює перехід за адресою, що задається акумулятором.

Формат команди:

BACC

Виконується операція (ACC(15-0)) → PC. Управління програмою передається команді, адреса якої визначається молодшим словом акумулятора. Команда займає у пам'яті одне слово.

Команда BANZ (branch on auxiliary register not zero) здійснює перехід, якщо значення допоміжного регістру не дорівнює нулю.

Формат команди:

BANZ < pma>, [<ка> [,<NARP>]]

Виконується операція if (AR(ARP)) ≠ 0 then pma → PC else (PC) + 2 → PC. Якщо вміст поточного допоміжного регістру не дорівнює нулю, то управління передається команді, яка знаходиться за адресою pma. В іншому випадку виконується наступна команда. Якщо не задано поле <ка>, вміст поточного допоміжного регістру зменшується на одиницю, тобто виконується інструкція *-. Команду BANZ можна використовувати для програмування циклічних алгоритмів. У цьому випадку допоміжний регістр виступає як лічильник циклів.

Приклад:

(AR(ARP))=1h	BANZ 40h, *-	(AR(ARP))=0
(PC)=58h		(PC)=40h

Команди умовного переходу, що наведені у табл. 3. 23 (BBZ, BC,…, BIOZ), мають загальний формат < pma>, [<ка> [,<NARP>]]. Вони виконують передачу управління команді, яка знаходиться за адресою pma, якщо виконується відповідна умова. В іншому випадку управління передається наступній команді. Потім модифікується вміст допоміжного регістру та покажчика ARP, якщо задані поля <ка> та <NARP>. Мнемоніка команд та відповідні умови зазначені у табл.. 3. 23.

Команда CALA (call subroutine indirect) здійснює непрямий виклик підпрограми.

Формат команди: CALA

Виконуються операції (PC) + 2 → TOS; pma → PC. Лічильник команд збільшується на 2 та завантажується у вершину стека. Управління передається команді, що знаходиться за адресою pma.

Команда RET (return from subroutine) здійснює повернення з підпрограми.

Формат команди: RET

Виконується операція (TOS) → PC. Вміст вершини стека копіюється у лічильник команд. Із стека виштовхується одне значення.

Перелік команд управління наведено в табл.. 3. 24.

Таблиця 3. 24 – Команди управління

Команда	Опис
B1T	Біт контролю
B1TT	Біт контролю, визначений Т-регістром
CNFD	Визначити блок B0 як пам'ять даних: 0 → CNF
CNFP	Визначити блок B0 як пам'ять програм: 1 → CNF
DINT	Блокування переривань
EINT	Дозвіл переривань
IDLE	Очікування переривань
LST	Завантаження регістру стану ST0
LST1	Завантаження регістру стану ST1
NOP	Немає операції
POP	Виштовхнути вершину стека в молодше слово акумулятора
POPD	Виштовхнути вершину стека в елемент пам'яті даних
PUSHD	Виштовхнути елемент пам'яті даних у вершину стека
PUSH	Виштовхнути молодше слово акумулятора у вершину стека
RC	Скинути біт переносу: 0 → C
RHM	Скинути режим захоплення: 0 → HM
ROVM	Скинути режим переповнення: 0 → OVM
RPT	Повторити команду стільки разів, скільки вказано в елементі пам'яті даних
RPTK	Повторити команду стільки разів, скільки вказано в безпосередній константі
RSXM	Скинути режим розширення знака: 0 → SXM
RTC	Скинути прапор контроль/управління: 0 → TC
SC	Встановити біт переносу: 1 → C
SHM	Встановити режим захоплення: 1 → HM
SOVM	Встановити режим переповнення: 1 → OVM
SST	Збереження регістру стану ST0
SST1	Збереження регістру стану ST1
SSXM	Встановити режим розширення знака: 1 → SXM
STC	Встановити прапор контроль/управління: 1 → TC
TRAP	Програмне переривання

До цієї групи команд входять такі команди: управління стеком; управління кількістю повторів циклів; управління перериваннями; завантаження та збереження регістрів стану; управління окремими бітами (C, OVM, HM, TC).

Для управління стеком використовуються команди: POP, POPD, PUSH, PUSHD, TRAP

Команда POP (pop top of stack to low accumulator) здійснює виштовхування вершини стека у молодше слово акумулятора.

Формат команди: POP

Виконується операція (TOS) → ACC(15-0). Вміст вершини стека (TOS) пересилається (виштовхується) у молодше слово акумулятора. На місце, що звільнюється, пересилаються елементи з нижніх рівнів стека. Після семи пересилань всі елементи стека будуть мати те саме значення, що відповідає останньому елементу стека. Процесор на має засобів, що виявляють антипереповнення стека.

Команда POPD (pop stack to data memory) здійснює виштовхування стеку у пам'ять даних.

Формат команди:

POPD <ac>

POPD <ка> [,<NARP>]

Виконується операція (TOS) → dma. Вміст вершини стека (TOS) пересилається (виштовхується) за адресою dma. Із стека виштовхується один елемент.

Команда PUSH (push low accumulator into stack) здійснює виштовхування молодшого слова акумулятора у стек.

Формат команди: PUSH

Виконується операція (ACC(15-0)) → TOS. Елементи стека послідовно пересилаються на рівень нижче. Вміст молодшого слова акумулятора пересилається до вершини стека, що звільняється.

Команда PUSHD (pop data memory value to stack) здійснює виштовхування значення пам'яті даних у стек.

Формат команди:

PUSHD <ac>

PUSHD <ка> [,<NARP>]

Виконується операція (dma) → TOS. Вміст пам'яті даних (dma) переміщується у вершину стека.

Команда TRAP (software interrupt) здійснює програмне переривання.

Виконуються операції (PC)+1 → TOS; 30 → PC. Команда передає управління команді, що знаходиться у комірці 30. У стек вштовхується адреса команди, що йде за командою TRAP. Це дозволяє повертатися у майбутньому за командою RET у точку переривання.

До команд управління циклами належать команди RPT, RPTK.

Команда RPT (repeat instruction as specified by data memory value) здійснює повторення відповідно до значення пам'яті даних.

Формат команди:

RPT <dma>

RPT <ка> [,<NARP>]

Виконується операція (dma(7-0)) → RPTC. Команда дозволяє занести у лічильник числа повторень RPTC значення молодших восьми бітів з пам'яті даних з адресою dma. При цьому команда, що йде за командою RPT, буде повторюватися на один раз більше, ніж кількість повторень, що задана безпосередньо числом, завантаженим у RPTC. Під час циклічного виконання команди, що йде за командою RPT, переривання будуть замасковані. Лічильник числа повторень RPTC скидається сигналом RC.

Приклад:

(AR(ARP))=1030 RPT * (RPTC)=Ah

(1030)=Ah

(RPTC)=0h

Команда RPTK (repeat instruction as specified by data immediate value) здійснює повторення відповідно до безпосередньо заданих значень.

Формат команди:

RPT K <константа>

Виконується операція 8-бітна константа → RPTC. Команда вміщує безпосередньо задану константу у лічильник числа повторень. У всьому іншому команда подібна до команди RPT.

До команд управління перериваннями входять команди: DINT, EINT, IDLE.

Команда DINT (disable interrupt) забороняє переривання.

Формат команди:

DINT

Виконується операція 1 → INTM. Команда встановлює біт режиму переривань INTM в одиницю. Після цього переривання, що маскуються, будуть заборонені. Переривання, що не маскуються, цією командою не забороняються.

Переривання також забороняються при надходженні сигналу скидання.

Команда EINT (enable interrupt) дозволяє переривання.

Формат команди:

EINT

Виконується операція 0 → INTM. Команда встановлює біт режиму переривань INTM в нуль. Після цього переривання, що маскуються, будуть дозволені.

Команда IDLE (idle until interrupt) дозволяє очікування переривання.

Формат команди:

IDLE

Виконується операція 0 → INTM. Виконання команди IDLE приводить до очікування переривання або сигналу скидання. Процесор переводиться у режим малого споживання енергії.

До групи команд завантаження та збереження регістрів станів входять команди LST, LST1, SST, SST1.

Команда LST (load status register ST0) завантажує регістр станів ST0.

Формат команди:

LST <ас>

LST <ка> [,<NARP>]

Виконується операція (dma) → ST0. Регістр станів ST0 завантажується вмістом пам'яті dma. Команда не діє на біт INTM та не здійснює завантаження ARB. Команда LST використовується для відновлення слова стану ST0 після переривань та виклику підпрограм. Виконання команди здійснює встановлення значень ARP, OV, OVM, DP. Якщо в команді задано поле NARP, воно ігнорується.

Команда LST1 (load status register ST1) завантажує регістр станів ST1.

Формат команди:

LST1 <ас>

LST1 <ка> [,<NARP>]

Виконується операція (dma) → ST1. Регістр станів ST1 завантажується вмістом пам'яті dma. Біти пам'яті даних, які завантажуються в поле ARB, також завантажуються у покажчик ARP. Якщо в команді задано поле NARP, воно ігнорується.

Команда SST (store status register ST0) зберігає регістр станів ST0.

Формат команди:

SST <ас>

SST <ка> [,<NARP>]

Виконується операція (ST0) → dma. Команда зберігає вміст регістру ST0 у пам'яті даних за адресою dma. У випадку прямої адресації регістр ST0 зберігається у нульовій сторінці незалежно від значення DP.

Команда SST1 (store status register ST1) зберігає регістр станів ST1.

Формат команди:

SST1 <ас>

SST1 <ка> [,<NARP>]

Виконується операція: (ST1) → dma. Команда аналогічна команді SST.

Команди управління окремими бітами наведено у табл. 3. 24. Окремо розглянемо 2 команди: BIT і BITT.

Команда BIT (test bit) встановлює біт контролю.

Формат команди:

BIT <ас> <двійковий код>

BIT <ка> <двійковий код> [,<NARP>]

Виконується така операція: заданий двійковим кодом біт (dma) → TC. Команда копіює заданий біт пам'яті даних у біт TC регістру станів ST1. Номер біта, який потрібно копіювати, задається полем <двійковий код>. Можливі значення цього поля 0..15.

Команда BITT (test bit specified by T register) встановлює біт контролю, що визначається Т-регістром.

Формат команди:

BITT <dma>

BITT <ка> [,<NARP>]

Виконується така операція: заданий за допомогою Treg(3-0) біт (dma) → TC. Команда аналогічна команді BIT, але номер біта, що копіюється у TC, задається чотирма молодшими бітами Т-регістру.

Для решти команд цієї групи, що управляють бітами CNF, C, HM, OVM, TC, SXM, відповідні операції мають простий формат, якій містить тільки мнемоніку команди. Операції, які здійснюються наведено у табл. 3. 24. Команди є парними – скинути/встановити.

3.4.3. Особливості програмування. Директиви асемблеру. Емуляція

Програмування на мові асемблеру передбачає запис всіх елементів програми у символічній формі. Перетворення символічних імен у машинні коди покладається на спеціальну програму, яка називається асемблером.

Програми на мові асемблеру записуються у вигляді послідовності команд, які також називаються операторами. У багатьох випадках один оператор асемблеру породжує одну машинну команду. Тому програмування на асемблері є зручною формою запису машинних кодів і дозволяє досягти максимальної ефективності програм.

Програма-асемблер створює об’єктний файл. Розрізняють два різних формати об’єктниих файлів: COFF (common object file format) формат та TI-формат. У подальшому розглядається асемблер, що підтримує COFF-формат.

Команда асемблеру містить такі поля:

[<мітка>] <мнемоніка> <операнди> [<коментар>]

Необов’язкове поле мітки – це символічне найменування 16-розрядної адреси тієї комірки пам'яті, у якій буде розміщена помічена команда. Символічне найменування (ім’я, ідентифікатор) повинне починатися з літери

і містити не більш 6 символів. Мітки використовуються як адреси переходів у командах та директивах передачі управління. Приклад використання мітки:

```
Мітка    Мнемоніка    Операнди    Коментар
LOOP     LACK         0FFFh
         …
         BGZ          LOOP
         …
         BLEZ         LOOP
```

Команди BGZ та BLEZ передають управління за однією адресою пам'яті програм, що містить команду LACK.

Поле мнемоніки містить символічне ім’я команди. Мнемоніки команд зазвичай є абревіатурами речень, що характеризують основні функції, які виконує команда. Довжина мнемоніки не перевищує 5 символів. Поле мнемоніки відокремлюється від поля мітки хоча б одним пропуском. Мнемоніки – це ключові слова, тому якщо вони записані з помилками, асемблер формує відповідне повідомлення.

Вміст поля операндів залежить від команди. Як операнди можуть виступати абсолютні та символічні адреси пам'яті, програмно доступні внутрішні регістри процесора (PRD, TIM, AR0 та ін.), адреси портів вводу/виводу (PA1-PA15), числові та символічні константи, режими непрямої адресації (*+, +0-, *BR0 та ін.). Поля операндів деяких команд можуть бути пустими (SST, CALA, RTC та ін.).

Якщо в полі операнда міститься шістнадцяткове число, воно повинне починатися з цифри та закінчуватися літерою h (hex).

Число, що починається з літери, доповнюється зліва незначущим нулем, наприклад, 0AFCh, 0FCEDh. У TI-форматі запис шістнадцяткового числа починається з символу «>», наприклад, >FF00, >A7. У іншому випадку число вважається десятковим. У полі операндів дозволяється вказувати символічні імена, які визначені у самому асемблері та які пов’язані з архітектурою процесора. У мову асемблеру вбудовані імена програмно доступних регістрів: AR0-AR7, TIM, PRD, IMR, GREG, DRR, DXR; мітки: DATn, PRGn, що асоціюються відповідно з *n*-ю коміркою пам'яті даних та пам'яті програм.

Замість імен внутрішніх регістрів дозволяється зазначати їх адреси у десятковій або шістнадцятковій системі. Наприклад, наведені нижче команди ідентичні:

```
SALK 3h      ;(ACC) → 3h
SALK PRD     ;(ACC) → PRD
```

У командах передачі управління у полі операнда можна зазначати мітки, які введені у поля міток інших команд. Мітки, розташовані у полі операнду, є символічними адресами переходу. У процесі обробки та

завантаження програми міткам ставляться у відповідність адреси. Мітки, що введені як операнди, повинні обов'язково один раз з'явитися у полі мітки деякої команди.

Приклад:

```
        LAC    0200h          ;
        SUBK   01h            ;
        BGZ    MITKA1         ; Перехід на мітку
        …
MITKA1  LAC    0300h          ;
        CALL   SUBR           ; Виклик підпрограми
        …
SUBR    SST
        SST1
```

У рядках асемблерної програми можуть використовуватися директиви, що передають вказівки програмі асемблеру про виконання визначених дій. Директиви визначають порядок дій, розміщують у пам'яті команди та дані, присвоюють чисельні значення символічним найменуванням, резервують пам'ять та ін. Нижче розглядаються спрощені формати деяких директив COFF-асемблеру.

Директива .title – директива заголовка програми. Формат директиви:

.title <рядок>

Поле <рядок> – послідовність символів (не більш 65), що взяті у подвійні лапки. Наприклад:

.title "Програма фільтрації"

Директива .set (встановити) виконує присвоєння. Формат директиви:

< ім'я > .set <вираз>

При виконанні директиви імені, що стоїть у полі мітки, присвоюється значення виразу. Зазвичай вираз задається шістнадцятковим числом. Коли ім'я зустрічається у полі операнда команди, замість нього підставляється присвоєне значення:

```
SADR   .set    00400h
EADR   .set    02000h
…
LRLK   AR0,EADR
LRLK   AR1,SADR
```

У команді LRLK замість імен SADR та EADR будуть використані значення 400h та 2000h відповідно.

Директива .bss резервує пам'ять під змінну та має такий формат:

.bss < ім'я > <константа>

Числова константа визначає кількість комірок пам'яті, що резервуються для зберігання змінної. Наприклад, запис .bss var,1 резервує одну комірку пам'яті під змінну var.

Директива .sect (секція, сегмент) визначає іменований сегмент та має формат:

.sect <рядок>

Програма може бути розбита на сегменти, які являють собою блоки коду або даних, що переміщуються. Сегменти розміщуються у пам'яті процесора. Конкретне розміщення сегментів у адресному просторі пам'яті визначається на стадії редагування зв'язків. Поле <рядок> відповідає назві сегмента. Наприклад,

```
.sect "reset"
B MAIN
```

Тут сегмент "reset" містить одну команду передачі управління на мітку MAIN, що відповідає початку програми. Абсолютна адреса, за якою буде розміщена команда, визначається при виклику редактора зв'язків.

Окрім іменованих сегментів, об'єктні файли формату COFF завжди містять три стандартних сегменти: .text, .data, .bss.

Директива .text визначає сегмент, у якому розташовується код програми. Код програми буде вміщуватися у даний сегмент, доки не буде відкритий інший сегмент за допомогою директив .sect, .data.

Директива .data відкриває сегмент, у якому зазвичай розміщуються таблиці вхідних даних та інші ініціалізовані змінні.

Існує два основних типи сегментів: ініціалізовані та неініціалізовані. Ініціалізовані сегменти містять дані або код. Сегменти .sect, .text, .data належать до ініціалізованих. Неініціалізовані сегменти резервують пам'ять у адресному просторі процесора для неініціалізованих даних. Наприклад, сегмент .bss.

Нижче наведено текст програми, що містить директиви .set, .bss, .sect, .data. Для кращого розуміння призначення директив програма містить додаткове поле адреси та поле машинного коду.

```
Адреса  Код
        0400    SADR    .set    0040h
        2000    EADR    .set    02000h
                *
0000                    .bss    VAR,1
                *
0000                    .sect   "RESET"
0000    FF80            B       MAIN
0001    000'
```

```
                *
0000                        .text
0000    C800    MAIN    LDRK    0               ; DP ← 0
0001    CA00            LACK    0               ; ACC ← 0
                …
0005    D000            LRLK    AR0, EADR       ; AR0 ← EADR
0006    2000
0007    D100            LRLK    AR1, SADR       ; AR1 ←  SADR
0008    0400
                …
0012    6000            SACL    VAR             ; (ACC) ← VAR
                …
                        .end
```

Друге слово машинного коду команди B містить відносну адресу переходу 000’ на мітку MAIN, яка може бути визначена тільки тоді, коли буде відомо, де розташовується сегмент .text. Абсолютна адреса розташування секцій задається користувачем на етапі редагування зв’язків.

Директива .bss також формує відносні адреси. Абсолютні адреси визначаються користувачем при редагуванні зв’язків.

Директива .end інформує програму-асемблер про досягнення фізичного кінця вхідної програми.

Однією з основних проблем програмування на мові асемблеру є те, що весь процес обчислень має бути сформульований у термінології команд процесора. Але команди процесора виконують дрібні операції у порівнянні з операціями, що потребуються для розв’язання задачі на змістовному рівні, тому програмування на асемблері досить трудомістке.

Для зменшення труднощів, пов’язаних з цим, програміст може визначати більш змістовні команди, які потім використовуються при складанні програм сумісно із звичайними командами. Такі команди називають макрокомандами. Макрокоманда вводиться за допомогою макровизначення, яке по суті близьке до визначення процедур у мовах програмування високого рівня. Формат макровизначення:

<ім’я макрокоманди> $MACRO [параметр{,параметр}]
тіло макровизначення
$ENDM

Директива $MACRO відкриває визначення макрокоманди, присвоює їй ім’я, що стоїть зліва від слова $MACRO та задає параметри макрокоманди, які не є обов’язковими. Макрокоманда викликається за допомогою свого імені та операндів. Коли макрокоманда викликана, асемблер ставить у відповідність перший операнд у виклику макрокоманди з першим операндом у визначенні макрокоманди і т.п. Директива $ENDM закінчує визначення

макрокоманди. При виклику макрокоманди асемблер підставляє команди, що входять у її визначення, до коду програми.

Контрольні питання

1. Режими адресації та формати команд.
2. Класифікація команд ЦПОС.
3. Перелік та призначення основних арифметичних та логічних команд роботи з акумулятором. Особливості їх використання.
4. Перелік та призначення команд завантаження та збереження вмісту акумулятора. Особливості їх використання.
5. Перелік та призначення команд додаткових (допоміжних) регістрів. Особливості їх використання.
6. Перелік та призначення команд Т- та Р-регістрів. Особливості їх використання.
7. Перелік та призначення команд множення. Особливості їх використання.
8. Перелік та призначення команд вводу/виводу та пересилання даних. Особливості їх використання.
9. Перелік та призначення команд переходів та виклику підпрограм. Особливості їх використання.
10. Перелік та призначення команд управління. Особливості їх використання.
11. Формат команд асемблеру. Призначення полів команди.
12. Перелік та призначення основних директив асемблеру.
13. Призначення макрокоманд. Формат макровизначення.

4. АРХІТЕКТУРА ТА МЕТОДИ СИНТЕЗУ ПЛІС ПРИСТРОЇВ

4.1 Технології програмування апаратних засобів і області їх застосування

Інтегральні мікросхеми з логікою, що програмується (ПЛІС), або FPGA (field programmable gate arrays) є цифровими інтегральними мікросхемами (ІС), що складаються з програмованих логічних блоків і програмованих з'єднань між цими блоками. Можливість конфігурувати ці пристрої дозволяє інженерам-розробникам розв'язувати безліч різних задач.

Залежно від способу виготовлення, ПЛІС можуть програмуватися або один раз, або багатократно. Пристрої, які можуть програмуватися тільки один раз, називаються одноразово програмованими.

Словосполучення «field programmable», що міститься в розшифровці абревіатури FPGA, означає, що програмування FPGA-пристроїв виконується в конкретних («польових») умовах залежно від задачі, що розв'язується, (на відміну від пристроїв, внутрішня функціональність яких жорстко задана виробником). Воно може також означати, що FPGA-пристрої (ПЛІС) конфігуруються в лабораторних умовах, або свідчити про те, що йдеться про можливість модифікації функцій пристрою, вбудованого в електронну систему, яка вже використовується. Якщо пристрій може бути запрограмований, залишаючись у складі системи вищого рівня, він називається внутрішньосистемно програмованим [24].

Існує багата кількість різних типів цифрових мікросхем, у тому числі і такі, як «розсипна логіка» (невеликі компоненти, що містять декілька простих фіксованих логічних функцій), пристрої пам'яті і мікропроцесори. В даному випадку інтерес викликають програмовані логічні пристрої (ПЛП), спеціалізовані замовні інтегральні мікросхеми ASIC (application specific integrated circuit), спеціалізована інтегральна схема і ASSP (application specific standard parts) та ПЛІС. Причому термін ПЛП об'єднує два типи пристроїв: прості програмовані логічні пристрої (прості ПЛП) і складні програмовані логічні пристрої (складні ПЛП).

У українській технічній літературі ASIC прийнято називати «замовними інтегральними схемами (мікросхемами)», хоча сьогодні часто ASIC називають «спеціалізованими інтегральними схемами». Часто робиться уточнення, і мікросхеми поділяють на «замовні» і «напівзамовні».

Внутрішня архітектура ПЛП визначена виробником таким чином, що вони можуть бути конфігуровані (перепрограмовані) «на місці» для виконання самих різних функцій. На відміну від ПЛІС ці пристрої містять меншу кількість вентилів і використовуються для розв'язання невеликих і досить простих задач.

Разом з тим, існують замовні інтегральні схеми ASIC і ASSP, які містять сотні мільйонів логічних вентилів і можуть виконувати неймовірно великі і складні функцій. У основі ASIC і ASSP лежать ті самі конструкторські рішення, і у них та сама технологія виробництва. Обидва типи пристроїв розробляються для використання у складі спеціальних додатків, але при цьому ASIC розробляються і виробляються за замовленням спеціалізованих компаній, а ASSP призначаються масовому користувачу.

Не дивлячись на те що пропоновані користувачу замовні мікросхеми відрізняються високим ступенем інтеграції, рівнем складності задач, що розв'язуються, і продуктивністю, їх розробка і виробництво є досить тривалим і дорогим процесом. До того ж остаточний варіант схеми «заморожується в кремнії», і для її модифікації потрібне створення нової версії.

Таким чином, ПЛІС займають проміжне положення між ПЛП і замовними інтегральними схемами. З одного боку, їх функціональність може бути задана безпосередньо на місці відповідно до вимог замовника-користувача. З іншого боку, вони можуть містити мільйони логічних вентилів і, отже, реалізовувати надзвичайно великі і складні функції, які спочатку могли бути реалізовані тільки за допомогою замовних інтегральних схем.

Що стосується вартості ПЛІС, то вона набагато нижча за вартість замовних інтегральних схем (хоча остаточна версія замовної мікросхеми при масовому виробництві виявляється дешевшою). До того ж, у разі використання ПЛІС, внесення змін у цей пристрій не викликає особливих складнощів і навіть істотно скорочує терміни його виходу на ринок. Все це робить ПЛІС привабливими не тільки для крупних розробників, але і для невеликих новаторських конструкторських бюро. Іншими словами, «апаратні» або «програмні» ідеї окремих інженерів або невеликих груп інженерів можуть бути реалізовані у вигляді випробувальних стендів на основі ПЛІС без великих одноразових витрат на проектування або закупівлю дорогого оснащення, необхідного для розробки замовних мікросхем.

В даний час ПЛІС заповнюють п'ять крупних сегментів ринку [24]:

• замовні інтегральні схеми. Сучасні ПЛІС використовуються для створення пристроїв такого рівня, який до цього могли забезпечити тільки замовні мікросхеми;

• системи цифрової обробки сигналів. Високошвидкісна цифрова обробка сигналів традиційно здійснювалася за допомогою спеціально розроблених мікропроцесорів, так званих цифрових сигнальних процесорів (ЦСП) або digital signal processors (DSP). Проте сучасні ПЛІС містять вбудовані помножувачі, схеми арифметичного перенесення і великий обсяг оперативної пам'яті усередині кристала. Все це в поєднанні з високим

ступенем паралелізму ПЛІС забезпечує їх перевагу над найшвидшими сигнальними процесорами в 500 і більше разів;

• системи на основі вбудованих мікроконтролерів. Нескладні задачі управління звичайно виконуються вбудованими процесорами спеціального призначення, які називаються мікроконтролерами. Ці недорогі пристрої містять вбудовану програму, пам'ять команд, таймери, інтерфейси вводу/виводу, що розташовані поряд з ядром на одному кристалі. Найпростіші ПЛІС можна використовувати для реалізації програмного мікропроцесорного ядра з необхідними функціями вводу/виводу. В результаті ПЛІС стають все більш привабливими пристроями для реалізації функцій мікроконтролерів;

• мікросхеми, що забезпечують фізичний рівень передачі даних. ПЛІС вже давно використовуються як зв'язуюча логіка, що виконує функцію інтерфейсу між мікросхемами, які реалізують фізичний рівень передачі даних, і вищими рівнями мережних протоколів. Той факт, що сучасні ПЛІС можуть містити велику кількість високошвидкісних передавачів, означає, що мережні і комунікаційні функції можуть бути реалізовані в одному пристрої;

• системи з архітектурою, що перебудовується, або reconfigurable computing (RC). Можна застосовувати «апаратне прискорення» програмних алгоритмів, використовуючи такі властивості ПЛІС, як паралелізм та можливість перенастроювання. В даний час різні компанії зайняті створенням величезних перенастроюваних обчислювальних машин на основі ПЛІС. Такі системи можна використовувати для виконання широкого спектру задач — від моделювання апаратури до криптографічного аналізу або створення нових ліків.

Для програмування необхідний механізм, що дозволяє конфігурувати (програмувати) підготовлений кремнієвий кристал.

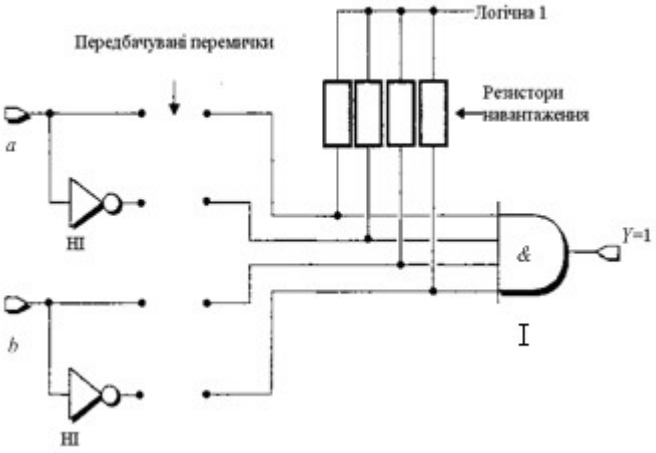


Рис. 4.1. Проста функція, що програмується

На рис. 4.1 наведено приклад простої функції, що програмується, з двома входами (*a* та *b*) та одним виходом. На входах присутні інвертори (операція НІ), а отже, кожен вхід може бути поданий або в прямій, або в інвертованій формі.

Особливої уваги заслуговує розташування передбачуваних перемичок. За відсутності перемичок на всі входи логічного елемента І через резистори навантаження буде подана логічна 1. Це означає, що вихід у даного пристрою завжди знаходитиметься в стані логічної 1. Щоб зробити просту програмовану функцію, необхідний якийсь механізм, що дозволив би встановлювати одну або декілька перемичок.

Одним з перших методів, який надав можливість користувачам програмувати власні пристрої, був і дотепер існує так званий метод плавких перемичок. При використанні цього методу пристрій виготовляється зі всіма можливими з'єднаннями, кожне з яких є плавкою перемичкою (рис. 4.2).

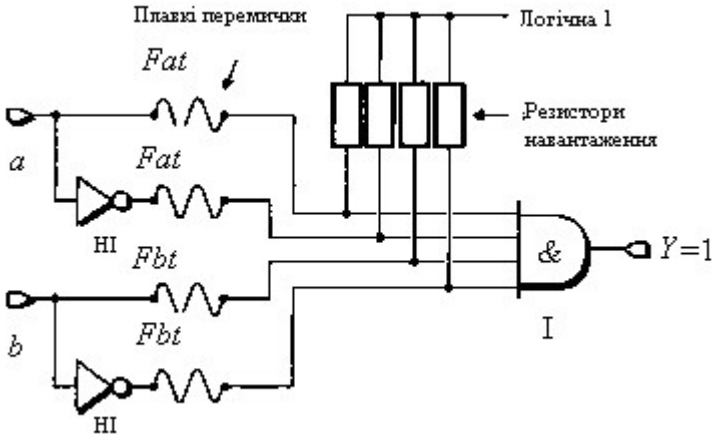


Рис. 4.2. Заповнення пристрою плавкими перемичками

Принцип дії цих перемичок аналогічний принципу дії запобіжників, які використовуються в побутовій техніці, наприклад в телевізорах. Якщо через непередбачені обставини телевізор починає споживати дуже велику потужність, це призведе до перегорання запобіжника. У результаті відбудеться розрив ланцюга (розрив провідника), який захистить пристрій від пошкодження.

Плавкі перемички мають мікроскопічні розміри, оскільки при їх виготовленні на кремнієвому кристалі використовуються ті ж технології, що і при виготовленні транзисторів і провідників.

Спочатку у програмованому пристрої на основі плавких перемичок всі перемички цілі. Це означає, що в незапрограмованому стані на виході функції, яка розглядається тут як приклад, завжди знаходитиметься рівень логічного 0. Наявність рівня логічного 0 хоча б на одному з входів вентиля І приведе до установлення 0 на виході. Так, якщо на вході *a* встановлений 0, та на виході вентиля І також буде 0. Аналогічно, якщо на вході *a* встановлена логічна 1, то вихід вентиля, який позначатимемо як *a*, буде встановлений в 0, і, отже, на виході вентиля І буде 0. Така ж ситуація має місце при використанні входу *b*.

Вся суть полягає у тому, що розробники можуть вибірково видаляти непотрібні плавкі перемички, подаючи на входи пристрою імпульси відносно високої напруги і великого струму. Розглянемо, наприклад, що відбудеться, якщо видалити плавкі перемички *Fat* і *Fbt* (рис. 4.3).

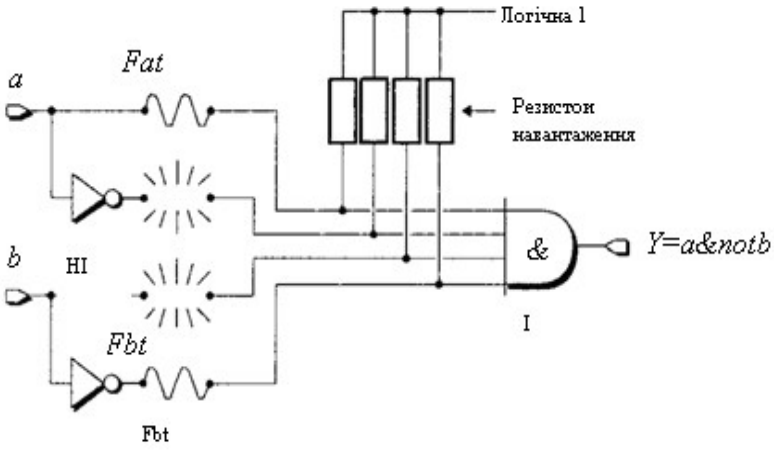


Рис. 4.3. Плавкі перемички після програмування

Видалення цих перемичок приведе до від'єднання інвертованого входу *a* і прямого входу *b* від логічного елемента І (через резистори навантаження на цих входах встановлюється значення логічної 1). Ця обставина вимушує пристрій формувати нову функцію *y* = *a* & not *b*.

Такий процес видалення плавких перемичок зазвичай називається процесом програмування пристрою, але може також називатися перепалом перемичок або пропалюванням пристрою.

Пристрої, конфігурацію яких засновано на методі плавких перемичок, є одноразово програмованими пристроями (OTP — one-time programmable), оскільки після перепалу плавка перемичка не може бути замінена або повернена в первинний стан.

Для різних ПЛІС використовуються різні методи програмування, але метод плавких перемичок до сучасних ПЛІС не застосовують. Він наведений як елементарний приклад.

Метод нарощуваних перемичок протилежний методу плавких перемичок. У цьому випадку кожне з'єднання, що конфігурується, має лінію зв'язку, що називають нарощуваною перемичкою. У незапрограмованому стані нарощувана перемичка має такий високий опір, що її можна розглядати як розімкнений ланцюг (розрив провідника, рис. 4. 4).

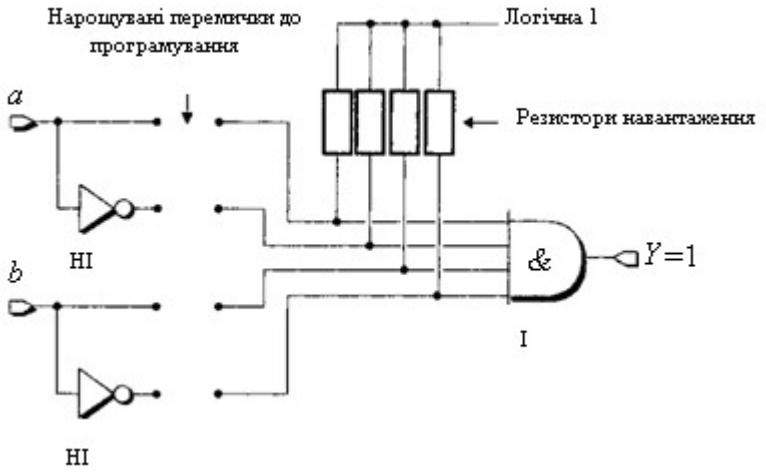


Рис. 4.4. Нарощувані перемички до програмування

Проте перемички можуть вибірково «нарощуватися» (програмуватися) за допомогою імпульсів відносно високої напруги і великого струму, що подаються на входи пристрою. Наприклад, якщо додати нарощувані перемички в ланцюг інвертованого входу *a* і прямого входу *b*, пристрій реалізує функцію *y* = not *a* & *b* (рис. 4.5).

Спочатку нарощувана перемичка являє собою мікроскопічний стовпчик аморфного (некристалічного) кремнію, що зв'язує два металеві провідники. У такому «незапрограмованому» стані аморфний кремній поводиться як діелектрик з опором, що досягає мільярда Ом (рис. 4.6, а).

Процес програмування окремого елемента полягає у вирощуванні зв'язку, що називається з'єднанням, і здійснюється шляхом перетворення аморфного кремнію-ізолятора в полікристалічний кремній, що проводить електричний струм. (рис. 4.6, *б*).

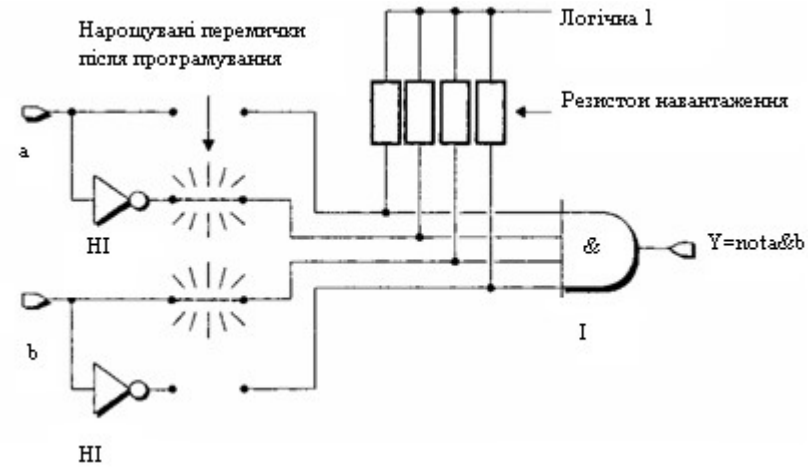


Рис. 4.5. Нарощувані перемички після програмування

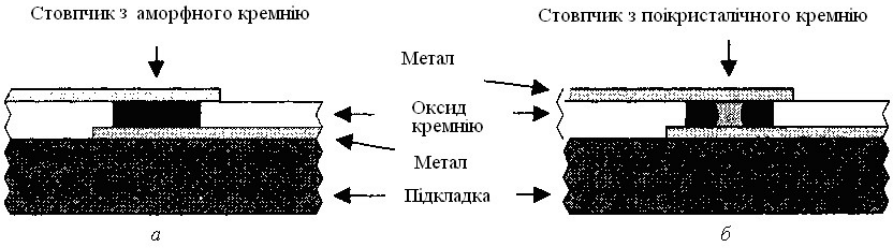


Рис. 4.6. Вирощування перемичок:
а – до програмування; б – після програмування

Пристрої, конфігурація яких здійснена методом нарощуваних перемичок, є однократно програмованими тому, що перемичка не може бути зруйнована або повернена до початкового стану і «перепрограмована».

Електронні системи загалом і комп'ютери зокрема використовують два типи запам'ятовуючих пристроїв: постійний запам'ятовуючий пристрій (ПЗП) і оперативний запам'ятовуючий пристрій (ОЗП).

ПЗП називають незалежними пристроями, оскільки інформація, що зберігається в них, не руйнується при відключенні живлення системи. Інші компоненти системи можуть зчитувати інформацію з пристроїв ПЗП, але не можуть записувати в них нові дані. На відміну від ПЗП, в ОЗП дані можуть як записуватися, так і зчитуватися. Такі пристрої є енергозалежними, оскільки при відключенні живлення вся інформація, що зберігається в ОЗП, буде втрачена.

Типові мікросхеми ПЗП також називаються програмованими фотошаблоном, або масочно-програмованими, оскільки будь-які дані, що містяться в них, жорстко прошиваються у процесі виробництва за допомогою фотошаблона. Фотошаблони використовуються для створення транзисторів і металевих провідників, що з'єднують їх, які також називаються шарами металізації.

Розглянемо, наприклад, транзисторну комірку ПЗП, яка може зберігати один біт даних (рис. 4.7). Стандартний пристрій ПЗП складається з деякої кількості рядків (адреси) і стовпців (дані), які разом утворюють масив даних. До кожного стовпця підключений один резистор навантаження, який дозволяє підтримувати на виході стовпця рівень логічної 1, а в кожному перетині рядка і стовпця присутній транзистор і, при необхідності, перемичка. Наявність/відсутність перемички задається фотошаблоном.

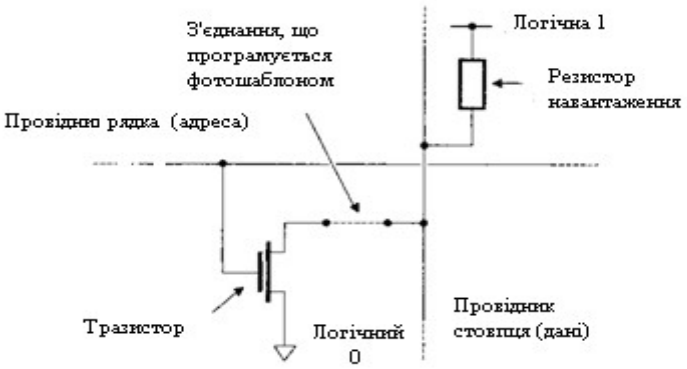


Рис. 4.7. Транзисторна комірка ПЗП, що програмується фотошаблоном

Більшість мікросхем ПЗП спочатку конфігурують для масового користувача. Коли необхідно конфігурувати пристрій відповідно до особливих вимог використовується індивідуально виготовлений фотошаблон, за допомогою якого визначаються комірки з перемичками і комірки без перемичок.

При подачі активного сигналу на провідник рядків цей рядок намагається перевести всі підключені до нього транзистори у відкритий стан. У разі, коли комірка містить запрограмоване за фотошаблоном з'єднання, транзистор цієї комірки, що активується, з'єднує провідник стовпця з рівнем логічного 0, встановлюючи таким чином значення виходу в 0. І навпаки, якщо в комірці немає запрограмованого з'єднання, транзистор не

виконуватиме будь-якої дії і через резистор навантаження, приєднаний до стовпця, підтримуватиме на виході мікросхеми рівень логічної 1.

Недолік масочно-програмованих пристроїв полягає у тому, що їх виробництво є досить дорогим задоволенням, якщо тільки не передбачається їх виробництво в дуже великих кількостях. Крім того, вони досить рідко використовуються при розробці апаратури, коли виникає необхідність часто змінювати склад компонентів.

Перші програмовані постійні запам'ятовуючи пристрої (ППЗП) були розроблені компанією Harris Semiconductor. Для створення цих пристроїв використовувався метод плавких перемичок з ніхрому.

Спрощену схему комірки ППЗП на основі транзистора з плавкою перемичкою наведено на рис. 4.8.

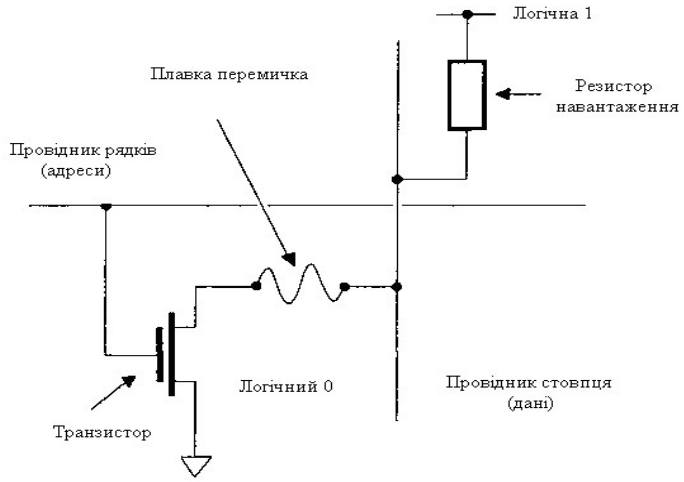


Рис. 4.8. Комірка ППЗП на основі транзистора з плавкою перемичкою

Мікросхема, що надходить від виробника, перебуває в незапрграмованому стані, але при цьому вона містить всі передбачені конструкцією плавкі перемички. В цьому випадку при переключенні рядка в активний стан включатимуться всі транзистори, під'єднані до цього рядка, вимушуючи всі стовпці знижувати рівень вихідної напруги до логічного 0 через відповідні їм транзистори. Інженер-розробник може на свій розсуд вибірково видаляти непотрібні плавкі перемички, подаючи на вхід пристрою імпульси відносно високої напруги і великого струму. При видаленні плавкої перемички на виході комірки формуватиметься рівень логічної 1.

Ці мікросхеми спочатку призначалися для використання як пристрої пам'яті, тобто для зберігання комп'ютерних програм і значень постійних

величин (звідси абревіатура ПЗП). Проте розробники знайшли їм корисне застосування при реалізації простих логічних функцій, таких як, таблиці відповідності і кінцеві автомати.

З часом з'явилися різні ПЛП загального призначення, засновані на методах пропалюваних і нарощуваних перемичок.

Пристрої на основі пропалюваних або нарощуваних перемичок можуть бути запрограмовані тільки одноразово. Тому внесення в систему яких-небудь змін після перепалу або нарощування перемички вимагає великих витрат часу. В деяких випадках можна модифікувати пристрій шляхом пропалення або нарощування ще не модифікованих початкових перемичок, але можливість скористатися цією властивістю випадає вкрай рідко. У зв'язку з цим виникла ідея створення таких пристроїв, які можна було б програмувати, стирати і знов програмувати, тобто перепрограмувати.

Одним з варіантів реалізації цієї ідеї став ППЗП із стиранням (СППЗП). Перший такий пристрій був розроблений в 1971 році компанією Intel, яка привласнила йому назву 1702.

СППЗП-транзистор має таку ж структуру, як стандартний МОП-транзистор але з додатковим (другим) плаваючим затвором з полікристалічного кремнію, ізольованого шарами оксиду кремнію (рис. 4.9).

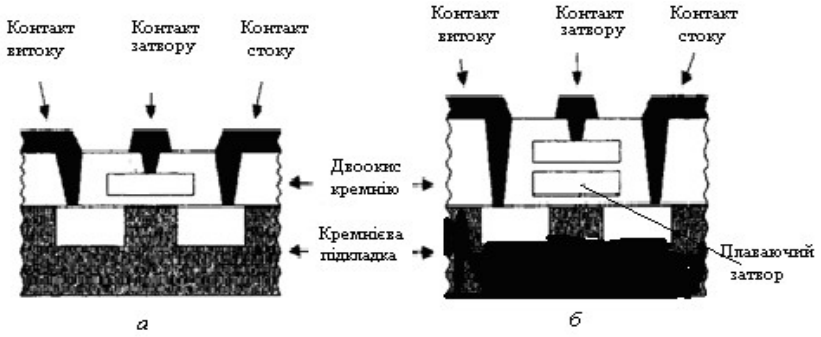


Рис. 4.9. Порівняння МОП та СППЗП транзисторів:
а – МОП-транзистор; б – СППЗП-транзистор

У незапрограмованому стані плаваючий затвор не заряджений і не впливає на роботу звичайного затвора. Щоб запрограмувати транзистор, необхідно прикласти до контактів затвора і стоку польового транзистора відносна високу напругу, близько 12 вольт. При цьому транзистор різко включається, і швидкі електрони долають шар оксиду кремнію, прямуючи в плаваючий затвор. Цей процес відомий як інжекція гарячих, або високо енергетичних, електронів. Після зняття сигналу програмування

негативно заряджені частинки залишаються в плаваючому затворі. Їх заряд стабільний і при дотриманні правил експлуатації вони не розсіюються впродовж більше 10 років. Накопичені на плаваючому затворі заряди блокують нормальну роботу звичайного затвора, і таким чином дозволяють розрізняти запрограмовані комірки від незапрограмованих. Завдяки цій властивості такі транзистори можна використовувати для формування елементів пам'яті (рис. 4.10).

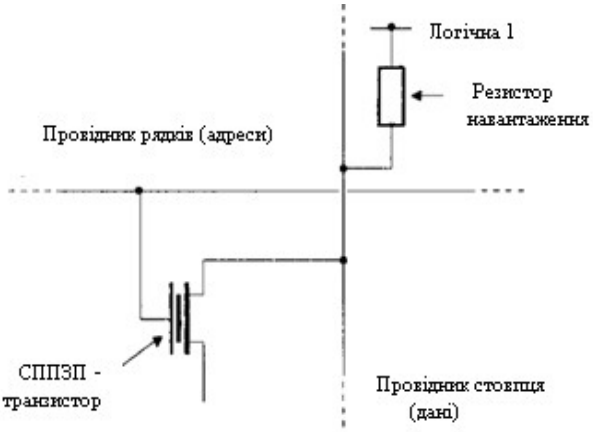


Рис. 4.10. Комірка пам'яті на основі СППЗП-транзистора

Такий елемент пам'яті більше не потребує плавких перемичок, нарощуваних перемичок або програмованих фотошаблоном з'єднань. У незапрограмованому стані, який забезпечується виробником, всі плаваючі затвори у СППЗП-транзисторах будуть не заряджені. В цьому випадку при переведенні провідника рядків в активний стан включатимуться всі транзистори, приєднані до цього рядка, примушуючи всі провідники стовпців скидати сигнал на виході до рівня логічного 0 через відповідні транзистори. Для програмування можуть використовуватися входи пристрою для заряду плаваючих затворів обраних транзисторів, тим самим блокуючи їх роботу. У цих випадках в елементах пам'яті зберігатимуться логічні 1.

З погляду розташування на кремнієвому кристалі елемента пам'яті СППЗП більш ефективні, оскільки їх розміри істотно менші, ніж розміри їх аналогів на плавких перемичках. Однак основною перевагою таких елементів пам'яті є можливість стирання та перепрограмування. Стирання елементів пам'яті є не що інше, як «витікання» електронів з плаваючого затвора. Енергія, необхідна для витікання електронів, забезпечується за допомогою

джерела ультрафіолетового (УФ) випромінювання. Мікросхеми СППЗП поставляються в керамічному або пластиковому корпусі, зверху якого є невелике кварцове вікно. Це вікно звичайно закрите шматочком непрозорої клейкої стрічки. Щоб стерти мікросхему, спочатку треба витягнути її з монтажної плати, відкрити кварцове вікно і потім помістити в контейнер, що закривається, з потужним джерелом УФ-випромінювання.

Головними недоліками СППЗП є їх дорогий корпус з кварцовим вікном і час, необхідний для їх очищення, який складає приблизно 20 хвилин. Проблема, яку доведеться вирішувати найближчим часом, пов'язана з удосконаленням технологічного процесу, який дозволить виконувати транзистори меншого розміру. Через те, що напівпровідникові структури стають меншими і їх густина, тобто кількість транзисторів і з'єднань, збільшується, великий відсоток поверхні кристала виявляється покритим металом. Це утрудняє поглинання УФ-випромінювання комірками СППЗП і збільшує час експозиції, що необхідний для стирання.

Дані мікросхеми спочатку призначалися для використання як програмовані постійні запам'ятовуючи пристрої, що знайшло віддзеркалення в їх назві — ППЗП. Пізніше цю технологію стали застосовувати в більш універсальних ПЛП, які одержали назву ПЛП із стиранням.

Наступний ступень технології являє собою ППЗП з електричним стиранням (ЕСППЗП). Комірка пам'яті ЕСППЗП наведена на рис. 4.11.

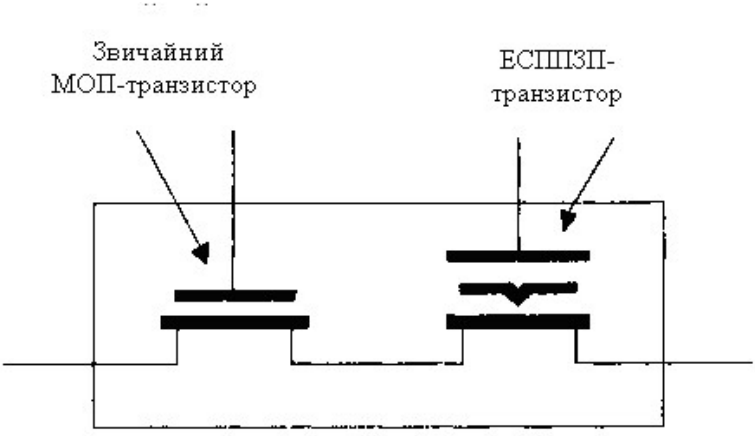


Рис. 4.11. Комірка пам'яті ЕСППЗП

Комірка ЕСППЗП приблизно в 2.5 рази більша, ніж еквівалентна їй комірка СППЗП, оскільки вона складається з двох віддалених один від одного транзисторів.

ЕСППЗП-транзистор має плаваючий затвор, але цей затвор оточений дуже тонким ізолюючим шаром оксиду кремнію. Другий транзистор може використовуватися для стирання елемента пам'яті електричним способом.

На початку свого існування ЕСППЗП-мікросхеми використовувалися як комп'ютерна пам'ять. Ця ж технологія була згодом застосована до пристроїв ПЛП, які стали називатися ПЛП з електричним стиранням (ЕСПЛП) або програмованими логічними інтегральними схемами (ПЛІС).

Технологія, відома як Flash (спалах, мить, миттєвість), починалась від технології виготовлення як СППЗП, так і ЕСППЗП. Спочатку назва Flash була присвоєна пристроям, виготовленим за цією технологією, щоб відобразити характерний для них надзвичайно малий час стирання в порівнянні з пристроями СППЗП. Компоненти, виконані за Flash-технологією, можуть бути реалізовані в різних типах архітектури. Одні пристрої можуть мати елементи пам'яті, виконані на одному транзисторі з плаваючим затвором такої ж площі, як в комірках СППЗП, але з набагато тоншими ізолюючими шарами оксиду кремнію. Такі пристрої можуть стиратися електричним способом, але при цьому тільки шляхом очищення всього пристрою або більшої його частини. Інші типи пристроїв побудовані на елементах пам'яті з двома транзисторами, подібно до мікросхемам ЕСППЗП, що дозволяє стирати або перепрограмувати інформацію послівно.

Перші версії Flash-пристроїв могли зберігати тільки один біт інформації на комірку. Але вже до початку 2002 року технологи виконали ряд експериментів по збільшенню місткості комірок. Виявилося, що, згідно з одним методом, запам'ятовування двох бітів в одній комірці можливе завдяки зберіганню різних рівнів зарядів в плаваючих затворах Flash-транзисторів, а згідно з іншим методом – завдяки створенню двох дискретних запам'ятовуючих вузлів в шарі під затвором.

Існують два основні типи напівпровідникових пристроїв оперативної пам'яті: динамічний ОЗП і статичний ОЗП. У разі динамічного ОЗП кожен елемент пам'яті формується за допомогою пари транзистор-конденсатор, яка займає мало місця на поверхні кремнієвого кристала. Визначення «динамічне» відображає той факт, що з часом конденсатор втрачає свій заряд, тобто для збереження даних кожна комірка повинна періодично перезаряджатися. Ця операція називається регенерацією. Вона є достатньо складною і вимагає значної кількості додаткових схемних рішень. Застосування цих мікросхем виявляється виправданим, якщо «вартість» ланцюгів регенерації покривається десятками мільйонів біт в одній мікросхемі динамічного ОЗП. Проте з погляду програмованої логіки технологія динамічного ОЗП не викликає великого інтересу.

Визначення «статичне» у назві статичного ОЗП відображує те, що значення, однократно записане до комірки пам'яті, буде залишатися

незмінним доти, поки не буде спеціально змінено, або поки система не буде відключена від електроживлення. Позначення комірки пам'яті, що програмується на основі статичного ОЗП, наведено на рис. 4.12.

Рис. 4.12. Програмована комірка пам'яті на основі статичного ОЗП

Комірка містить мультитранзисторний елемент статичного ОЗП, вихід якого підключений до додаткового управляючого транзистора. Залежно від вмісту (логічний 0 або логічна 1) елементу пам'яті управляючий транзистор буде закритий (тобто відключений) або відритий (тобто включений). Один з недоліків програмованих пристроїв на основі елементів пам'яті статичного ОЗП полягає у тому, що кожна комірка займає значну площу на поверхні кремнієвого кристала, оскільки складається з чотирьох або шести транзисторів, конфігурованих у вигляді регістра-клямки. Іншим недоліком є те, що дані про конфігурацію пристрою будуть втрачені при відключенні живлення системи. Програмовані пристрої і пов'язані з ними технології програмування наведено у табл. 4.1 [24].

Таблиця 4.1 – Технології програмування

Технології	Умовне позначення	Галузь використання
Плавкі перемички		Прості ПЛП
Нарощувані перемички		ПЛІС
СППЗП		Прості та складні ПЛП
ЕСППЗП та Flash		Прості та складні ПЛП (деякі ПЛІС)
Статичний ОЗП	SRAM	ПЛІС (деякі складні ПЛП)

Треба мати на увазі, що постійно з’являються нові технології. Одна з таких технологій називається магнітним ОЗП (MRAM — magnetic RAM). Ця технологія почалася з того, що фірма IBM розробила так званий магнітний тунельний перехід. Йшлося про тришарову структуру, що складається з двох

феромагнітних шарів, розділених тонким ізолюючим шаром. Комірки магнітного ОЗП могли створюватися на перетині двох провідників, провідника рядків і провідника стовпців, з магнітним тунельним переходом між ними.

Комірки магнітного ОЗП — це висока швидкість роботи статичного ОЗП, великий обсяг динамічного ОЗП, незалежність Flash-технології і мінімальна кількість енергії, що споживається.

Класифікація архітектурних особливостей ПЛП наведена на рис. 4.13.

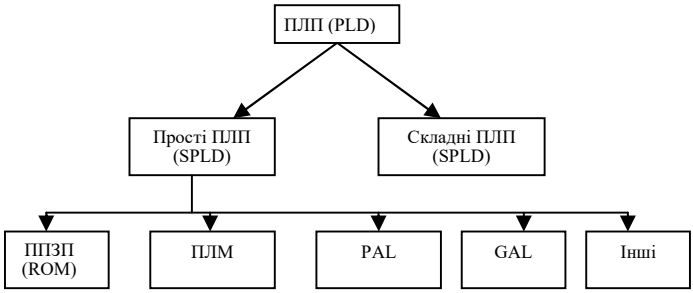


Рис.4.13. Класифікація архітектури побудови ПЛП

Першими представниками простих ПЛП були ППЗП-мікросхеми. Для розуміння роботи цих пристроїв краще всього уявити їх такими, що складаються з фіксованого масиву логічних функцій І, приєднаного до програмованого масиву логічних функцій АБО.

Для прикладу розглянемо роботу ППЗП з трьома входами і трьома виходами (рис. 4.14).

Як програмовані зв'язки в масиві елементів АБО можуть застосовуватися плавкі перемички або СППЗП-транзистори і комірки ЕСППЗП відповідно для мікросхем СППЗП і ЕСППЗП. Рис. 4.14 дає лише загальне уявлення про принцип дії даного пристрою і не відображає реальну діаграму з'єднань. В дійсності кожна функція І визначеного масиву має три входи, які приєднані до істинних або інвертованих входів a, b або c пристрою. Аналогічно кожна функція АБО програмованого масиву має вісім входів, які приєднані до виходів масиву функцій І.

Мікросхеми ППЗП спочатку призначалися для зберігання програмних інструкцій і значень констант, тобто для виконання функцій комп'ютерної пам'яті. Проте розробники використовують їх також для реалізації простих логічних функцій, таких як таблиці відповідності або кінцеві автомати. У дійсності мікросхеми ППЗП можуть використовуватися для реалізації будь-якого блока комбінаційної логіки, або комбінаційного пристрою, за умови,

що він має невелику кількість входів і виходів. Наприклад, простий ППЗП з трьома входами і трьома виходами, що показаний на рис. 4.14**,** може використовуватися для синтезу будь-якої комбінаційної функції з не більш ніж трьома вхідними і трьома вихідними параметрами. Щоб зрозуміти, як він працює, розглянемо невеликий логічний блок, наведений на рис. 4.15**.** Слід мати на увазі, що ця схема має сенс тільки для даного прикладу.

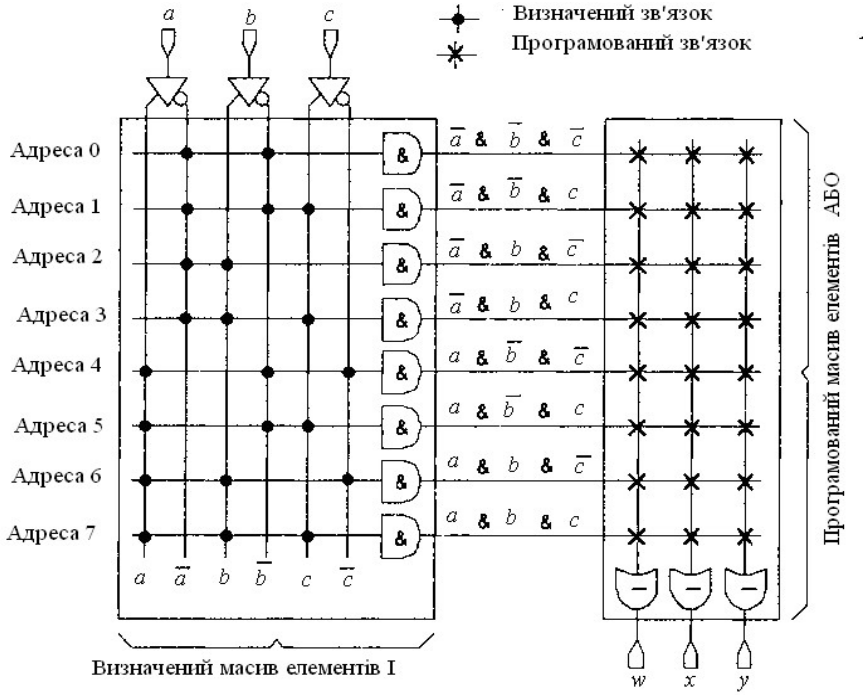
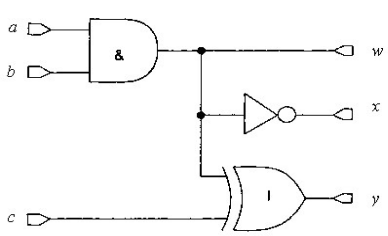


Рис. 4.14. Незапрограмована мікросхема ППЗП

Логічний блок (рис. 4.15) може бути замінений мікросхемою ППЗП з трьома входами і трьома виходами. Для цього треба всього лише запрограмувати відповідні зв'язки в масиві логічних елементів АБО (рис. 4.16).

У виразах на цьому рисунку використані такі позначення: «&» — операція І (AND), «|» — операція АБО (OR), «^» — операція ВИКЛЮЧНОГО АБО (XOR), «¯» — операція НІ (NOT).



a	b	c	w	x	y
0	0	0	0	1	0
0	0	1	0	1	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	0	1	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	0	0

Рис. 4.15. Блок комбінаційної логіки

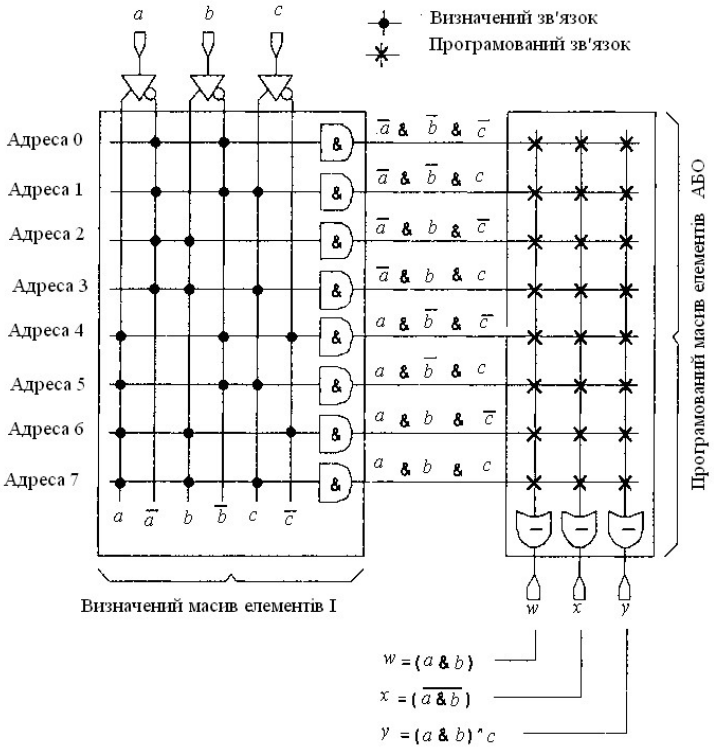


Рис. 4.16. Запрограмована мікросхема ППЗП

Розглянута вище мікросхема ППЗП є дуже простою. Насправді мікросхеми ППЗП мають значно більше входів і виходів і можуть використовуватися для реалізації великих блоків комбінаційної логіки.

Комбінаційна логіка в загальному випадку реалізовувалася за допомогою мікросхем таких як, наприклад, 74-та серія компанії Texas Instruments [24].

Той факт, що досить велике число таких мікросхем можна замінити однією мікросхемою ППЗП, означає, що монтажну плату можна зробити меншою, простішою, дешевшою і менш схильною до збоїв (кожне паяне з'єднання на монтажній платі — потенційно збійний вузол). Крім того, якщо в цій частині системи виявиться логічна помилка, наприклад якщо розробник помилково використовував функцію І замість І-НІ**,** то вона може бути легко виправлена шляхом прошивання нової ППЗП-мікросхеми або за допомогою очищення і перепрограмування СППЗП або ЕСППЗП-мікросхем. Цей спосіб більш прийнятний, ніж спосіб виявлення помилок на друкарській платі, що виконана на основі окремих мікросхем.

У логічних виразах оператор І («&») називається логічним множенням або добутком, а оператор АБО («|») — логічним складанням або сумою. Проте, коли йдеться про логічний вираз вигляду

$$y = (a\&b\&c)|(a\&b\&c)|(a\&b\&c)|(a\&d\&c),$$

поняття «літерал» означає або істинну, або інвертовану змінну (a, b і т. д.), а група літералів, що зв'язана оператором «&», називається добутком. Таким чином, добуток (*a* & *b* & *c*) складається з трьох літералів, а саме з *a*, *b* і *c*, і наведене вище рівняння буде сумою добутків.

Суть полягає у тому, що коли ці вирази використовуються для реалізації комбінаційної логіки (рис. 4.15 і 4.16), ППЗП зручно використовувати для виразів з великою кількістю добутків, але невеликою кількістю входів, оскільки всі вхідні комбінації жорстко зашиті в матриці І та завжди декодуються.

4.2. Програмовані схеми ПЛМ, PAL, GAL — призначення, архітектура, технології

Наступний ступінь розвитку ПЛП — вирішення проблем, пов'язаних з обмеженнями, властивими архітектурі ППЗП [24]. Перші програмовані логічні матриці (ПЛМ) з'явилися приблизно в 1975 році. Більшість користувачів застосовувала їх як прості ПЛП, оскільки обидва масиви, тобто і масив функцій І, і масив функцій АБО, були програмованими.

Якщо узяти просту ПЛМ з трьома входами і трьома виходами в незапрограмованому стані (рис. 4.17), то**,** на відміну від ППЗП, можна побачити, що кількість функцій І в однойменному масиві не залежить від кількості входів матриці. При цьому додаткові функції І можуть бути сформовані шляхом простого додавання в масиві рядків.

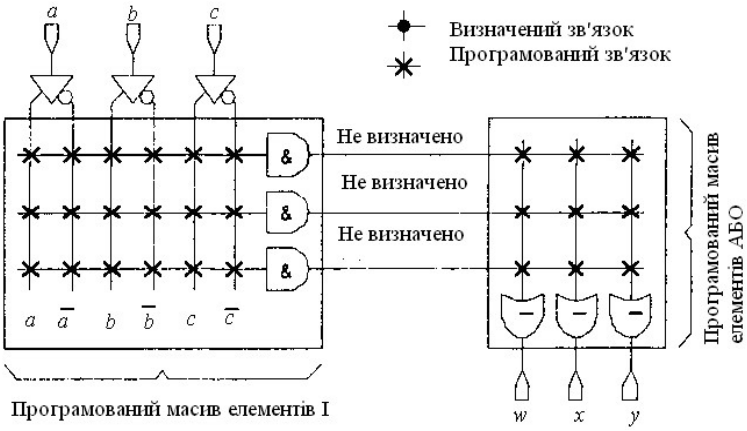


Рис.4.17. Незапрограмована ПЛМ

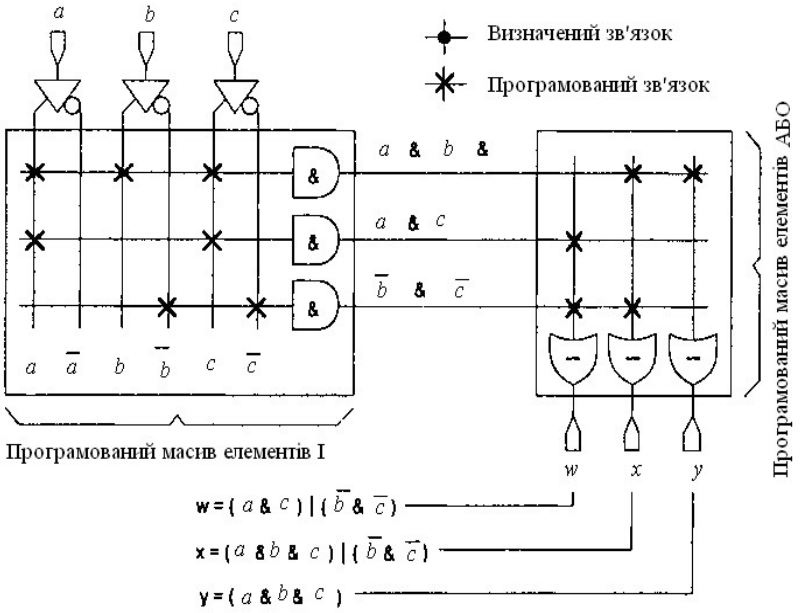


Рис. 4.18. Запрограмована ПЛМ

Аналогічна кількість функцій АБО в однойменному масиві не залежить від кількості входів матриці і від розміру масиву функцій І.

Додаткові функції АБО можуть бути сформовані шляхом простого додавання в цьому масиві стовпців.

Припустимо, що ПЛМ повинна реалізувати наступні три рівняння:

w= (a & c) | (b & c), x = (a & b & c) | (b & c) та y = (a & b & c).

Розв'язати цю задачу можна шляхом програмування відповідних зв'язків, як показано на рис. 4.18.

У ПЛМ немає жорсткої необхідності підключати масив функцій АБО до виходу масиву функцій І. Тому деякі альтернативні конфігурації, наприклад масив функцій І, підключений до входу масиву АБО-НІ, іноді показував непогані результати. Проте, не дивлячись на теоретичну можливість реалізації на практиці, функції АБО-І, І-НІ-АБО, І-НІ-АБО-НІ відносно рідко зустрічалися або просто не існували. Одною з причин, з якої ПЛМ продовжують працювати з архітектурою І-АБО чи І-АБО-НІ, стало те, що вираз «сума добутків» найчастіше використовується в логічних рівняннях. Інші типи рівнянь, такі, як добуток сум, зводяться до них за допомогою стандартних методів алгебри. Таке перетворення звичайно виконується за допомогою програм.

Мікросхеми ПЛМ рекламувалися як дуже корисні пристрої для великих схем, логічні рівняння яких характеризувалися великою кількістю однакових добутків. Наприклад (рис. 4.18), добуток вигляду (*b* & *c*) використовується двічі: для обчислення виходів *x* та *y*. Ця властивість називається розподілом добутків.

Разом з тим для проходження через програмовані зв'язки сигналам потрібно набагато більше часу, ніж при проходженні через їх визначені аналоги. Тому ПЛМ працює повільніше ніж ППЗП, оскільки обидва масиви (і функції І, і функцій АБО) є програмованими.

Для того щоб розв'язати проблему швидкодії, властивої програмованим логічним матрицям, з'явився новий клас пристроїв, що має назву «програмований масив логіки» (PAL — Programmable Array Logic). На понятійному рівні пристрої PAL майже повністю протилежні мікросхемам ППЗП, оскільки вони складаються з програмованого масиву логічних функцій І і визначеного масиву функцій АБО. Пристрої GAL (Generic Array Logic — змінний масив логіки), розроблені компанією Lattice Semiconductor Corporation, є більш складними КМОП-різновидами ідеології PAL з електричним стиранням [24].

Як приклад розглянемо простий PAL-пристрій з трьома входами і трьома виходами (рис. 4.19). Перевагою мікросхем PAL (у порівнянні з програмованими логічними матрицями) є вища швидкість, оскільки з двох масивів у них тільки один програмований.

Разом з тим програмований масив логіки більш обмежений щодо функціональності, оскільки він дозволяє складати тільки обмежену кількість добутків.

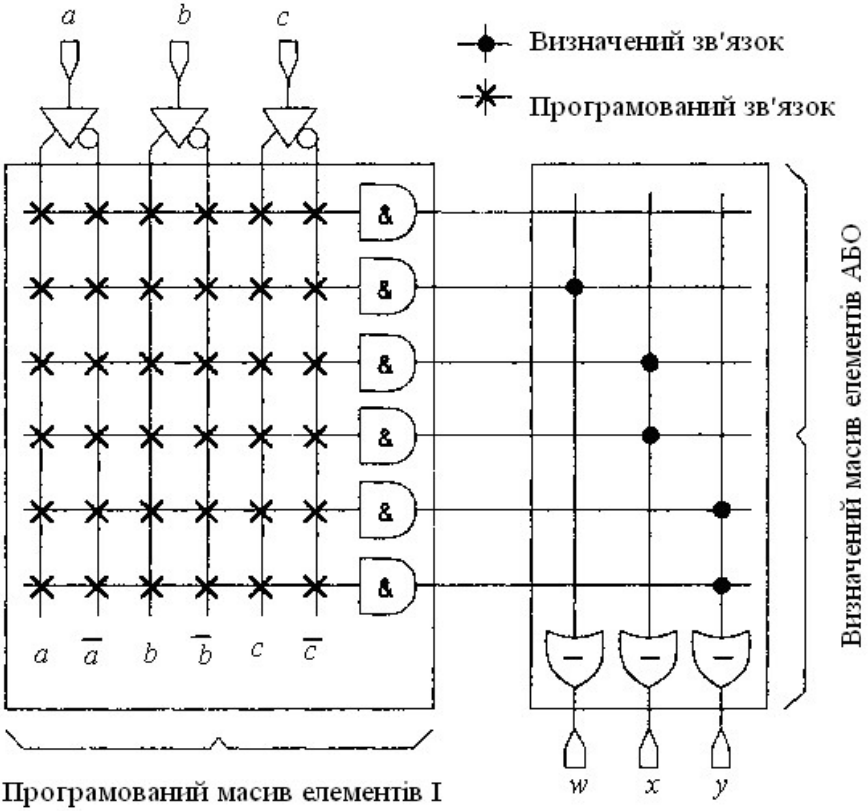


Рис. 4.19. Запрограмований пристій PAL

Розглянуті вище як приклади ПЛМ і PAL насправді дуже великі, з безліччю входів, виходів і з внутрішніми сигналами. У них можуть бути передбачені різні додаткові програмовані опції, такі як можливість інвертувати виходи, або мати виходи з трьома станами, або і те, і інше. Крім того, деякі з них підтримують регістрові виходи, тобто виходи з клямкою, аналогічно програмованим мультиплексорам, які дозволяють користувачу встановлювати по черзі для кожного виводу, яку версію виходу використовувати — регістрову або нерегістрову. Деякі пристрої дозволяють конфігурувати певні виводи як або виходи, або додаткові входи.

Проблема полягає у тому, що різні пристрої можуть забезпечувати різні набори опцій, що ускладнює вибір оптимального пристрою для конкретного додатку. У подібних випадках інженери або обмежують себе у

виборі пристроїв і підганяють свою конструкцію під ці пристрої, або використовують комп'ютерні програми, за допомогою яких вибирають пристрій максимально задовольняюче їх вимогам.

Типове задача для електроніки полягає в тому, щоб розробити пристрій з максимальними функціональними можливостями та мінімальними розмірами, ціною та потужністю, що споживається.

У зв'язку з цим в кінці 70-х — початку 80-х з'явилися складніші програмовані логічні пристрої, так звані складні ПЛП (CPLD — complex PLD) [24].

Лідерами були винахідники оригінальних пристроїв PAL — розробники компанії Monolithic Memories (MMI), які представили компонент під назвою Мега-PAL (Mega-PAL). Цей пристрій з 84 виходами по суті включав 4 PAL-пристрої і з'єднання між ними. Але Мега-PAL споживала непропорційне багато енергії, і давала невелику перевагу в порівнянні з використовуванням чотирьох окремих пристроїв [24].

Істотний прорив припав на 1984 рік, коли компанія Altera представила складний ПЛП, заснований на поєднанні КМОП- і СППЗП-технологій. Використовування КМОП дозволило компанії Altera досягти неймовірної функціональної густини і складності при порівняно невеликому споживанні енергії. Основою для програмування цих пристроїв були комірки СППЗП, і саме це зробило їх ідеальними для використовування при розробці і створенні прототипів устаткування.

Компанія Altera заслужила свою славу не тільки завдяки комбінації технологій КМОП і СППЗП. При нарощуванні архітектури простих ПЛП до рівня великих пристроїв, подібних Мега-PAL, інженери вважали, що центральний комутаційний масив, що з'єднує індивідуальні блоки простих ПЛП, відомий також як програмована комутаційна матриця, повинен на 100% з'єднуватися зі всім входам і виходам блоків. В результаті це приводило до істотного зниження швидкості, підвищення розсіюваної потужності і збільшення вартості компонентів.

Altera вдалося зробити технологічний прорив, використовуючи центральний комутаційний масив з кількістю з'єднань з входами/виходами блоків менше 100%**.** Це ускладнило програмне забезпечення для проектування ПЛП, але швидкість, споживана потужність і вартість цих пристроїв були цілком прийнятними [24].

Не дивлячись на те, що кожен виробник складних ПЛП реалізовував свої унікальні технології, в загальному випадку пристрій складався з декількох блоків простих ПЛП, зазвичай PAL, об'єднаних загальною програмованою комутаційною матрицею (рис. 4.20). Крім окремих блоків простих ПЛП можна було також запрограмувати з'єднання між ними за допомогою програмованої комутаційної матриці.

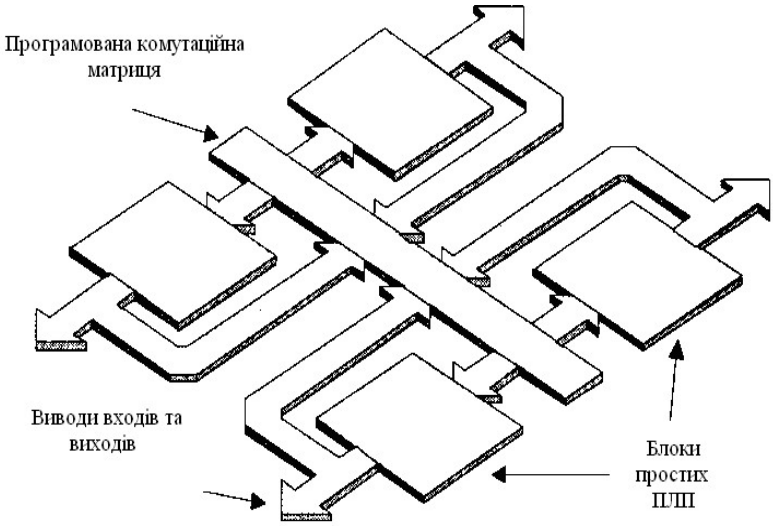


Рис. 4.20. Загальна структура складного ПЛП

На цьому рисунку не показані різні додаткові компоненти, він дає лише поверхневе уявлення про роботу складного ПЛП. Насправді всі структури пристрою сформовані на одному шматку кремнію. Наприклад, програмована комутаційна матриця може містити велику кількість провідників, наприклад 100. Але це більше, ніж може бути підключене до блока простого ПЛП, який здатний працювати тільки з обмеженою кількістю сигналів, наприклад 30. Блоки простих ПЛП пов'язані з комутаційною матрицею свого роду програмованими мультиплексорами (рис. 4.21).

Залежно від виробника і від типу пристрою, програмовані перемикачі складних ПЛП можуть бути виконані на елементах пам'яті типу СППЗП, ЕСППЗП, Flash або на статичному ОЗП. При використанні статичного ОЗП з'являється можливість збільшити універсальність цієї пам'яті, використовуючи її як програмовані перемикачі і як фактичну оперативну пам'ять.

У 1980 році комісія Об'єднаної інженерної ради з електронних пристроїв (JEDEC), підрозділ Асоціації електронної промисловості США, запропонувала стандарт форматів текстових файлів для програмування пристроїв ПЛП [24].

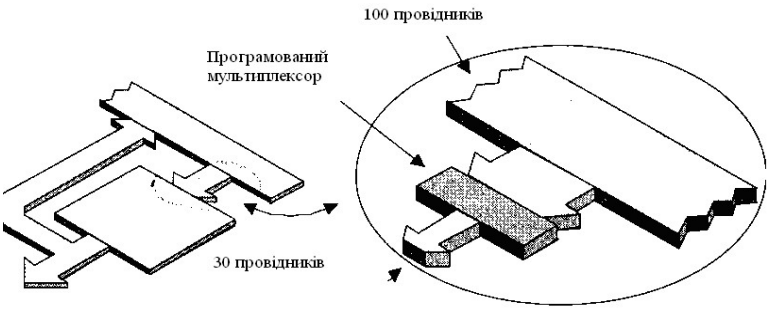


Рис. 4.21. Використання програмованого мультиплексора

У цей же час Джон Беркнер (John Birkner), людина яка розробила перші пристрої PAL і яка стояла на чолі їх подальшого розвитку, створив PAL Асемблер (PALASM). PAL Асемблер поєднує в собі мову опису апаратних засобів (HDL — Hardware Description Language) і прикладне програмне забезпечення. Як мова опису апаратних засобів PAL Асемблер дозволяє інженерам-розробникам точно описати функції принципової схеми в текстовому файлі з початковими кодами. Файл складався з булевих рівнянь, записаних у форматі «сума добутків». PALASM читав текстовий файл з початковими кодами і автоматично генерував текстовий програмний файл, придатний для програмування пристрою.

Поява PALASM, поза сумнівом, була епохальною подією для того часу, але його первинні версії підтримували тільки пристрої компанії MMI (Monolithic Memories Inc) і не підтримували які-небудь види мінімізації або оптимізації. Для вирішення цих проблем в 1983 році компанія Data I/O представила мову ABEL (Advanced Boolean Expression Language — розширена мова логічних виразів). Приблизно в цей же час компанія Assisted Technology розробила пакет CU PL (Common Universal tool for Programmable Logic — Загальні універсальні засоби проектування для програмованої логіки). Обидва продукти поєднували в собі мову HDL і програмні додатки. Крім того, що вони підтримували конструкції кінцевих автоматів і алгоритми автоматичної мінімізації логіки, вони дозволяли працювати з численними типами ПЛП пристроїв різних виробників.

Окрім PALASM, ABEL і CUPL, які були, поза сумнівом, кращими з ранніх версій мови HDL, існувало багато інших версій, таких, як AMAZE (Automated Map and Zap of Equation — автоматичне перетворення і виправлення виразів) компанії Signetics. Ці прості мови і засоби проектування, що їх супроводжують, прокладали шлях для високорівневих версій мови HDL, таких, як Verilog або VHDL, і для засобів проектування, таких як логічний синтез, які в даний час використовуються для

проектування сучасних замовних мікросхем і пристроїв з використанням ПЛІС [24].

4.3. Класифікація спеціалізованих замовних ВІС (ASIC)

Замовні інтегральні мікросхеми (ASIC — application specific integrated circuit) представлені чотирма основними класами. У порядку збільшення складності такими пристроями є вентильні матриці (для вентильної матриці часто можна зустріти інший термін — базовий матричний кристал (БМК)), структуровані ASIC, схеми на стандартних елементах і повністю замовні мікросхеми (рис. 4.22) [24].

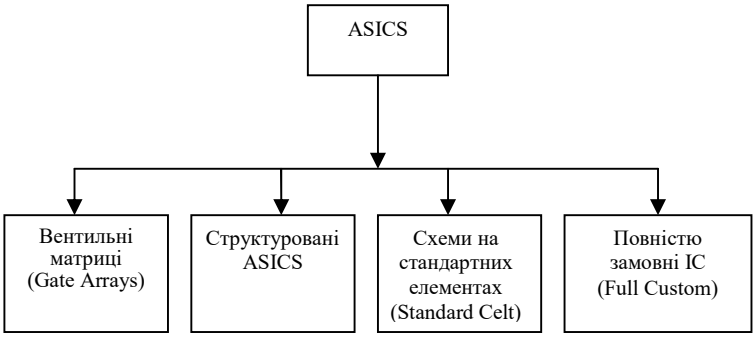


Рис. 4.22. Різні типи спеціалізованих замовних ІС

Під час появи цифрових мікросхем існувало тільки два типи пристроїв, окрім, природно, мікросхем пам'яті. До першого типу належали відносно прості блоки, які вироблялися такими компаніями, як Texas Instruments або Fairchild, і які продавалися як стандартні компоненти. До другого типу належали замовні спеціалізовані мікросхеми, такі як, мікропроцесори, які розроблялися і вироблялися за замовленням.

При створенні замовних пристроїв розробка маски для кожного шару мікросхеми була прерогативою інженерів — розробників кінцевого пристрою. Постачальники спеціалізованих мікросхем заздалегідь не створювали ніяких компонентів на кремнієвому кристалі заводським способом і не поставляли бібліотек визначених логічних вентилів або функцій.

Використовуючи відповідні засоби проектування інженери, задавали розміри окремих транзисторів і на їх основі створювали функції більш високого рівня. Наприклад, при необхідності збільшити швидкість логічного

вентиля вони могли змінити розміри транзисторів, що використовувалися для побудови вентилів. Засоби проектування для розробки замовних пристроїв часто створювалися інженерами фірм-розробників.

Розробка замовних пристроїв є дуже складним і трудомістким процесом, але створені у такий спосіб мікросхеми містять необхідну кількість вентилів з мінімальними втратами вільного місця на кремнієвому кристалі.

4.4. Архітектура базисних модулів структурованих ASIC

У середині 60-х компанія Fairchild Semiconductor представила новий пристрій під назвою мікроматриця (Micromatrix). Цей пристрій складався з невеликої кількості, близько 100, ні до чого не підключених транзисторів. Для того щоб цей пристрій міг виконувати корисні функції, розробники уручну викреслювали на двох пластикових листах шари металізації для з'єднання транзисторів.

На першому листі зображалися провідники по осі У, тобто зверху вниз, за допомогою яких виготовлявся перший шар металізації. На другому листі зображалися провідники по осі Х, тобто зліва направо, для виготовлення другого шару металізації. Крім того, використовувалися додаткові листи для зображення перехідних отворів, що сполучають перший шар металізації з транзисторами, і для перехідних отворів, що сполучають перший і другий шари металізації.

Виготовлення пристрою у такий спосіб було вкрай трудомістким заняттям і сприяло виникненню помилок, але так чи інакше дорога і по-справжньому трудомістка робота із створення транзисторів виконувалася. Іншими словами, мікроматриця дозволяла розробникам створювати замовні пристрої, прийнятні за вартістю і термінами виготовлення .

Після декількох років, а точніше в 1967 році, компанія Fairchild представила новий пристрій, що одержав назву мікромозаїки (Micromosaic). Цього разу пристрій містив декілька сотень окремих транзисторів, які могли з'єднуватися разом, утворюючи близько 150 логічних елементів І, АБО і НІ. Унікальність мікромозаїки полягала у тому, що розробники могли задавати функції замовного пристрою, використовуючи логічні вирази у вигляді текстового файла, за допомогою якого комп'ютерна програма визначала необхідні міжтранзисторні з'єднання і виготовляла фотошаблон, за яким вже створювався пристрій. Такий підхід був справжньою революцією для того часу, і сьогодні мікромозаїка вважається передвісником сучасних вентильних матриць як різновиду спеціалізованих замовних мікросхем і першим справжнім додатком систем автоматизованого проектування (САПР).

Ідея створення матриць логічних елементів, або вентильних матриць**,** розглядалася ще в кінці 60-х в компаніях IBM, Fujitsu та ін. Проте перші пристрої використовувалися тільки для внутрішнього споживання, і лише у середині 70-х вентильні матриці, виготовлені за технологією КМОП, стали доступні широкому кругу споживачів. Перші вентильні матриці називалися нескомутованими логічними матрицями (ULA — uncommitted logic array). З часом цей термін перестав вживатися [24].

Основу вентильних матриць складає базисна комірка, що складається з набору нез'єднаних транзисторів та резисторів. Кожен постачальник замовних мікросхем самостійно визначає оптимальний набір компонентів, що входять до складу окремої базисної комірки (рис. 4.23).

Постачальники замовних мікросхем починали свою роботу з виготовлення заводським способом кремнієвого кристала, що містить масив базисних комірок. У разі канальної матриці базисні комірки, як правило, розташовувалися у вигляді одностовпцевих або двостовпцевих масивів, причому між стовпцями були вільні області, так звані канали (рис. 4.24).

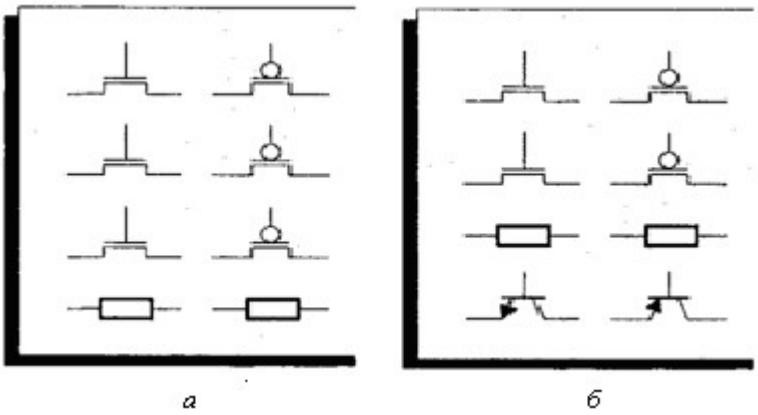


Рис. 4.23. Приклади простих базисних комірок вентильної матриці:
а – однорідна КМОП; б – змішана біполярна КМОП

За відсутності каналів на матриці комірки розташовувалися у вигляді одного великого масиву. Поверхня такого пристрою була покрита великою кількістю комірок і не мала виділених каналів для внутрішніх з'єднань. Такі матриці називалися безканальними**.**

Розробники можуть використовувати визначений постачальником мікросхем набір логічних функцій, таких, як прості логічні вентилі, мультиплексори і регістри. Кожна з цих функцій-блоків розглядається як окремий елемент або комірка (не плутати з базисною коміркою). Набір таких

функцій, що підтримується постачальником, називається бібліотекою елементів.

Рис. 4.24. Архітектура канальних логічних матриць:
а – одностовпцеві масиви: б – двостовпцеві масиви

Розробники одержують таблицю з'єднань на рівні вентилів, яка описує, які вентилі використовуватимуться, і які між ними будуть з'єднання. Щоб встановити відповідність логічних вентилів і базисних комірок і визначити, як ці комірки зв'язуватимуться між собою, використовуються спеціальні програми визначення відповідностей, програми розставляння, трасування та інші. На основі отриманих результатів виготовляються фотошаблони, за допомогою яких створюються шари металізації. Ці шари зв'язуватимуть компоненти усередині базисної комірки, а також комірки між собою, з входами та виходами пристрою.

Вентильні матриці мають розумну ціну, оскільки транзистори і інші компоненти виробляються заводським способом, а за замовленням виготовляються тільки шари металізації. Більшість таких пристроїв не використовує всіх внутрішніх ресурсів. Крім того, розташування вентилів жорстко визначене, а трасування внутрішніх з'єднань далеке від оптимального. Усі ці недоліки негативно позначилися на продуктивності і споживаній потужності цих пристроїв.

Вирішення проблем, властивих вентильним матрицям стало можливим з появою на початку 90-х схем на стандартних елементах. Ці компоненти мали багато спільного з вентильними матрицями.

Постачальник спеціалізованої замовної (ASIC) мікросхеми складає бібліотеку елементів, і цією бібліотекою можуть користуватися розробники. Постачальник також підтримує бібліотеки апаратних і програмних макровизначень, які включають такі елементи, як процесори, функції зв'язку,

ОЗП або ПЗП. Розробники можуть також використати раніше створені функції або придбати блоки інтелектуальної власності.

Коли команда інженерів конструює складну мікросхему, вона може ухвалити рішення про покупку вже готового проекту одного або декількох функціональних блоків. Проект таких функціональних блоків називається інтелектуальною власністю або IP (intellectual property). Засоби інтелектуальної власності можуть охоплювати будь-які функціональні блоки, зокрема функції зв'язку і мікропроцесори. Найскладніші функції, такі, як мікропроцесори, можуть називатися ядрами.

Тим або іншим способом в наші дні за допомогою складних систем автоматизованого проектування розробники завершують свою роботу створенням таблиці з'єднань на рівні вентилів. Така таблиця описує, які вентилі використовуються і як вони з'єднані між собою.

На відміну від вентильних матриць, схеми на стандартних елементах не використовують концепцію базисних комірок, і компоненти на кремнієвому кристалі не виготовляються заводським способом. Для індивідуального розташування кожного вентиля в таблиці з'єднань і для оптимального розводення з'єднань між ними використовуються спеціальні засоби автоматизованого проектування. Отримані результати використовуються для створення замовних фотошаблонів для кожного шару мікросхеми.

Концепція схем на стандартних елементах дозволяє створювати логічні функції, використовуючи мінімальну кількість транзисторів без якої-небудь надмірності компонентів; самі функції можуть бути реалізовані так, щоб зробити менш трудомістким створення внутрішніх зв'язків між ними. Саме тому схеми на стандартних елементах забезпечують майже оптимальне використання поверхні кремнієвого кристала в порівнянні з вентильними матрицями.

Структуровані спеціалізовані мікросхеми, до речі спочатку вони називалися інакше, з'явилися на світ приблизно на початку 90-х. Через 10 років, приблизно в 2001-2002 році, деякі виробники приступили до вивчення способів зниження витрат на проектування спеціалізованих мікросхем і скорочення тривалості їх розробки. Приблизно в 2003 році з'явилася назва «структуровані спеціалізовані мікросхеми» (structured ASIC) [24].

У кожного постачальника власна архітектура пристрою. У зв'язку з цим будуть розглянуті тільки загальні риси цих пристроїв. Кожен пристрій починається з фундаментального елемента, який одні називають модулем, а інші – елементом мозаїки. Такий елемент може містити виготовлений заводським способом набір загальної логіки, виконаної у вигляді логічних вентилів, мультиплексорів або таблиць відповідності, одного або декількох регістрів, і, можливо, невеликого локального ОЗП (рис. 4.25).

Масив таких елементів виготовляється заводським способом по всій поверхні кристала. Деяка альтернативна архітектура починається або з базисної комірки, або з базисного модуля, або з базисного елемента мозаїки, або з чогось іншого. До її складу входять тільки елементи загальної логіки у формі виготовлених заводським способом вентилів, мультиплексорів або таблиць відповідності. Масив таких базисних одиниць (говорять 4x4, 8x8 або 16x16) в поєднанні з деякими спеціальними модулями, що містять регістри, невеликі елементи пам'яті і іншу логіку, утворює базову комірку або базовий модуль, або базовий елемент мозаїки, або ін. Цей масив виготовляється заводським способом по всій поверхні кристала.

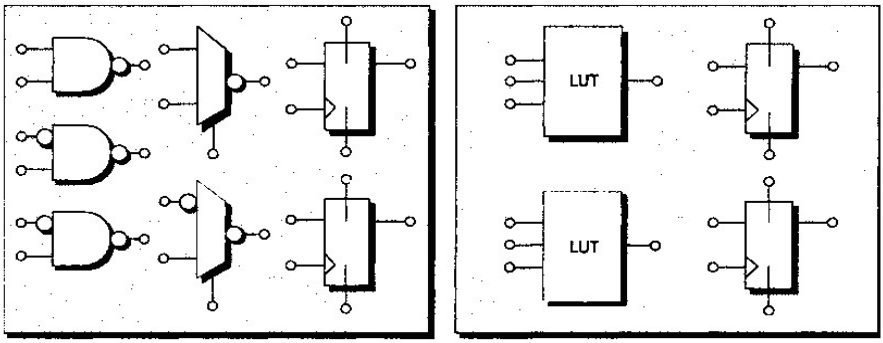


Рис. 4.25. Модулі спеціалізованих структурованих ІС:
а – модуль, що містить вентилі, мультиплексори та тригери;
б- модуль, що містить таблиці відповідності (LUT) та тригери

Заводським способом також виготовляються, зазвичай по контуру кристала, деякі додаткові функції, наприклад блоки ОЗП, тактові генератори, логіка периферійного сканування та інші функції (рис. 4.26).

При виконанні пристрою за замовленням виготовляються тільки шари металізації, як і у разі стандартних вентильних матриць. Відмінність криється у тому, що внаслідок більшої складності модулів структурованих спеціалізованих ІС більшість шарів металізації вже визначена. Таким чином, при виготовленні багатьох структурованих мікросхем за замовленням їх архітектура потребує всього лише двох або трьох шарів металізації, а в деяких випадках вимагається виготовити тільки один шар з перехідними отворами. У результаті значно знижується вартість і скорочується час виготовлення фотошаблонів, що залишилися, необхідних для створення пристрою.

Визначити точне значення параметрів досить складно, проте визначена і виготовлена заводським способом логіка структурованих мікросхем, в

порівнянні з схемами на стандартних елементах, споживає велику потужність, має більшу продуктивність і займає більше місця на кристалі. Для виконання тих самих функцій структуровані мікросхеми в середньому потребують в три рази більше місця на кристалі і споживають в два-три рази більше енергії, ніж схеми на стандартних елементах. На практиці ці параметри істотно залежать від типу пристрою і його архітектури.

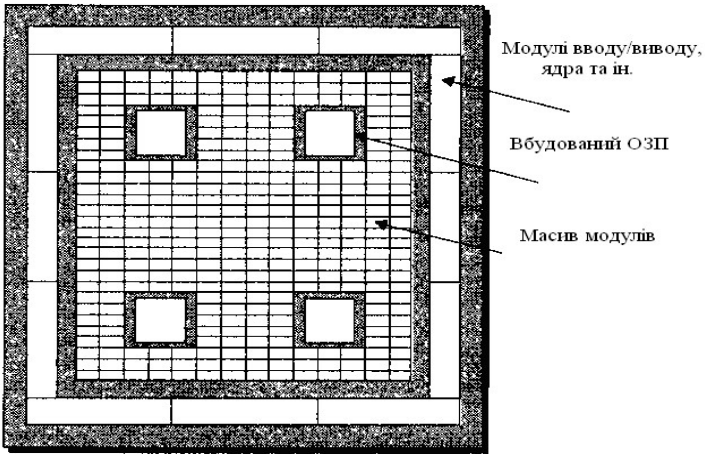


Рис. 4.26. Типова спеціалізована структурована ІС

4.5. Архітектура й технології проектування ПЛІС

На початку 80-х існували програмовані пристрої, подібні до простих і складних ПЛП, які мали відносно невелику вартість при малому часі виготовлення та модифікації. Але ці пристрої не були здатні підтримувати великі і складні функції.

З іншого боку, існували замовні спеціалізовані інтегральні мікросхеми. Вони підтримували надзвичайно великі і складні функції, але були надзвичайно дорогими, і для їх виготовлення був потрібен значний час. Крім того, остаточний варіант замовної мікросхеми «заморожувався в кремнії».

Для ліквідації недоліків означених пристроїв фірма Xilinx розробила новий клас мікросхем FPGA (Field programmable gate arrays), або ПЛІС (програмовані логічні інтегральні схеми) [24], які з'явилися у продажу в 1984 році. Перші ПЛІС виготовлялися за КМОП-технологією і для зберігання конфігурації використовували статичний ОЗП. Не дивлячись на те що перші версії цих пристроїв були порівняно простими і містили, за сьогоднішніми

мірками, відносно мало вентилів або їх еквівалентів, багато уявлень, що лежать в основі їх архітектури, використовуються і в наші дні.

Основу перших ПЛІС складала концепція програмованих логічних блоків, які включали 3-входову таблицю відповідності (LUT — lookup table), регістр, що виконує функцію тригера або клямки, мультиплексор, а також деякі інші елементи. На рис. 4.27 як приклад показаний дуже простий програмований логічний блок (логічний блок сучасної ПЛІС може бути набагато складнішим).

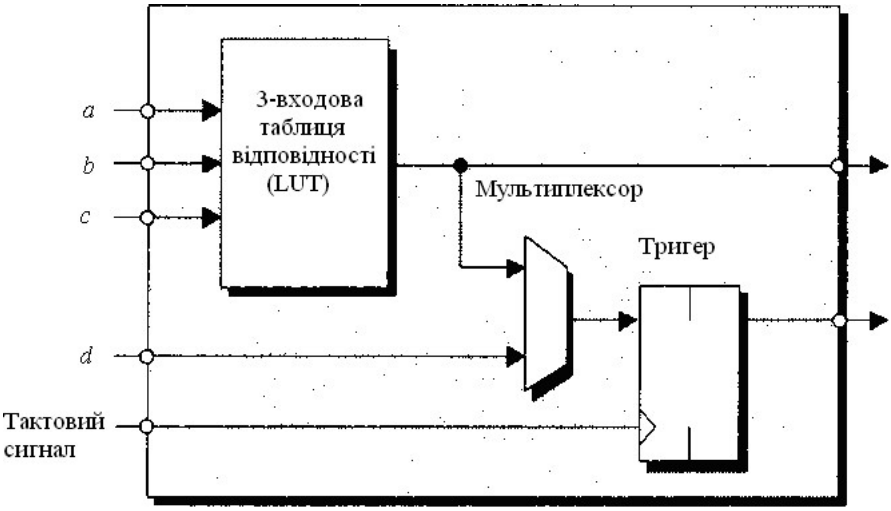


Рис. 4.27. Простий програмований логічний блок

Кожна ПЛІС містить велику кількість таких програмованих блоків. Шляхом програмування відповідних комірок статичного ОЗП кожен логічний блок пристрою може бути конфігурований для виконання різних функцій. У процесі конфігурації в кожен регістр записується його початкове значення у вигляді логічного 0 або логічної 1, а також визначається, чи регістр виконуватиме функцію тригера або клямки (рис. 4.27). У першому випадку також визначається, по якому фронту тактового сигналу, позитивному або негативному, вироблятиметься перемикання. Тактові сигнали є загальними для всіх логічних блоків. Мультиплексор, підключений до входу тригера, може конфігуруватися на передачу сигналів з виходу таблиці відповідності або з окремого входу логічного блока. Таблиця відповідності може програмуватися на роботу як будь-яка логічна функція з трьома входами і одним виходом.

Припустимо, що таблиці відповідності необхідно сформувати функцію

$y = (a \,\&\, b) \mid \text{not } c$.

Для цього в таблицю відповідності необхідно завантажити відповідні вихідні значення цієї функції (рис. 4.28).

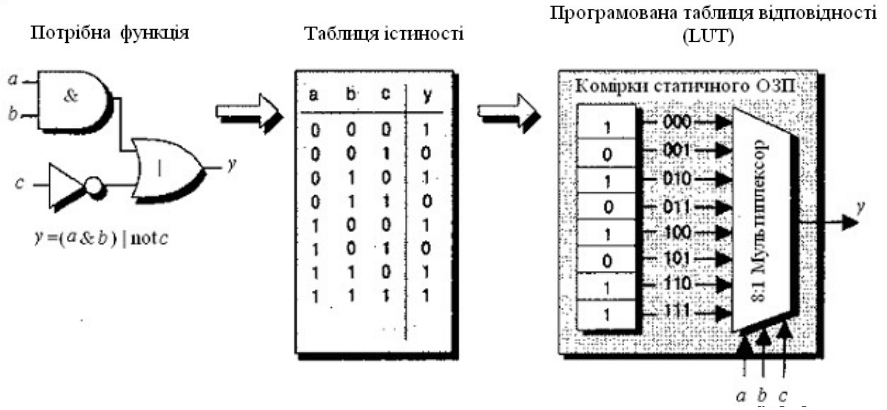


Рис. 4.28. Конфігурація таблиці відповідності

Таблиця відповідності з мультиплексором 8:1, наведена на рис. 4.28, є спрощеним варіантом існуючих таблиць відповідності. ПЛІС складається з великого числа програмованих логічних блоків з програмованими внутрішніми з'єднаннями (рис. 4.29).

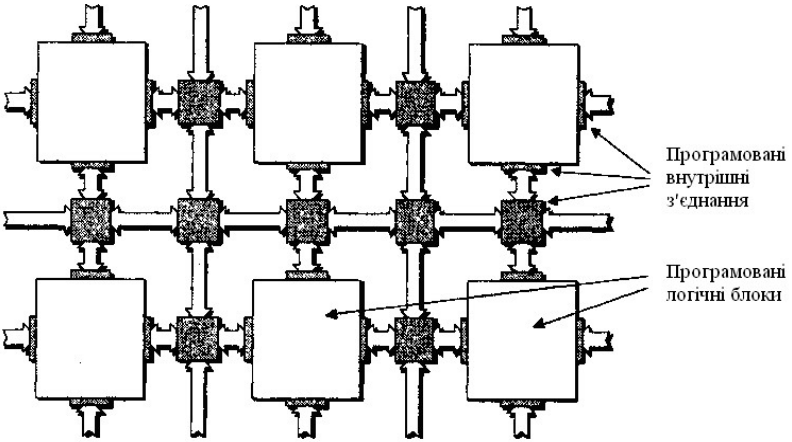


Рис. 4.29. Спрощена архітектура ПЛІС

Спрощена схема являє собою лише наближене уявлення про архітектуру ПЛІС. Насправді всі транзистори і внутрішні з'єднання

виконуються на одному кристалі кремнію за допомогою стандартних технологій виготовлення інтегральних мікросхем.

Окрім локальних внутрішніх зв'язків, показаних на рис. 4.29, існують глобальні, або високошвидкісні, з'єднання, які передають сигнали через кристал без участі численних локальних перемикаючих елементів.

Пристрій також містить контактні площадки і ніжки для вводу/виводу даних (на рис. 4.29 не показані). За допомогою елементів пам'яті статичного ОЗП внутрішні з'єднання програмуються так, щоб входи мікросхеми з'єднувалися з входами одного або декількох програмованих логічних блоків, а виходи цих блоків підключалися до входів інших блоків та/або подавалися на виходи мікросхеми.

ПЛІС успішно справилися з роллю моста між ПЛП і замовними спеціалізованими мікросхемами. З одного боку, вони мають високий ступінь можливості конфігурації з малим часом виготовлення і модифікації, як і ПЛП. З іншого боку, вони можуть використовуватися для реалізації великих і складних функцій, які раніше реалізовувалися тільки за допомогою замовних мікросхем. Спеціалізовані замовні мікросхеми призначалися для дійсно великих, складних і високотехнологічних систем. Проте на місце, відведене цим мікросхемам у сфері програмованої логіки, претендують ПЛІС, які у міру вдосконалення почали поступово їх витісняти.

Принцип розробки за зразком, або розробки платформи, тривалий час використовувався на рівні друкарської плати. Згідно з цим принципом на основі базової конфігурації може бути реалізована безліч похідних продуктів.

Сучасні високотехнологічні ПЛІС, крім величезної кількості програмованої логіки, містять вбудовані блоки ОЗП, вбудовані процесорні ядра, високошвидкісні блоки вводу/виводу і багато що інше. Крім того, розробники мають доступ до великого набору блоків інтелектуальної власності. Все це сприяло розвитку концепції ПЛІС-платформи. Суть її полягає у тому, що будь-яка компанія може використовувати вже спроектовану ПЛІС-платформу для багатьох пристроїв внутрішнього користування або запропонувати початкові проекти іншим компаніям для виготовлення замовних пристроїв.

Очевидно, що немає ніякого сенсу вбудовувати компоненти замовних інтегральних схем всередину ПЛІС, оскільки отримана інтегральна схема буде джерелом класичних проблем, які властиві технології створення замовних мікросхем: високі витрати на проектування і виробництво, тривалі терміни виходу на ринок і інші. Проте існує ряд випадків, коли один або декілька компонентів ПЛІС використовуються як частина замовних мікросхем на стандартних елементах.

Одна з причин вбудовування компонентів ПЛІС всередину замовних мікросхем — зробити якомога простішим принцип розробки платформи. В

цьому випадку платформа належатиме до класу замовних мікросхем, але вбудовані компоненти ПЛІС забезпечуватимуть один з механізмів, що використовується для адаптації і модифікації виробу.

Інша причинна пов'язана з тим, що останні декілька років спостерігається розширення сфери застосування ПЛІС, зокрема вони можуть використовуватися для надання додаткових властивостей виробам на замовних мікросхемах. В цьому випадку великі і складні спеціалізовані мікросхеми з'єднуються з ПЛІС, розташованими на одній друкарській платі в безпосередній близькості один від одного (рис. 4.30).

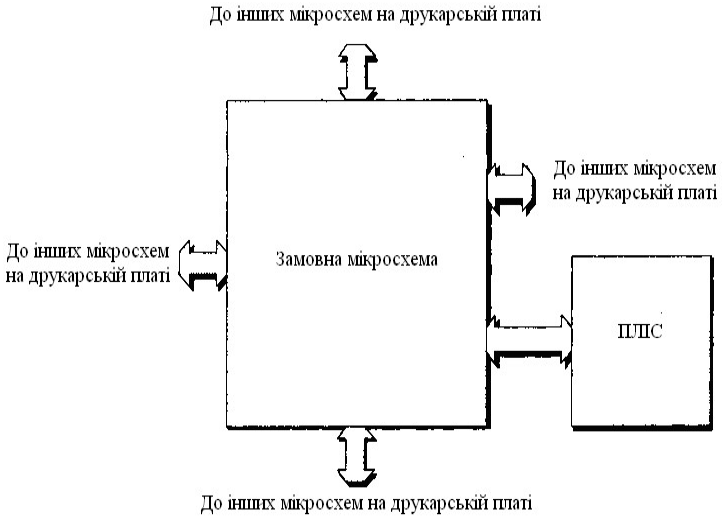


Рис. 4.30. Використання ПЛІС в комплексі із замовними мікросхемами

Застосування цієї схеми виправдане великою трудомісткістю і дорожнечею операцій з усунення помилок і зі зміни функціональності системи при використанні замовних мікросхем. Навіть якщо виріб на замовних мікросхемах не містить помилок, застосування ПЛІС можна використовувати для виконання низькорівневої модифікації і модернізації системи. Недоліком такої технології є тривалість проходження сигналів від замовної мікросхеми до ПЛІС і назад. Усунути цей недолік можна, вбудувавши ядро ПЛІС всередину замовної мікросхеми. У результаті одержимо такий гібрид: ПЛІС — замовна інтегральна схема (FPGA-ASIC).

При цьому треба мати на увазі, що системи автоматизованого проектування і методи проектування замовних мікросхем і ПЛІС істотно розрізняються. Наприклад, про замовні мікросхеми можна сказати, що вони дрібномодульні, оскільки в основному вони створюються на рівні простих

логічних елементів. Це означає, що традиційні методи проектування, такі, як логічний синтез, традиційні методи розміщення елементів і трасування з'єднань також застосовні до проектування дрібномодульних замовних інтегральних схем.

Що стосується ПЛІС, то про них можна сказати, що вони є середньо модульними (або крупномодульними, оскільки фізично вони реалізуються з використанням високорівневих блоків, таких як, програмовані логічні блоки). В цьому випадку для проектування краще використовувати специфічні для ПЛІС методи синтезу, розміщення елементів і трасування з'єднань.

Областю застосування гібридів вигляду ПЛІС — замовна мікросхема є структуровані спеціалізовані мікросхеми, або структуровані ASIC, оскільки вони також відповідають принципам блокової побудови. Коли постачальники структурованих спеціалізованих мікросхем обирають засоби проектування, вони частіше спілкуються з постачальниками ПЛІС-ОРІЄНТОВАНИХ засобів синтезу, технологій розміщення елементів і трасування з'єднань, ніж з їх колегами, що пропонують традиційні засоби. Таким чином, для проектування гібридів вигляду ПЛІС-ЗАМОВНА мікросхема, заснованих на структурованій ASIC, автоматично можна користуватися єдиними засобами розробки, оскільки ті самі методи блокового синтезу, розміщення і трасування можуть використовуватися для виготовлення як «замовної», так і «ПЛІС»-частини мікросхеми.

Деякі частини пристроїв подібні до програмованих логічних блоків і до базових сполучних структур, і в цьому випадку постачальники борються за кожен квадратний мікрон і кожну частинку наносекунди. Такими частинами пристроїв є транзистори і провідники, що виготовляються вручну за технологією замовних мікросхем. Ці частини пристрою є відносно малими і часто повторюються, тобто будучи один раз створені, вони можуть тисячі разів відтворюються на поверхні кристала.

Існують і допоміжні частини пристроїв, такі як схеми управління, що конфігуруються, які зустрічаються в одиничному екземплярі і до яких не ставляться жорсткі вимоги за розмірами і продуктивністю. Ці частини виготовляються методами, які використовуються для створення схем на стандартних елементах.

4.5.1. Архітектура базисного модуля ПЛІС. Реалізація на комірках ОЗП та на мультиплексорах

При побудові ПЛІС використовуються раніш розглянуті технології: метод нарощуваних перемичок, використання комірок статичного ОЗП, ЕСППЗП та ін. До складу ПЛІС також можуть входити вбудовані блоки: мультиплексори, суматори, блоки ОЗП, мікропроцесорні ядра та ін.

Більшість ПЛІС використовує для зберігання конфігурації комірки пам'яті статичного ОЗП, які можуть бути багато разів перепрограмовані. Головною перевагою цієї технології є те, що вона дозволяє легко і швидко реалізувати і протестувати всі нові ідеї, відносно легко підстроюючись під нові стандарти і протоколи. Крім того, при включенні системи така ПЛІС може бути спочатку запрограмована для виконання певних функцій, наприклад для самотестування або тестування всієї системи, а потім може бути перепрограмована для виконання своєї головної задачі.

Інша істотна перевага використання комірок статичного ОЗП полягає у тому, що ця технологія є передовою. На постачальників ПЛІС також діє той факт, що багато компаній, які спеціалізуються на пристроях пам'яті, витрачають величезні ресурси на дослідження і розвиток комірок статичного ОЗП. Більш того, ці елементи пам'яті створюються за такою ж КМОП-технологіїю, як і решта частин ПЛІС, тобто для створення цих компонентів не потрібні якісь спеціальні технології.

Раніше для апробації виробництва мікросхем за новими технологіями часто використовувалися мікросхеми пам'яті. Останнім часом сукупність таких характеристик, як розмір, складність і безперебійність останніх поколінь ПЛІС зробили можливим їх застосування для розв’язання і цих задач. На відміну від пристроїв пам'яті, ПЛІС дозволяють легко ідентифікувати і знаходити дефекти структури, тобто встановлювати факт помилки і навіть визначати, що і де відбулося. Наприклад, коли компанії IBM і UMC впроваджували технологію 0.09 мкм (90 нм), ПЛІС фірми Xilinx стали першими пристроями, виготовленими за такою технологією [24].

Недоліком пристроїв на основі статичного ОЗП є те, що потрібно змінювати їх конфігурацію кожного разу при включенні системи. Для цього доводиться використовувати спеціальну зовнішню пам'ять, що збільшує вартість системи і вимагає місця на друкарській платі, або використовувати вбудований мікропроцесор, або реалізувати якісь інші варіанти.

На відміну від пристроїв, заснованих на комірках статичного ОЗП, які програмуються при включенні системи, пристрої на основі нарощуваних перемичок програмуються у вимкненому стані за допомогою так званого програматора.

ПЛІС на основі нарощуваних перемичок мають ряд переваг. По-перше, ці пристрої є енергонезалежними, тобто їх конфігураційні дані не стираються при відключенні живлення системи. Це означає, що вони готові до роботи відразу після включення системи. Будучи незалежними, ці пристрої не вимагають додаткової мікросхеми зовнішньої пам'яті для зберігання конфігураційних даних. Це дозволяє зменшити вартість всієї системи і зберегти вільне місце на друкарській платі.

Одна з переваг ПЛІС на основі нарощуваних перемичок заслуговує особливої уваги: структура їх внутрішніх з'єднань є дійсно «наджорсткою», а також такі пристрої відносно стійкі до радіації. Ця властивість перемичок може становити певний інтерес для військових і космічних додатків. Стан конфігураційних комірок компонентів статичного ОЗП може бути «перемкнутий» в довільний стан, якщо ці комірки будуть уражені радіацією, якої так багато в космосі. Порівняно з ними, одного разу запрограмована нарощувана перемичка не може бути змінена у такий спосіб. При цьому слід мати на увазі, що будь-які тригери в цьому пристрої залишаються чутливими до дії радіації, тому мікросхеми, призначені для роботи в умовах радіаційної дії, повинні мати захист тригерів у вигляді системи потрійного резервування. Іншими словами, у системі повинні бути три копії кожного регістра, і рішення про значення вихідного сигналу ухвалюється більшістю голосів. У ідеальному випадку всі три регістри міститимуть однакове значення, але якщо один з регістрів «перемикається» так, що два з них містять 0, а третій 1, то вихідним значенням буде 0, і, навпаки, якщо два регістри містять 1, а третій 0, то вихідним значенням вважатиметься 1.

Але, можливо, найважливішою перевагою ПЛІС на основі нарощуваних перемичок є те, що їх конфігураційні дані приховані глибоко всередині. За визначенням програматор може прочитати ці дані, оскільки це частина його роботи. Після нарощування кожної перемички програматор повинен провести тестування, щоб переконатися, що елемент успішно запрограмований, і лише потім перейти до наступної перемички. Крім того, програматор може використовувати автоматичну перевірку успішної конфігурації пристрою. Щоб це зробити, програматору потрібно буде здійснити читання стану нарощуваних перемичок і отримані результати порівняти з необхідним станом, який визначається в конфігураційному файлі.

Після програмування пристрою слід скористатися можливістю нарощувати спеціальний захист перемичок, і, таким чином, забезпечити захист від зчитування будь-яких запрограмованих даних з пристрою (у формі наявності або відсутності перемичок). Навіть якщо пристрій буде розкритий, запрограмовані і незапрограмовані перемички залишаються ідентичними, до того ж той факт, що нарощувані перемички втоплені у внутрішніх шарах металізації, робить конструкцію недоступною для зворотного проектування.

Пристрої на нарощуваних перемичках є одноразово програмованими, і це їх головний недолік, оскільки, якщо ПЛІС вже була одного разу запрограмована, змінити її конфігурацію вже неможливо.

Конфігураційні комірки ПЛІС на основі ЕСППЗП або Flash-пам'яті так само, як і елементи пам'яті статичного ОЗУ, утворюють довгий ланцюжок, подібно до регістру із зсувом. Ці пристрої можуть програмуватися у відключеному стані за допомогою програматора. Деякі версії пристроїв

можна програмувати, не вимикаючи живлення, але при цьому час їх програмування приблизно в три рази перевищує час програмування пристроїв на статичному ОЗП.

Після програмування дані, що містяться в пристроях, будуть енергонезалежними, тобто ці пристрої будуть відразу готові до роботи після подачі напруги на систему. Що стосується захисту, то деякі з цих пристроїв використовують принцип мультибітного ключа, розмір якого приблизно становить від 50 до декількох сотень бітів. Після програмування пристрою можна завантажити в нього особистий ключ, тобто бітову комбінацію, для захисту конфігураційних даних. Річ у тому, що, якщо завантажити ключ, то прочитати дані з пристрою або записати в нього нові дані можна буде тільки, завантаживши копію ключа через JTAG-порт. Якщо при цьому врахувати, що JTAG-порт сучасних пристроїв працює на частоті 20 Мгц, то злом ключа методом перебору кожного можливого значення забере мільйони років.

Двотранзисторні ЕСППЗП- і Flash-елементи пам'яті приблизно в два з половиною рази більші за розміром ніж однотранзисторні, але вони істотно менші, ніж комірки статичного ОЗУ. Це означає, що пристрій може бути виконаний компактнішим, з меншими затримками на внутрішніх з'єднаннях.

Разом з тим ці пристрої вимагають приблизно п'ять додаткових технологічних кроків після завершення стандартного технологічного циклу КМОП. Саме в цьому криється причина їх відставання від пристроїв на статичному ОЗП. Ще один суттєвий недолік полягає в тому, що ці пристрої мають тенденцію зберігати відносно високу потужність споживання в статичному режимі через велику кількість резисторів навантажень, що міститься в них.

Деякі виробники ПЛІС часто пропонують різні комбінації технологій програмування. Розглянемо, наприклад, пристрій, кожен конфігураційний елемент якого є комбінацією Flash- (або ЕСППЗП-) елемента пам'яті і пов'язаного з нею елемента пам'яті статичного ОЗП.

У цьому випадку Flash-елементи можуть бути заздалегідь запрограмовані. Потім, після включення системи, вміст Flash-елементів пам'яті масово паралельно копіюється у відповідні до них елементи пам'яті статичного ОЗП. Така технологія забезпечує незалежність, властиву пристроям на основі нарощуваних перемичок, а це означає, що пристрій буде негайно готовий до роботи після включення системи. Але, на відміну від пристроїв на основі нарощуваних перемичок, елементи пам'яті статичного ОЗП можна використовувати для перепрограмування пристрою у включеному стані. Аналогічно можна перепрограмувати Flash-елементи пам'яті пристрою, не знімаючи напруги живлення, або виконати процедуру перепрограмування за допомогою програматора після виключення системи.

У загальному випадку ПЛІС підрозділяються на дрібномодульні та крупномодульні. Щоб зрозуміти цю класифікацію, треба пам'ятати, що головною особливістю ПЛІС є їх внутрішня структура, яка переважно складається з великої кількості простих програмованих логічних блоків та програмованих внутрішніх зв'язків (рис. 4.29). У дрібномодульній архітектурі кожен логічний блок може використовуватися для реалізації тільки дуже простої функції.

Дрібномодульні структури використовуються при реалізації зв'язуючої логіки та неоднорідних структур, подібних до кінцевих автоматів. Дрібномодульні структури також ефективні при реалізації алгоритмів систол (функції, які надзвичайно ефективні за рахунок реалізації масового паралелізму). Ці структури мають певну перевагу при використанні технології традиційного логічного синтезу, яка базується на дрібнодульній архітектурі замовних мікросхем.

Пік інтересу до дрібномодульної архітектури ПЛІС був відмічений у середині 90-х років. Проте з часом переважна більшість представників цього сімейства припинила своє існування, а залишилися лише представники крупномодульної архітектури. У подібній архітектурі кожен логічний блок містить відносно велику кількість логіки. Так, наприклад, логічний блок може містити чотири 4-входових таблиці відповідності, чотири мультиплексори, чотири D-тригери і деяку кількість логіки швидкого переносу.

Для дрібномодульних ПЛІС характерна велика кількість з'єднань усередині блоків і між ними. У міру збільшення модульності пристроїв до середньомодульних і вище кількість з'єднань в блоках зменшується. Це важлива властивість, оскільки внутрішні зв'язки визначають величину переважної більшості затримок, пов'язаних з проходженням сигналів через ПЛІС.

У класифікації ПЛІС є деяка неоднозначність. Деякі компанії створюють крупномодульні пристрої. Ці пристрої містять масиви вузлів, де кожен вузол є складним елементом, що реалізовує алгоритмічні функції, наприклад швидке перетворення Фур'є (ШПФ), або навіть ядро мікропроцесора загального призначення. Суть полягає у тому, що насправді ці пристрої не класифікуються як ПЛІС. З цієї причини архітектуру ПЛІС на основі таблиць відповідності (LUT) часто класифікують як середньомодульну, тим самим, звільняючи термін «крупно модульний» для позначення таких пристроїв, як пристрої на основі вузлів.

Існують два основні способи реалізації програмованих логічних блоків, що використовуються для формування середньомодульних пристроїв на основі мультиплексорів (MUX — від multiplexer) і на основі таблиць відповідності (LUT — від lookup table).

Як прикладу реалізації пристроїв на основі мультиплексорів розглянемо 3-входовую функцію $y = (a \,\&\, b) \mid c$, реалізовану за допомогою блока, що містить тільки мультиплексори (рис. 4.31).

Пристрій може бути запрограмований таким чином, що на кожен його вхід може подаватися логічний 0 або логічна 1, або істинне, або інверсне значення вхідного сигналу (у прикладі *a*, *b* або *c*), що приходить з іншого блока або з входу мікросхеми. Такий підхід дозволяє для кожного блока створювати величезну кількість варіантів конфігурації для виконання різноманітних функцій (*x* на вході центрального мультиплексора на рис. 4.31 означає, що на вхід можна подавати будь-який сигнал — 0 або 1).

Основна концепція таблиць відповідності відносно проста. У таких мікросхемах група вхідних сигналів використовується як індекс (покажчик, або адреса комірки) таблиці відповідності. Вміст цієї таблиці організований таким чином, що комірки, на які вказує кожна вхідна комбінацією, містять необхідне вихідне значення. Припустимо, що вимагається реалізувати функцію $y = (a \,\&\, b) \mid c$ (рис. 4.32).

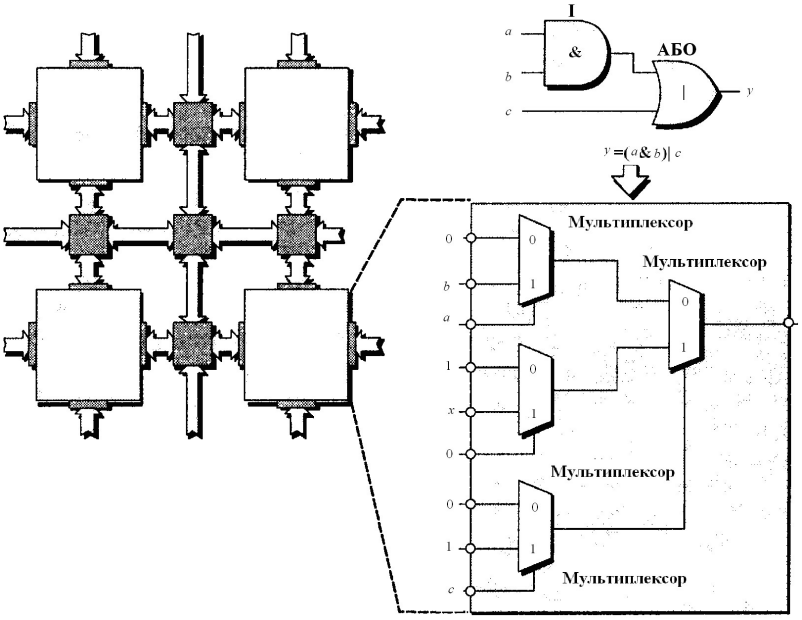


Рис. 4.31. Логічний блок на мультиплексорах

Якщо узяти групу логічних вентилів глибиною в декілька шарів, то таблиця відповідності може бути досить ефективною з погляду використання

ресурсів і затримки розповсюдження сигналу. Тут під «глибиною» мається на увазі кількість логічних елементів між входом і виходом ланцюжка (на рис. 4. 33 глибина складає два шари). Проте недоліком архітектури на таблицях відповідності є те, що якщо з їх допомогою реалізувати невелику функцію, наприклад 2-входовий логічний елемент І, то для цього доведеться використовувати всю таблицю. Результуюча затримка для такої простої функції виявиться достатньо великою.

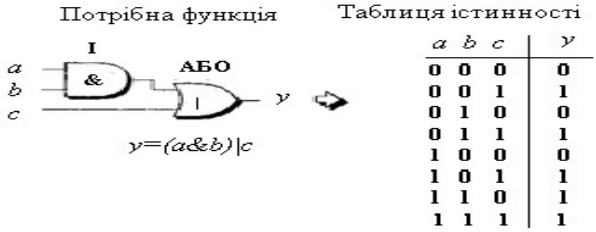


Рис. 4.32. Логічна функція та відповідна до неї таблиця істинності

Для цього треба завантажити 3-входову таблицю відповідними значеннями. Таблиця відповідностей може формуватися з елементів пам'яті статичного ОЗП. Вона може також бути сформована нарощуваними перемичками, ЕСППЗП- або Flash-елементами пам'яті. Для вибору необхідної комірки ОЗП за допомогою каскаду передавальних вентилів використовуються вхідні сигнали, як показано на рис. 4.33. При цьому елементи пам'яті статичного ОЗП для завантаження конфігураційних даних повинні бути поєднані в довгий ланцюжок, але ці ланцюжки на рис. 4.33 не показані з метою його спрощення.

На цій схемі відкритий, або активний, передавальний вентиль пропускає сигнал з входу на вихід. Закритий вентиль відключає свій вихід від провідника, до якого він приєднаний.

Передавальні вентилі, на позначеннях яких зображене невелике «коло», активуються при подачі на управляючий вхід логічного 0. І навпаки, вентилі, на позначеннях яких немає «кола», активуються при подачі на управляючий вхід рівня логічної 1. Виходячи з цього, легко прослідити, як різні вхідні комбінації можуть використовуватися для вибору вмісту необхідного елемента пам'яті.

До появи сучасних систем автоматизованого проектування, коли інженери уручну виготовляли схеми, деякі стверджували, що застосування архітектури на основі мультиплексорів дозволить досягти кращих результатів. Говорять також, що мультиплексорна архітектура має перевагу при розробці управляючої логіки. Проте деякі реалізації цієї архітектури не забезпечують роботу високошвидкісних ланцюжків логічного перенесення.

Їх аналоги на таблицях відповідності залишаються лідерами у всіх додатках, в яких передбачені арифметичні дії.

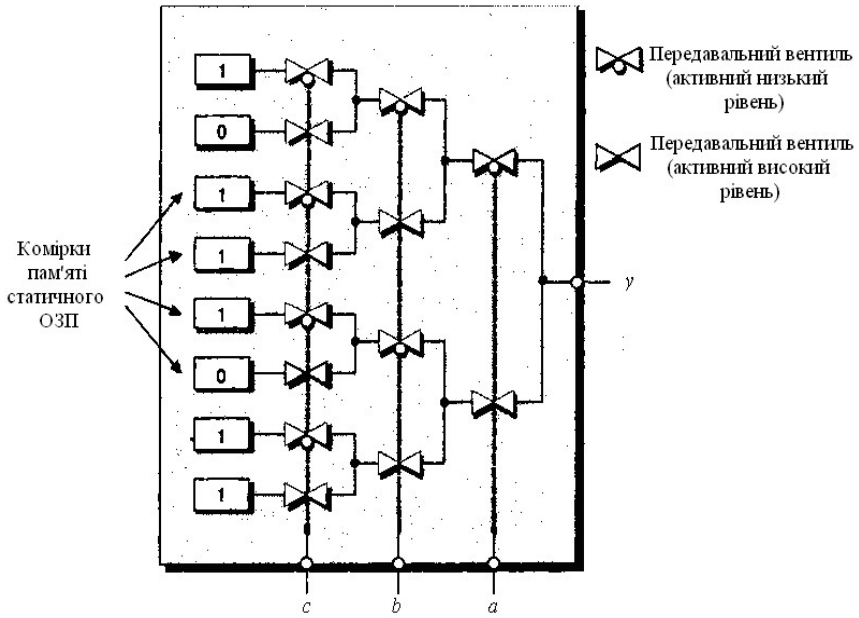


Рис. 4.33. Таблиця істинності на основі передавальних вентилів

Порівняно з таблицями відповідності, при використовуванні мультиплексорної архітектури, що містить суміш мультиплексорів і логічних вентилів, часто вдається здійснити доступ до проміжних значень сигналів, що проходять між логічними елементами і мультиплексорами. В цьому випадку при реалізації невеликих функцій непотрібні (зайві) логічні блоки можуть бути відключені. Отже, мультиплексорна архітектура може мати переваги з погляду продуктивності і використання ресурсів кристала при реалізації пристроїв, що містять велику кількість простих незалежних логічних функцій.

У 90-х роках ПЛІС набули великого поширення у області мереж передачі даних і зв'язку. Обидві сфери мають на увазі передачу великих потоків даних, в яких архітектура на основі таблиць відповідності зарекомендувала себе дуже добре. У міру того як пристрої і їх пропускна спроможність ставали більшими, то і технологія синтезу мікросхем ставала складнішою, а ручне виготовлення схем разом з мультиплексорною

архітектурою поступово зменшувалося. У результаті більшість сучасної архітектури ПЛІС базується на таблицях відповідності.

Важливою особливістю багатовходової таблиці відповідності є те, що вона може реалізувати будь-яку w-входову комбінаційну, або комбінаторну логічну функцію. За допомогою більшої кількості входів можна реалізувати складніші функції, але при цьому слід враховувати, що додавання кожного нового входу супроводжується подвоєнням кількості комірок статичного ОЗП.

Перші ПЛІС базувалися на 3-входових таблицях відповідності. Згодом було проведено аналіз порівняльних експлуатаційних характеристик 3-, 4-, 5- і навіть 6-входових таблиць відповідності. Результати досліджень показали, що оптимальній рівновазі всіх «за і проти» відповідають 4-входові таблиці відповідності.

У минулому деякі постачальники використовували в своїх пристроях таблиці відповідності різних розмірів, наприклад, на три і на чотири входи, оскільки вони обіцяли найбільш оптимальне використання пристрою. Проте це рішення не задовольняло більшість інженерів, які вважали за краще використовувати програми синтезу схем. Тому в даний час вся дійсно вдала архітектура базується тільки на 4-входових таблицях відповідності. Це зовсім не означає, що архітектура на змішаних за розміром таблицях відповідності не з'явиться знов, оскільки програмне забезпечення систем проектування продовжує ускладнюватися

Ядро таблиці відповідності в пристрої на статичному ОЗП використовує для своєї роботи декілька елементів пам'яті. Це дозволяє використовувати деякі цікаві можливості. Крім основного призначення, тобто формування таблиці відповідності, пристрої деяких постачальників дозволяють використовувати комірки, що формують таблицю, як невеликі блоки оперативної пам'яті. Наприклад, 16 елементів пам'яті, що формують 4-входову таблицю, можуть виступати в ролі блока ОЗП 16х1. Такі ділянки пам'яті називаються розподіленим ОЗП, оскільки, по-перше, таблиці відповідності розкидані (розподілені) по всій поверхні кристала, а по-друге, ця назва відрізняє їх від великих блоків ОЗП.

Інший варіант альтернативного використання таблиць заснований на тому, що всі конфігураційні комірки, включаючи і ті, які формують таблицю відповідності, ефективно зв'язані разом в один довгий ланцюжок (рис. 4.34).

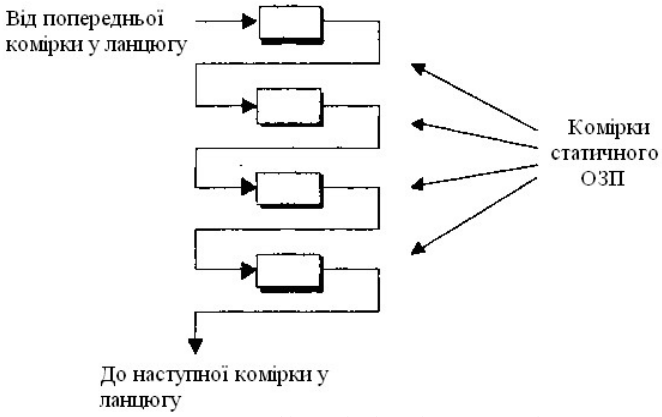


Рис. 4.34. Конфігураційні комірки, зв'язані в ланцюжок

Річ у тому, що в деяких пристроях комірки пам'яті, що формують таблицю відповідності, після програмування можуть розглядатися незалежно від головної структури ланцюжка і використовуватися як регістр зсуву (рис. 4.34). Таким чином, кожну таблицю відповідності можна розглядати як багатофункціональний компонент.

4.5.2. Архітектура базисного модуля, що конфігурується (логічна комірка фірми Xilinx)

Блоки, з яких складається сучасна ПЛІС фірми Xilinx, називаються логічними комірками (logic cell). Крім всього іншого, логічна комірка містить 4-входову таблицю відповідності, яка може працювати як ОЗП 16x1 або як 16-бітовий регістр зсуву, а також мультиплексор і регістр (рис. 4.35) [24].

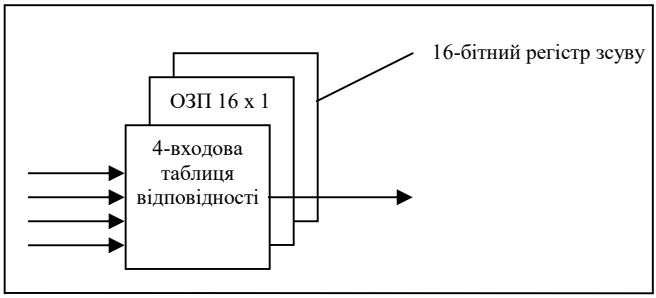


Рис. 4.35. Багатофункціональна таблиця відповідності

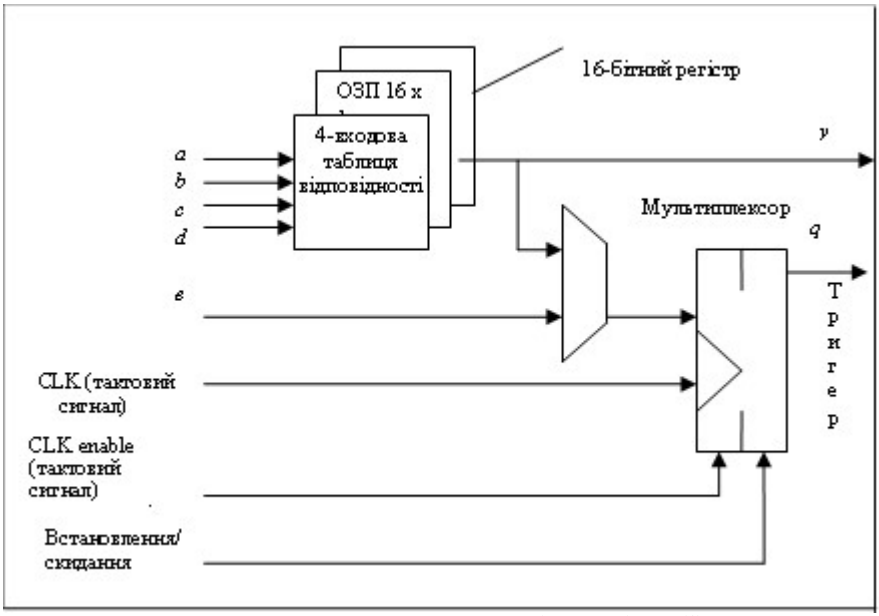


Рис. 4.36. Спрощений вигляд логічної комірки Xilinx

Схема, наведена на рис. 4.35, сильно спрощена, але, проте, вона задовольняє контекст даного матеріалу. Регістр може бути конфігурований для роботи як тригер або як клямка. Полярність тактового сигналу (реакція тригера на фронт або спад синхронізуючого імпульсу) може задаватися програмно, так само, як полярність сигналів «тактовий сигнал дозволено» і «встановлення/скидання» (активний високий або низький рівень).

Окрім таблиць відповідності, мультиплексорів і регістрів, логічні комірки містять невелику кількість інших елементів, включаючи спеціальну логіку швидкого перенесення для використання в арифметичних діях [24].

Блоки, з яких складаються ПЛІС компанії Altera, називаються логічними елементами (logic element). Між логічними комірками Xilinx і логічними елементами Altera існує ряд відмінностей, але в цілому їх концепції дуже схожі.

4.5.3. Конфігурація логічних блоків ПЛІС (комірки, секції, логічні масиви, функціональні модулі)

Наступним ступенем в ієрархії побудови мікросхем програмованої логіки є так звана, за визначенням фірми Xilinx, секція (slice). Під час введення цієї термінології секція містила дві логічні комірки (рис. 4.36) [24].

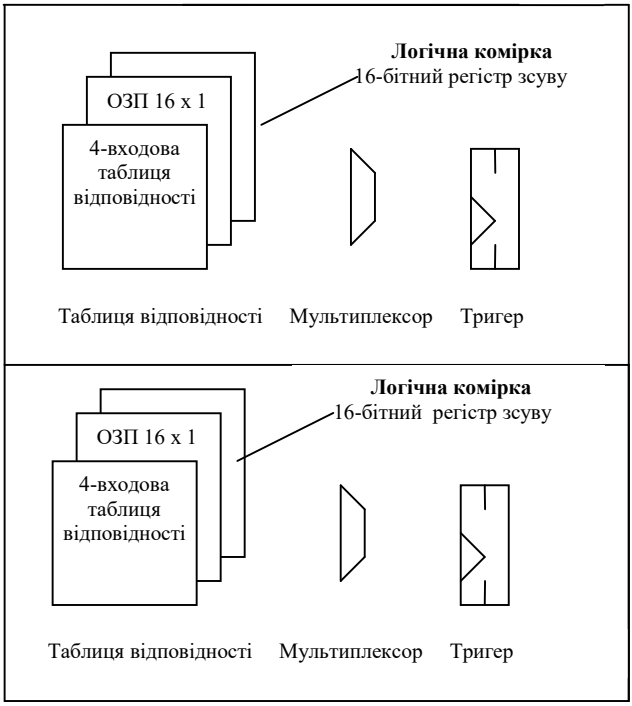


Рис. 4.37. Секція, що містить дві комірки

Компанія Altera і інші постачальники називають цей ступінь якось інакше, використовуючи власні визначення.

На рис. 4.36, для його спрощення, не показані внутрішні зв'язки. Необхідно відзначити, що таблиці відповідності, мультиплексори і регістри кожної логічної комірки мають власні входи і виходи даних. Секція має загальні тактові сигнали, дозвіл тактових сигналів і встановлення/скидання для обох логічних комірок.

Наступний рівень ієрархії, за термінологією компанії Xilinx, має назву «логічний блок КЛБ, що конфігурується» (CLB — configurable logic block). Компанія Altera, у свою чергу, називає його блоком логічних масивів або LAB (LAB — logic array block). Інші постачальники ПЛІС дають їм свої еквівалентні назви [24].

Визначення логічного блока (КЛБ), що конфігурується, змінюється з часом. На початку свого існування КЛБ містили дві 3-входові таблиці відповідності і один регістр. Пізніше в них стали включати дві 4-входові таблиці відповідності і два регістри. Далі кожен блок містив дві або чотири

секції, кожна з яких складалася з двох 4-входових таблиць відповідності і двох регістрів. І цей процес постійно триває.

Використовуючи конфігураційні логічні блоки, як приклад можна відзначити, що деякі ПЛІС фірми Xilinx містять по дві секції в кожному блоці, інші пристрої — по чотири секції в кожному блоці. КЛБ візуально можна подати у вигляді осередків програмованої логіки з програмованими з'єднаннями (рис. 4.37) [24].

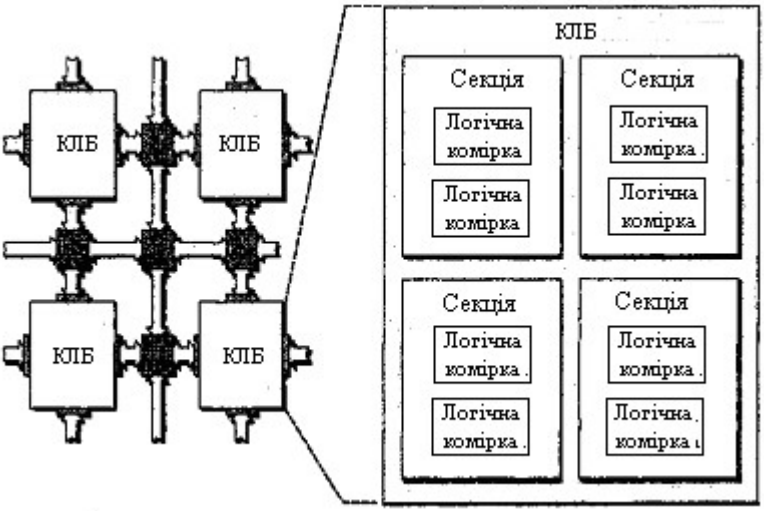


Рис. 4.38. КЛБ, що містить чотири секції

Усередині логічного блока знаходяться швидкі програмовані внутрішні з'єднання. Ці провідники використовуються для з'єднання сусідніх секцій (на рис. 4.37 для простоти викладу вони не показані).

Причина існування такої логіко-блокової ієрархії, тобто логічна комірка → секція з двома логічними комірками → КЛБ з чотирма секціями, полягає у тому, що вона доповнюється еквівалентною ієрархією внутрішніх з'єднань. Тобто, існують швидкі внутрішні з'єднання між логічними комірками усередині секції, потім менш швидкі з'єднання між секціями усередині логічного блока і з'єднання між блоками. Подібна ієрархія відображає покрокове досягнення оптимального компромісу між простотою з'єднання внутрішніх структур і надмірними затримками сигналу на внутрішніх з'єднаннях.

Кожна 4-входова таблиця відповідності може використовуватися як блок ОЗП 16x1. Якщо узяти за основу один чотирисекційний КЛБ (рис. 4.38),

та всі таблиці відповідності усередині цього блока можуть бути конфігуровані для реалізації наступних функцій:

- однопортовий блок ОЗП 16х8 бітів;
- однопортовий блок ОЗП 32х4 біти;
- однопортовий блок ОЗП 64х2 біти;
- однопортовий блок ОЗП 128х1 біт;
- двопортовий блок ОЗП 16х4 біти;
- двопортовий блок ОЗП 32х2 біти;
- двопортовий блок ОЗП 64х 1 біт.

Набір сигналів управління і даних, що розглядаються як єдине ціле, зазвичай називають портом. У однопортовому ОЗП дані записуються та зчитуються з комірки через загальну шину даних. У двопортовому ОЗП дані записуються та зчитуються через різні шини (порти). На практиці в цьому випадку операції читання та запису, як правило, працюють із своїми адресними шинами, що використовуються для визначення необхідної комірки усередині ОЗП, тобто операції читання та запису в двопортовому ОЗП можуть виконуватися одночасно.

Кожна 4-бітова таблиця відповідностей може також використовуватися як 16-бітовий регістр зсуву. Для цього існують спеціальні з'єднання між логічними комірками всередині секції і між секціями, які дозволяють з'єднати останній біт одного регістру зсуву з першим бітом іншого регістру без залучення до цього процесу вихідних сигналів таблиць відповідності.

Останні також можуть використовуватися для перегляду вмісту визначеного біта в 16-бітовому регістрі. Це дозволяє при необхідності з'єднати разом таблиці відповідності усередині одного логічного блока і реалізувати регістр зсуву величиною до 128 біт.

Ключовою особливістю сучасних ПЛІС є те, що вони містять спеціальну логіку і внутрішні з'єднання, необхідні для реалізації схем прискореного переносу. У контексті програмованих логічних блоків, розглянутих вище, слід згадати про те, що кожна логічна комірка містить спеціальну логіку переносу. Ця логіка доповнюється спеціальними внутрішніми з'єднаннями між двома логічними комірками в межах кожної секції, між секціями в рамках кожного логічного блока і між блоками.

Спеціальна логіка швидкого переносу і виділена маршрутизація сприяють виконанню логічних функцій, таких, як лічильники, і арифметичних функцій, таких, як суматори. Можливості схем прискореного переносу сумісно з можливостями інших засобів, аналогічних регістрам зсуву на основі таблиць відповідності, вбудованим помножувачам і іншим блокам, забезпечують необхідний набір засобів для використання ПЛІС в додатках цифрової обробки сигналів (ЦОС).

У процесі реалізації більшості додатків виникає необхідність використовувати елементи пам'яті, тому сучасні ПЛІС містять досить великі блоки вбудованої пам'яті, що називаються блоками вбудованого ОЗП. Залежно від архітектури мікросхеми, ці блоки можуть бути розташовані по периметру кристала, розкидані по його поверхні і відносно ізольовані один від одного або організовані в стовпці, як показано на рис. 4.39.

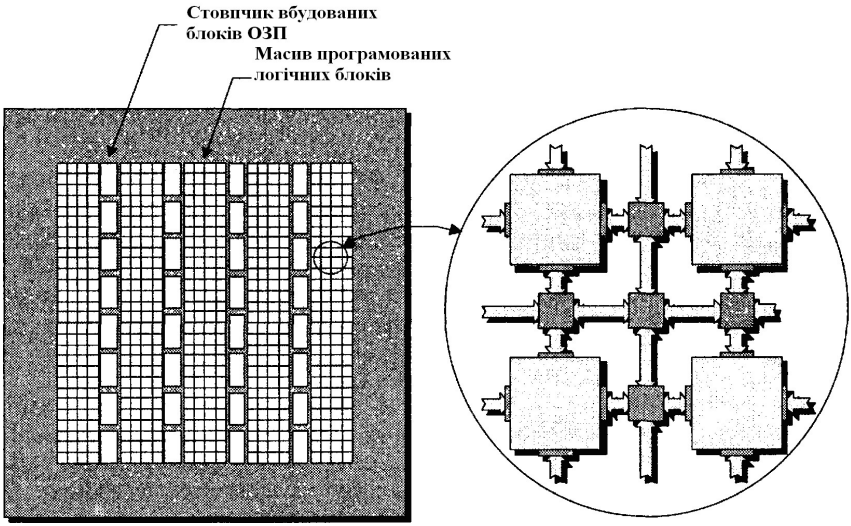


Рис. 4.39. Кристал із стовпцями вбудованих блоків ОЗП

Залежно від пристрою обсяг блоків ОЗП може змінюватися від декількох тисяч до декількох десятків тисяч бітів. Кожна мікросхема може містити від декількох десятків до декількох сотень таких блоків. Таким чином, повна місткість складає від декількох сотень тисяч бітів до декількох мільйонів бітів.

Кожен блок ОЗП може використовуватися або як незалежний запам’ятовуючий пристрій, або як пристрій, що знаходитися у зв'язці з декількома блоками для реалізації масивів пам'яті великого обсягу. Блоки можуть використовуватися для різних цілей, наприклад, як стандартні одно- і двопортові блоки ОЗП, черги FIFO (first-in first-out), кінцеві автомати і так далі.

Деякі типи функцій, наприклад помножувачі, за своєю суттю є досить повільними, якщо їх реалізовувати з допомогою великої кількості програмованих логічних блоків, об’єднаних разом. Оскільки ці функції використовуються в численних додатках, багато ПЛІС містять спеціальні

апаратні блоки множення. Ці блоки зазвичай розташовані в безпосередній близькості від блоків вбудованого ОЗП, оскільки вони часто використовуються разом (рис. 4.40).

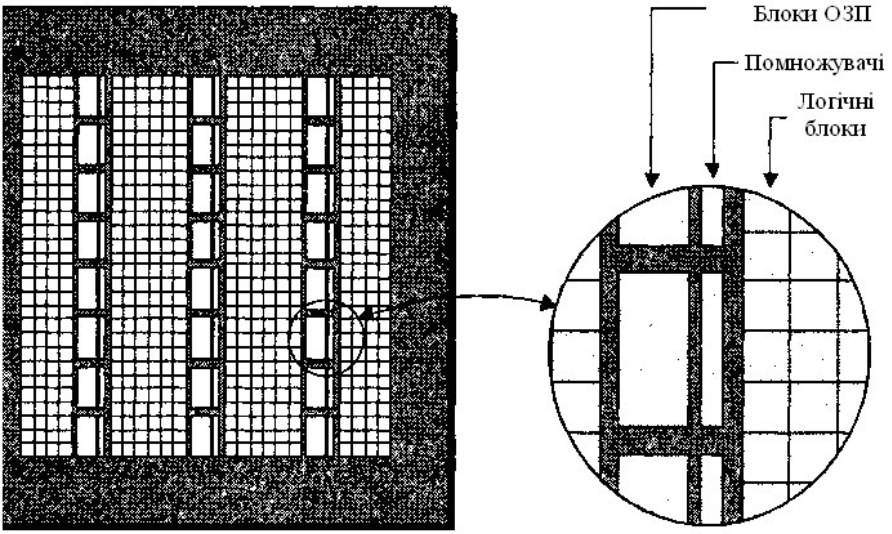


Рис. 4.40. Кристал із вбудованими помножувачами та блоками ОЗП

Деякі виробники ПЛІС також пропонують виділені суматори. У той же час однією з найпоширеніших операцій, що використовуються у додатках цифрової обробки сигналів, є множення з накопиченням {multiply-and-accumulate або MAC) (рис. 4.41). Як підказує назва, ця функція перемножує два числа і підсумовує результат з поточним числом, збереженим в акумуляторі.

При роботі з ПЛІС, яка містить тільки вбудовані помножувачі, для реалізації цієї функції необхідно з'єднати помножувач з суматором, сформованим з декількох програмованих логічних блоків. Результат зберігатиметься або в тригерах логічних блоків, або в блоках вбудованого ОЗП, або в розподіленому ОЗП. Задача спрощується, коли ПЛІС вже містять вбудовані суматори, а окремі їх види навіть підтримують вбудовані помножувачі з накопиченням.

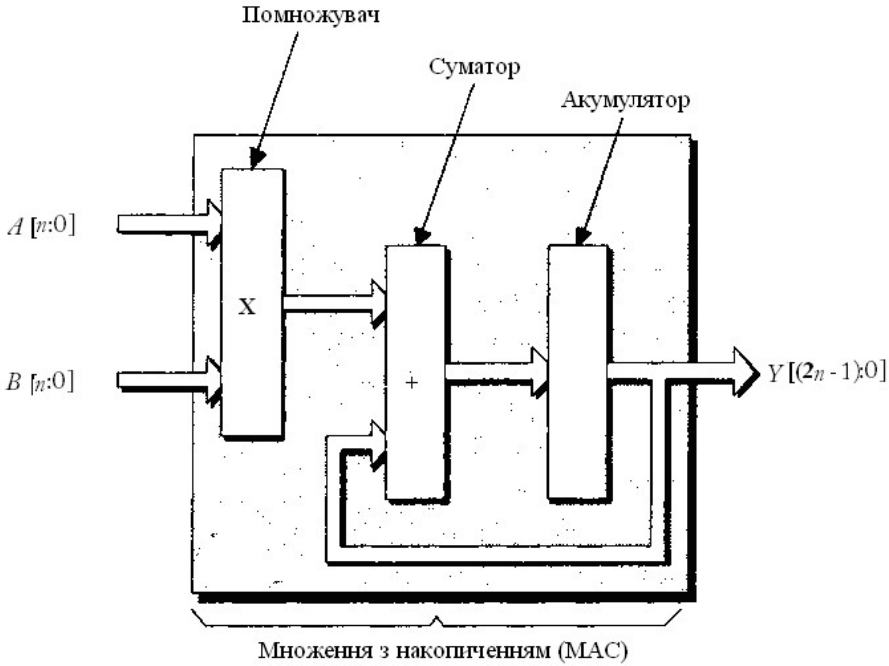


Рис. 4.41. Функції, що формують операцію множення з накопиченням

Важливою особливістю ПЛІС є те, що в них майже всі частини електронного пристрою можуть бути реалізовані або апаратно (з використанням логічних вентилів, регістрів і т. д.), або програмно (у вигляді інструкцій мікропроцесора). Одним з головних критеріїв вибору між апаратною і програмною реалізаціями функції є час, за який ця функція повинна виконувати свої задачі [24]:

• пікосекундна і наносекундна логіка повинна працювати дуже швидко і реалізовуватися апаратно (у структурі ПЛІС);

• мікросекундна логіка є помірно швидкою і може реалізовуватися як апаратно, так і програмно. Це є тип логіки, при якому основна маса часу витрачається на визначення того, яким шляхом піти;

• мілісекундна логіка використовується при реалізації інтерфейсів, таких, як опит стану перемикачів або запалення світлодіодів. Основні зусилля будуть направлені на уповільнення апаратної частини при реалізації цих функцій; для цього, наприклад, використовують величезні лічильники для генерації затримок. Таким чином, часто такі задачі краще реалізовувати в мікропроцесорному коді, оскільки процесор дозволяє знизити швидкість в

порівнянні з апаратною частиною, але при цьому задача може виявитися дуже складною.

Фактично більшість пристроїв в тій або іншій формі використовує мікропроцесори. До останнього часу мікропроцесори були дискретними елементами, що розташовуються на друкарській платі. Останнім часом з'явилися високотехнологічні ПЛІС, що містять один або декілька вбудованих мікропроцесорів, які звичайно називаються мікропроцесорними ядрами. При застосуванні таких мікросхем часто має сенс перекласти всі задачі, що виконуються зовнішнім мікропроцесором, на вбудоване ядро. Такий підхід забезпечує ряд переваг, не останніми з яких будуть зниження вартості, усунення великої кількості доріжок, посадочних місць і виводів на друкарській платі, а також зменшення розміру і ваги друкарської плати.

Апаратні мікропроцесорні ядра виготовляються як окремі визначені блоки. Існує два способи інтеграції таких ядер в ПЛІС. Перший передбачає розташування ядра у вигляді смуги (смуга — stripe) уздовж однієї із сторін головної частини, або головної структури ПЛІС (рис. 4.42).

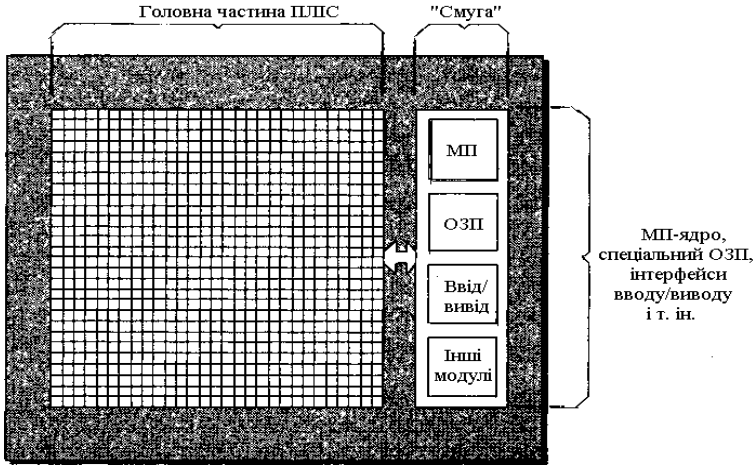


Рис. 4.42. Кристал з вбудованим ядром, що знаходиться за межами головної частини

При такому підході всі компоненти зазвичай формуються на одному кремнієвому кристалі, хоча вони можуть бути виконані і на двох кристалах і розміщені у вигляді багатокристального модуля (Multichip module — MCM). Головна частина ПЛІС також включає вбудовані блоки ОЗП, помножувачі і інші розглянуті вище блоки, які на цьому рисунку не показані з метою його спрощення.

Перевага такої реалізації виявляється в головній частині (структурі) ПЛІС, яка виходить ідентичною для пристроїв з вбудованим і без вбудованого мікропроцесорного ядра. До того ж постачальники ПЛІС можуть зв'язати всі додаткові функції, такі, як пам'ять, пристрої вводу/виводу і інші, в одну смугу для доповнення мікропроцесорного ядра.

Одним з можливих рішень є вбудовування одного або більш мікропроцесорних ядер прямо в головну частину ПЛІС. Існують мікросхеми з одним, двома і навіть чотирма ядрами (рис. 4.43).

Головна частина ПЛІС містить вбудовані блоки ОЗП, помножувачі і інші розглянуті вище блоки, які на цьому рисунку не показані з метою його спрощення.

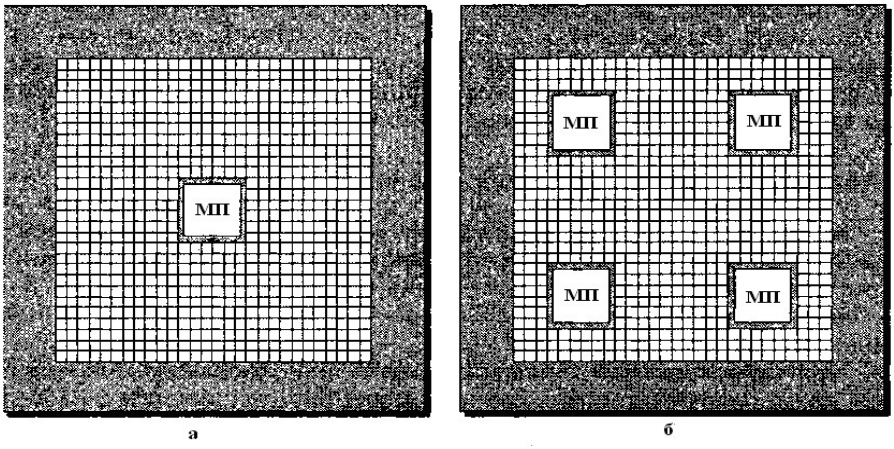


Рис. 4.43. Кристал з ядром, вбудованим до головної частини:
а – з одним вбудованим ядром; б – з чотирма вбудованими ядрами

При використанні цього способу засоби проектування, що застосовуються, повинні враховувати присутність мікропроцесорів у структурі мікросхеми. Пам'ять, що використовується ядром, формується з вбудованих блоків ОЗП, а будь-які функції сполучення реалізуються за допомогою груп програмованих логічних блоків загального призначення. Прихильники цієї схеми стверджують, що розміщення мікропроцесорного ядра в безпосередній близькості до головної частини (структури) ПЛІС забезпечує їй перевагу в швидкості.

Окрім фізичного вбудовування мікропроцесора в структуру кристала, можна конфігурувати групу програмованих логічних блоків для роботи в якості мікропроцесора. Таку групу блоків зазвичай називають програмним

ядром, але точніше вони можуть бути класифіковані як програмні і мікропрограмні, залежно від способу, за допомогою якого функціональність мікропроцесора реалізована логічними блоками. Програмні ядра простіші і повільніші, ніж їх апаратні аналоги. Проте у них є одна перевага — при необхідності можна реалізувати ядро або декілька ядер в тому обсязі, якого можна досягти поки не будуть вичерпані всі ресурси у вигляді програмованих логічних блоків.

Швидкість роботи програмного ядра звичайно складає 30...50 % від швидкості апаратного ядра.

Всі синхронні елементи всередині ПЛІС, наприклад регістри всередині програмованого логічного блока, конфігуровані для роботи у вигляді тригерів, необхідно синхронізувати за допомогою тактового сигналу. Тактові сигнали звичайно виробляються за межами мікросхеми і надходять до неї через спеціальні входи синхронізації, а потім розподіляються через спеціальні пристрої і подаються на відповідні регістри.

Розглянемо спрощене зображення дерева синхронізації (рис. 4.44, програмовані логічні блоки не показані).

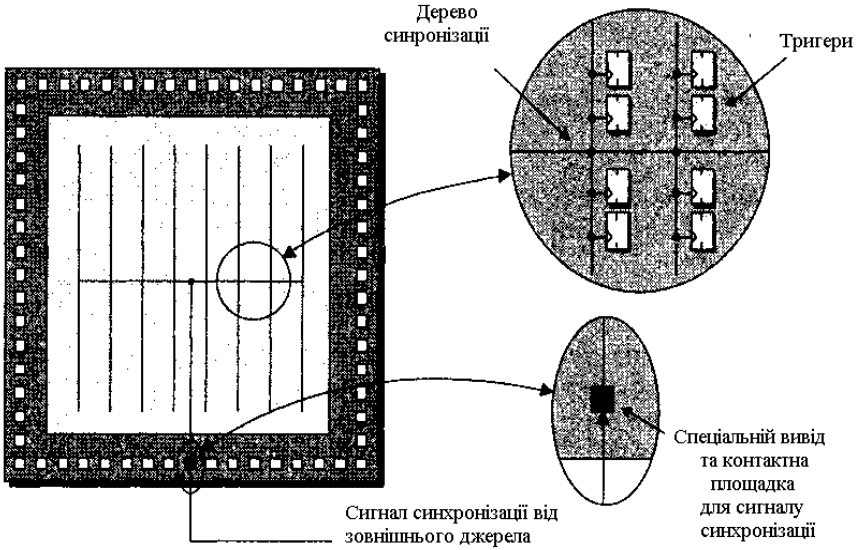


Рис. 4. 44. Просте дерево синхронізації

Назва «дерево синхронізації» виникла тому, що головний синхроімпульс розгалужується подібно до гілок дерева, при цьому тригери можуть розглядатися як «листя» на кінцях віток. Така структура вселяє упевненість у тому, що всі тригери отримають свої тактові сигнали одночасно, на скільки це можливо. Якби тактові сигнали розповсюджувалися

по одному довгому провіднику, синхронізуючи всі тригери по черзі, то тригер, розташований ближче до виводу синхронізації мікросхеми, отримав би синхроімпульс набагато раніше, ніж останні тригери в цьому ланцюжку. Подібна ситуація називається фазовим зсувом, і вона породжує цілий ряд проблем. Навіть при використанні дерева синхронізації може виникнути деякий зсув фаз між регістрами, що знаходяться на одній гілці, а також між гілками.

Дерево синхронізації реалізується за допомогою спеціальних провідників, які відокремлені від внутрішніх з'єднань загального призначення. Принцип дії «дерева синхронізації», розглянутий вище, насправді сильно спрощений. На практиці в мікросхемах існує множина виводів синхронізації (виводи синхронізації, що не використовуються, можуть бути використані як виводи загального призначення), а усередині пристрою є множинні домени синхронізації (дерева синхронізації).

Вивід синхронізації мікросхеми може бути підключений до дерева синхронізації. Проте, як правило, цей вивід підключають не напряму до дерева синхронізації, а до входу пристрою (або блока) управління синхронізацією, що має назву «диспетчер синхронізації», який генерує дочірні тактові сигнали (рис. 4.45).

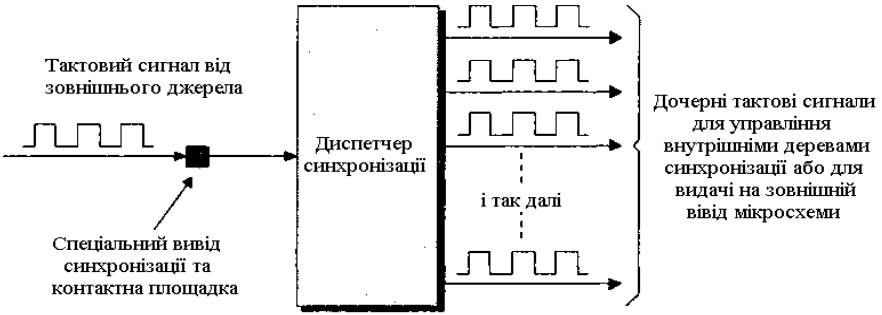


Рис. 4. 45. Диспетчер синхронізації, що генерує дочірні тактові сигнали

Дочірні тактові сигнали можуть використовуватися для управління внутрішніми деревами (доменами) синхронізації або для видачі сигналів на зовнішні виводи мікросхеми, які, у свою чергу, можна використовувати для синхронізації інших пристроїв, розташованих на друкарській платі. Кожне сімейство мікросхем ПЛІС має в своєму розпорядженні власний тип диспетчера синхронізації, і в одному пристрої можуть знаходитися безліч модулів диспетчера синхронізації. Різні диспетчери синхронізації можуть підтримувати всі або тільки деякі з наступних властивостей: усунення флуктуації, частотний синтез, фазовий зсув, автокорекція фазового зсуву.

Усунення флуктуації. Для спрощення прикладу припустимо, що сигнал синхронізації має частоту 1 Мгц. На практиці, звичайно ж, ця частота в багато-багато разів вища. У ідеальному випадку кожен фронт тактового сигналу від зовнішньої системи синхронізації приходить рівно через одну мільйонну частку секунди після попереднього. Проте в реальному житті фронт імпульсу може прийти небагато раніше або небагато пізніше.

Для візуалізації цього ефекту, що називається флуктуацією, зобразимо фронти імпульсів один під іншим, щоб отримати їх суперпозицію; в результаті може вийти «змазаний» тактовий сигнал (рис. 4.46).

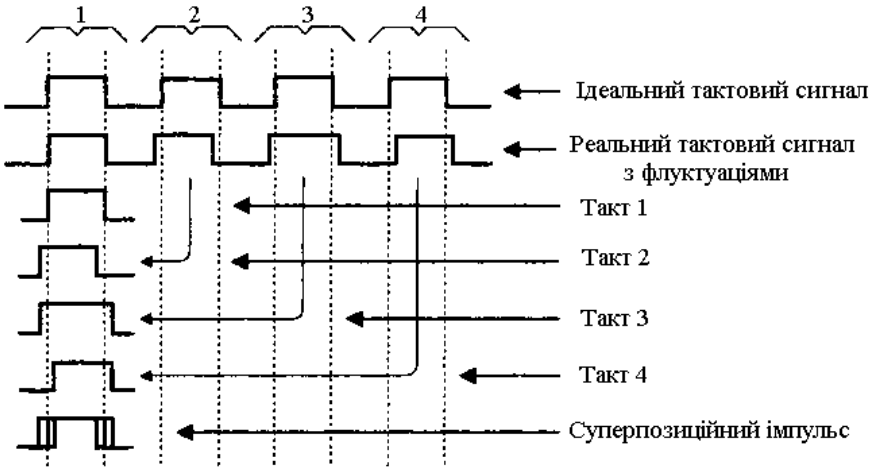


Рис. 4. 46. Флуктуація як результат «змазування» тактового сигналу

Диспетчер синхронізації ПЛІС може використовуватися для виявлення і корекції подібних флуктуації й може забезпечувати очищення дочірніх тактових сигналів для їх використання усередині пристрою .

Частотний синтез. Може трапитися так, що частота тактового сигналу, що надходить на ПЛІС від зовнішнього джерела синхронізації, не відповідає частоті, необхідній для вирішення поставлених задач. У цьому випадку диспетчер синхронізації може генерувати дочірні тактові сигнали, частота яких є похідною величиною від початкового сигналу і виходить шляхом його ділення або множення.

Як простий приклад розглянемо три дочірні тактові сигнали: 1) перший з частотою, еквівалентною початковій частоті тактового сигналу; 2) з частотою, що дорівнює частоті початкової послідовності, помноженої на два; 3) з частотою, що дорівнює половині частоти початкового сигналу (рис. 4.47).

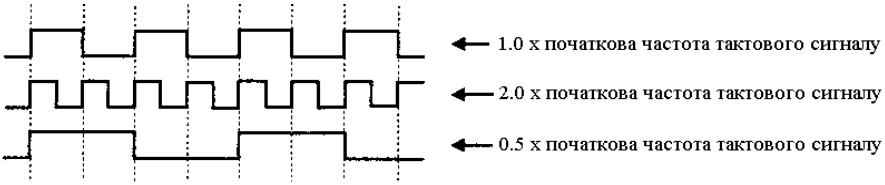


Рис. 4. 47. Використання диспетчера синхронізації для частотного синтезу

На рис. 4.47 наведено дуже простий приклад. На практиці ПЛІС можуть синтезувати всі типи внутрішніх тактових сигналів, наприклад чотири п'ятих від частоти первинної послідовності тактового сигналу.

Фазовий зсув. Деякі вироби вимагають використання тактових сигналів, фаза в яких зсунута (затримана) по відношенню один до одного. Деякі диспетчери синхронізації дозволяють вибирати загальні фіксовані значення фазових зсувів, наприклад 120° і 240° (для трифазної системи синхронізації) або 90°, 180° і 270° (для чотирифазної системі синхронізації). Інші диспетчери дозволяють конфігурувати точне значення фазового зсуву на вимогу користувача для кожного дочірнього тактового сигналу.

Припустимо, що ми отримуємо чотири внутрішні тактові сигнали з головної послідовності тактових сигналів, де перший перебуває у фазі з початковим сигналом, другий зсунутий по фазі на 90°, третій – на 180° і т.ін. (рис. 4.48).

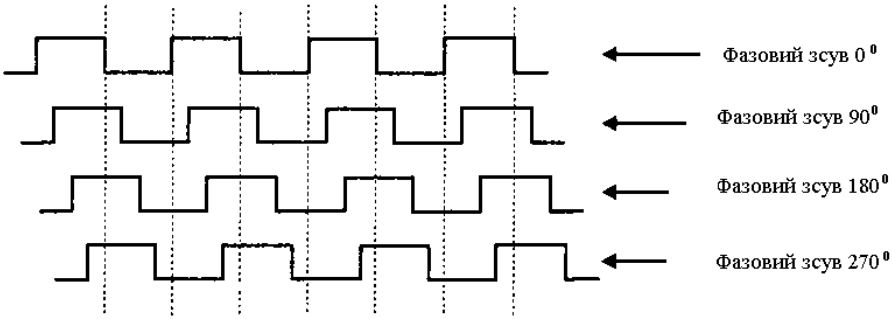


Рис. 4.48. Використання диспетчера синхронізації для фазового зсуву дочірніх тактових сигналів

Автокорекція зсуву. З метою спрощення припустимо, що йдеться про дочірні тактові сигнали, сформовані на тій же частоті і з тією ж фазою, що і головний тактовий сигнал, що приходить на вхід ПЛІС. Проте диспетчер синхронізації, за визначенням, додаватиме до сигналів деяку затримку. Крім того, ще більші затримки додають логічні вентилі і внутрішні з'єднання, що

використовуються в розподілі тактових сигналів. Якщо не вжити корекцію, то дочірні тактові сигнали відставатимуть від вхідних тактових сигналів на деяку величину. Різниця між фазами двох сигналів називається фазовим зсувом.

Залежно від того, як головні і дочірні тактові сигнали використовуються ПЛІС і рештою елементів друкарської плати, зсуви фаз можуть стати причиною різних проблем. Тому диспетчер синхронізації може містити спеціальний вхід для подачі на нього дочірнього тактового сигналу. В цьому випадку диспетчер синхронізації порівнює два сигнали і точно додає певну затримку до дочірніх сигналів, достатню для вирівнювання їх з початковим тактовим сигналом (рис. 4.49).

У такий спосіб буде очищений тільки первинний, тобто з нульовим фазовим зсувом, дочірній сигнал, а всі інші дочірні сигнали будуть фазовані вже щодо цього сигналу.

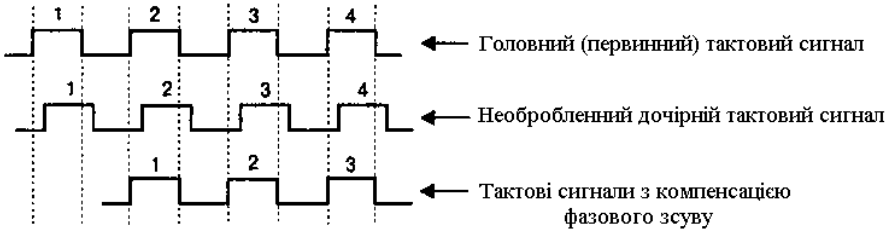


Рис. 4.49. Компенсація фазового зсуву відносно первинного тактового сигналу

Деякі диспетчери синхронізації ПЛІС працюють за принципом фазового автопідстроювання частоти (ФАПЧ), інші – за принципом цифрового автопідстроювання по затримці (digital delay-locked loop — DLL). Системи ФАПЧ можуть бути реалізовані аналоговим або цифровим способом, а системи цифрового автопідстроювання по затримці за своєю природою можуть бути тільки цифровими. Прихильники ФАПЧ стверджують, що ці системи мають переваги в точності, стабільності і споживаній потужності, в нечутливості до шумів і флуктуацій.

Сучасні ПЛІС можуть містити 1000 і більш виводів, які розташовуються по всьому корпусу мікросхеми. Вони також підходять до кристала усередині корпусу мікросхеми. Всередину корпусу кристал встановлюють в перевернеому вигляді. Це дозволяє під'єднувати загальний дріт, провідники живлення, синхронізації і сигналів вводу/виводу до будь-якої точки на його поверхні. Припустимо, що всі виводи на кристалі розташовані по його контуру, як і було насправді багато років тому.

Якщо подивитися на електронний пристрій з погляду інженерів, що розробляють друкарську плату, залежно від того, що вони роблять, від типів

використовуваних пристроїв, від навколишнього середовища, в якому працюватиме плата і так далі, розробники вибиратимуть певний стандарт передачі даних. (У цьому контексті поняття «стандарт» відноситься до електричних параметрів сигналів, таких, як рівень логічного 0 або логічної 1 у вольтах.)

Проблема у тому, що існує велика різноманітність стандартів, і було б важко створювати спеціальні ПЛІС для підтримки кожного варіанта. З цієї причини ввід/вивід загального призначення ПЛІС може бути конфігурований для прийому і передачі сигналів, відповідних до будь-якого стандарту. Ці сигнали вводу/виводу розділяються на декілька банків. Вважатимемо, що існує вісім банків, пронумерованих від 0 до 7 (рис. 4.50).

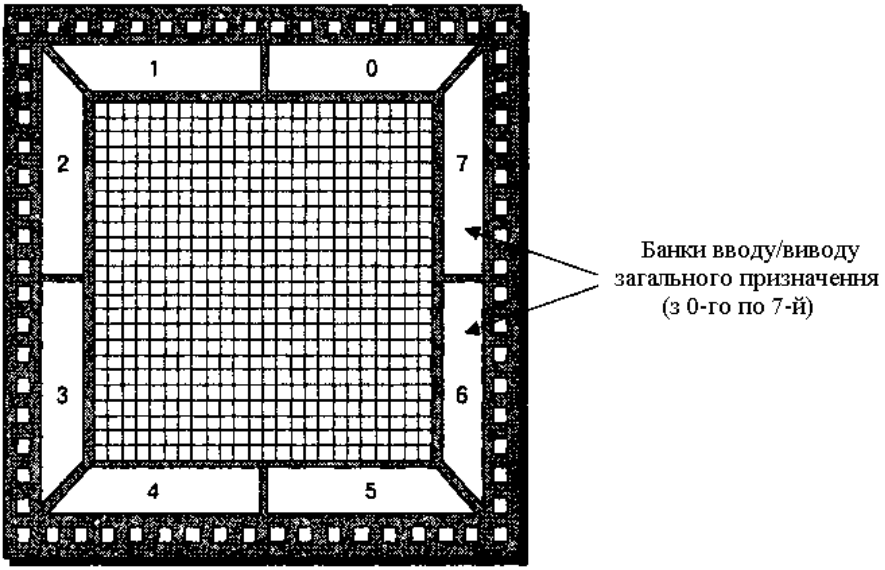


Рис. 4. 50. Вид на кристал з банками вводу/виводу

Кожен банк може бути індивідуально конфігурований для підтримки визначених стандартів вводу/виводу. Таким чином, ПЛІС може працювати з пристроями, використовуючи численні стандарти вводу/виводу. Ці мікросхеми також можуть служити інтерфейсом між різними стандартами вводу/виводу, і здійснювати зв'язок між різними протоколами, які можуть базуватися на різних електричних стандартах.

Сигнали в сучасній друкарській платі часто мають швидкий час перемикання. Мається на увазі час, який необхідний сигналу для перемикання з одного логічного рівня на інший. Щоб запобігти

віддзеркаленню сигналів, що приводить до виникнення пульсацій, до виводів мікросхеми, тобто до входу або виходу, необхідно підключити відповідні узгоджувальні резистори, тобто термінатори.

У минулому ці резистори застосовувалися як окремі компоненти, які розміщувалися на друкарській платі за межами ПЛІС. Проте такий підхід ставав все більш проблематичним, оскільки кількість виводів у мікросхеми збільшувалася, а крок, тобто відстань між ними, зменшувався. У зв'язку з цим сучасні ПЛІС дозволяють використовувати внутрішні узгоджувальні резистори, або внутрішні термінатори, опір яких може задаватися користувачем, для узгодження різного устаткування на друкарській платі і різних стандартів вводу/виводу.

У минулі часи, приблизно до 1995 року, більшість цифрових мікросхем використовувала напругу землі 0 В і напругу живлення +5 В. Крім того, сигнали вводу/виводу перемикалися між рівнями 0 В (логічний 0) і +5 В (логічна 1).

З часом розміри структур на кремнієвих кристалах ставали меншими, оскільки транзистори менших розмірів були дешевшими, а також більш швидкими і споживали менше енергії. Проте для цього вимагалося знизити напруги живлення. Зміна напруги живлення відображена у табл.. 4.2 [24].

Таблиця 4.2 – Залежність напруги живлення від технологічного процесу

Рік	Джерело живлення (напруга ядра) [В]	Технологічний процес [нм]
1998	3,3	350
1999	2,5	250
2000	1,8	180
2001	1,5	150
2003	1,2	130

Джерела живлення, що зазначені у цій таблиці, використовується для живлення внутрішньої логіки ПЛІС, тому ця напруга називається напругою ядра (насправді для підключення живлення і «землі» використовуються декілька виводів мікросхеми). Різні стандарти вводу/виводу використовують сигнали з напругами, рівні яких значно відрізняються від напруги ядра, тому кожен банк вводу/виводу даних може мати свій додатковий вивід живлення.

Починаючи з 350 нм процесу, напруга ядра змінювалася прямо пропорційно технологічному процесу. Проте існують фізичні причини, що не дозволяють напрузі опуститися значно нижче 1 вольт (ці причини засновані на технологічних аспектах, таких, як вхідний поріг перемикання транзисторів і перепад напруг).

4.6. Методи та засоби програмування архітектури ПЛІС

У кожного виробника ПЛІС своя термінологія, свої методи, засоби роботи. Механізми програмування ПЛІС повинні істотно змінюватися від сімейства до сімейства. Тому подальші міркування будуть присвячені виключно загальним питанням програмування мікросхем.

4.6.1. Конфігураційні файли

Інструменти та підходи, які можуть бути використані для проектування принципових схем і для реалізації ПЛІС-систем, дозволяють розробникам створити конфігураційний, або бітовий, файл, який містить інформацію, призначену для завантаження в мікросхему, тобто для її програмування з метою виконання певних функцій.

Якщо мікросхема використовує для зберігання своєї конфігурації комірки статичного ОЗП, та конфігураційний файл містить деяку сукупність конфігураційних даних, або біти, які використовуються для визначення стану елементів програмованої логіки і конфігураційних команд, тобто інструкцій, які вказують пристрою, що йому необхідно робити з конфігураційними даними. При завантаженні конфігураційного файла в пристрій інформацію, що передається, називають конфігураційним двійковим потоком [24].

Пристрої на елементах пам'яті ЕСППЗП або Flash-пам'яті програмуються аналогічно пристроям на комірках статичного ОЗП. На відміну від них, в пристроях, заснованих на методі нарощуваних перемичок, конфігураційний файл містить тільки конфігураційні дані, які використовуватимуться для нарощування перемичок.

4.6.2. Конфігураційні комірки

Основна концепція програмування ПЛІС відносно проста і є завантаженням в пристрій конфігураційного файла. Оскільки це досить-таки складний процес, то щоб він був зрозумілий, почнемо з основ. Спершу розглянемо елементарний пристрій, що складається з масиву дуже простих програмованих логічних блоків, оточених програмованими внутрішніми з'єднаннями (рис. 4.29) [24].

Всі можливості програмованих пристроїв реалізуються за допомогою спеціальних конфігураційних комірок. Більшість ПЛІС використовує комірки статичного ОЗП, інші використовують ЕСППЗП або осередки Flash-пам'яті, треті базуються на нарощуваних перемичках.

Незалежно від базової технології, внутрішні з'єднання пристрою містять велику кількість зв'язуючих (з'єднуючих) комірок, які можна

використовувати для його конфігурації, щоб з'єднати входи і виходи мікросхеми з програмованими логічними блоками, а також блоки між собою. Всі блоки воду/виводу, які не показані на рис. 4.29, мають в своєму розпорядженні декілька з'єднуючих комірок, які використовуються для їх конфігурації відповідно до певних стандартів інтерфейсів вводу/виводу або інших параметрів. Припустимо, що кожен програмований логічний блок містить всього лише 4-входову таблицю відповідності, мультиплексор і регістр (рис. 4.51).

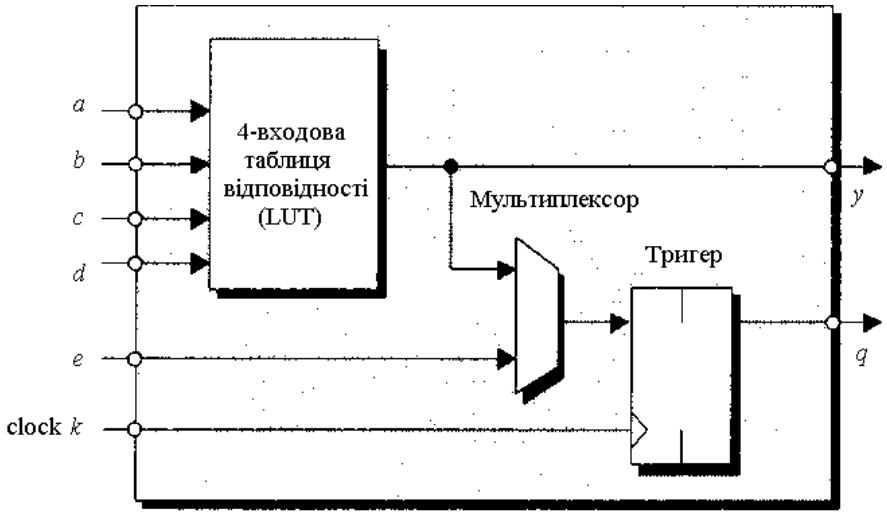


Рис. 4. 51. Простий логічний блок, що програмується

Мультиплексор потребує з'єднуючої конфігураційної комірки, щоб визначити вхід, сигнал з якого передаватиметься на його вихід. Регістр потребує з'єднуючих конфігураційних комірок, щоб визначити, чи буде він:

• виступати в ролі тригера (як показано на рис. 4.51) або в ролі клямки;

• перемикатися по фронту або по спаду синхроімпульсу (при роботі в режимі тригера) або перемикатися високим або низьким рівнем сигналу (у режимі клямки);

• ініціалізуватися логічним 0 або логічної 1.

У свою чергу, 4-входова таблиця відповідності містить 16 конфігураційних комірок.

4.6.3. ПЛІС на нарощуваних перемичках

У мікросхемах на нарощуваних перемичках конфігураційні комірки, як правило, розподілені по всій поверхні пристрою по певних ключових напрямах. Для програмування мікросхема поміщається в спеціальний програматор, в який з управляючого комп'ютера завантажується конфігураційний або бітовий файл. Цей файл використовується програматором як керівництво по видачі імпульсів щодо високої напруги і великого струму на певні виводи мікросхеми, унаслідок чого відбувається почергове нарощування перемичок.

Щоб краще зрозуміти процес програмування, уявимо, що кожна нарощувана перемичка має віртуальні координати x-у на поверхні кристала і ці значення x-у подаються у вигляді чисел. Далі уявимо, що одна група контактів вводу/виводу мікросхеми призначена для подання значення x, а інша група контактів – для подання значення у. У реальності все виглядає набагато складніше, але цей примітивний приклад дозволяє зрозуміти суть процесу [24].

Після нарощування всіх перемичок мікросхема витягується з програматора і поміщається на друкарську плату. Звичайно ж, пристрої на нарощуваних перемичках є одноразово програмованими, і з початком процесу програмування що-небудь поміняти в конфігурації пристрою буде неможливо.

4.6.4. ПЛІС на комірках статичного ОЗП

Розглянемо ПЛІС-структуру на основі комірок статичного ОЗП. Ці пристрої є енергозалежними. Це означає, що вони програмуються всередині системи, тобто на друкарській платі, причому програмуються кожного разу після включення системи.

Всі конфігураційні комірки статичного ОЗП можна представити у вигляді одного довгого регістра зсуву. Вважатимемо, що початок і закінчення, тобто вхід і вихід цього регістрового ланцюжка напряму підключені до зовнішніх контактів мікросхеми. Проте при цьому слід мати на увазі, що це можливо тільки в тому випадку, коли програмування здійснюється за допомогою комунікаційного порту в режимі послідовного завантаження. При послідовному завантаженні ПЛІС може знаходитися у двох режимах: «ведучій» (master) та «ведений» (slave) [24].

Також слід мати на увазі, що вихідні контакти і вихідні сигнали конфігураційних даних використовуються тільки у тому випадку, коли декілька ПЛІС конфігуровані для роботи в режимі каскадування, тобто

послідовного опиту, або якщо з тих або інших причин від пристрою вимагається прочитати конфігураційні дані.

Пристрої на основі елементів пам'яті Flash (і ЕСППЗП) звичайно програмуються таким же способом, яким і мікросхеми на основі статичного ОЗП, але при цьому є незалежними. Ця властивість дозволяє Flash-пристроям зберігати конфігураційну інформацію при відключенні живлення системи. Іншими словами, такі пристрої не треба перепрограмувати при включенні системи, хоча при необхідності така процедура може бути виконана. ПЛІС на основі Flash-елементів можуть програмуватися внутрішньосистемно, тобто на друкарській платі, або позасистемно за допомогою окремого програматора.

Найпростіший спосіб зрозуміти внутрішню структуру програмованих комірок статичного ОЗП полягає в поданні їх у вигляді довгого регістру зсуву. Так було б у тому випадку, коли б кожна комірка була реалізована у вигляді тригера, а всі тригери були з'єднані в ланцюжок та управлялися загальним синхросигналом.

Проблема полягає у тому, що ПЛІС може містити величезну кількість конфігураційних комірок. Вже до 2003-го року високотехнологічні пристрої могли містити 25 мільйонів таких комірок. Тут необхідно відмітити, що для реалізації тригера потрібно вісім транзисторів, а для клямки – всього лише чотири. Тому конфігураційні комірки в пристроях на статичному ОЗП виконуються на клямках. Так, в пристрої з 25 мільйонами конфігураційних комірок такий підхід дозволяє заощадити 100 мільйонів транзисторів.

Проблема криється у тому, що не можна створити такий великий регістр зсуву з клямок. Постачальники ПЛІС обходять цю проблему за допомогою групи тригерів, скажімо, з 1024-х, що об'єднані загальним синхросигналом і що конфігуруються як класичний регістр зсуву. Таку групу називають фреймом.

Усі 25 мільйонів конфігураційних комірок-клямок в пристрої з нашого прикладу також розбивалися на фрейми, при цьому довжина кожного дорівнювала довжині спеціального фрейму тригера. Зовні процес був простим завантаженням 25 мільйонів бітів конфігураційних даних в пристрій. Проте усередині пристрою, як тільки перші 1024 біти послідовно завантажувалися у фрейм тригера, спеціальна внутрішня схема автоматично паралельно копіювала ці дані в перший фрейм клямок. Потім наступні 1024 біти завантажувалися у фрейм тригера і автоматично паралельно копіювалися в другий фрейм клямок і так далі до повного заповнення пристрою. При читанні даних з пристрою процес перебігав в зворотному порядку.

Програмування ПЛІС може займати значний час. Якщо розглядати високотехнологічну мікросхему, що містить 25 мільйонів комірок статичного

ОЗП, конфігурацію такого пристрою в послідовному режимі з тактовою частотою 25 Мгц займуть одну секунду.

4.6.5. Програмування вбудованих блоків розподіленого ОЗП та інших ОЗП

Якщо ПЛІС містить великі вбудовані блоки ОЗП, то ядра цих блоків виконуються з клямок на основі елементів статичної пам'яті. Кожна клямка, у свою чергу, є конфігураційною коміркою, яка формує частину уявного регістрового ланцюжка, який розглядався в попередньому розділі.

Інтерес представляє той факт, що кожна 4-входова таблиця відповідності (рис. 4.49) може конфігуруватися для роботи як таблиця відповідності, або як невеликий блок (16x1) розподіленого ОЗП, або як 16-бітовий регістр зсуву. Всі ці конфігурації використовують для своєї роботи групу з 16 клямок на основі статичного ОЗП, причому кожна клямка є конфігураційною коміркою з складу розглянутого регістрового ланцюжка.

Особливість схеми полягає у тому, що в цьому випадку використовується концепція ємнісних клямок, які за багатьма параметрами перевершують класичні клямки. Дуже схожим способом розробники робили зовнішні тригери з дискретних транзисторів, резисторів і конденсаторів на початку 60-х років минулого століття [24].

4.6.6. Мультипрограмування конфігураційних ланцюжків

Оскільки кількість конфігураційних осередків може досягати декількох десятків мільйонів, ланцюжок дійсно може виявитися дуже довгим. У деяких мікросхемах ланцюжок розбивається на частини, і окремі ділянки цього ланцюжка підключаються до конфігураційного порту. Такий підхід дозволяє конфігурувати окремі частини пристрою і спрощує реалізацію різних концепцій, таких, як модульне і покрокове проектування

Регістри програмованих логічних блоків сполучені з конфігураційними осередками, в яких міститься їх початкове значення: логічний 0 або логічна 1. Кожне сімейство ПЛІС підтримує деякий механізм, наприклад вивід ініціалізації, який при активації дає команду регістрам повернутися до початкових значень. Такий механізм не ініціалізує вбудовані блоки пам'яті або розподілене ОЗП [24].

4.6.7. Конфігураційний порт

Перші ПЛІС використовували інструмент, що називається конфігураційним портом. Навіть сьогодні, коли доступні більш складні

методи, подібні до JTAG-інтерфейсу, конфігураційний порт все ще знаходить широке застосування, оскільки є досить простим інструментом.

Почнемо розгляд з невеликої групи спеціальних контактів режиму конфігурації, які використовуються для установлення режиму конфігурації пристрою. У перших ПЛІС для цих цілей використовувалися тільки два виводи, які дозволяли задавати чотири режими конфігурації (табл. 4.3) [24].

Таблиця 4.3 – Чотири первинних режими ініціалізації

Виводи режиму установлення	Найменування режиму
00	Послідовне завантаження, ПЛІС у режимі «ведучий»
01	Послідовне завантаження, ПЛІС у режимі «ведений»
10	Паралельне завантаження, ПЛІС у режимі «ведучий»
11	Паралельне завантаження, ПЛІС у режимі «ведений»

Слід мати на увазі, що назви режимів, а також відповідність коду на виводах установлення режиму до назви режиму приведені тільки як приклад. Справжні коди і назви режимів у кожного постачальника свої.

Виводи, або контакти, установлення режиму звичайно підключаються до необхідних значень логічного 0 або логічної 1 на друкарській платі. Ці виводи можуть підключатися до інших логічних пристроїв, які дозволяють змінювати режим програмування, але така схема рідко зустрічається на практиці.

Окрім виводів установлення режиму, використовується додатковий вивід для інформування ПЛІС про початок процесу конфігурації, і ще один вивід для видачі сигналу, що свідчить про закінчення процесу конфігурації. Існують також сигнали для визначення помилок, що виникають у ході конфігурації. Крім конфігурації ПЛІС при включенні системи, при необхідності вона може бути реініціалізована і повернена до первинних конфігураційних даних.

Конфігураційний порт також використовує додаткові виводи мікросхеми для управління завантаженням даних і для їх вводу. Кількість цих виводів залежить від вибраного режиму конфігурації. Важливе значення має той факт, що при завершенні конфігурації більшість цих виводів може використовуватися як входи/виходи загального призначення.

4.6.8. Послідовне завантаження, ПЛІС у режимі «ведучий»

Цей режим є найпростішим зі всіх режимів програмування. Раніше для роботи в цьому режимі використовувалися зовнішні мікросхеми ППЗП. Згодом їх замінили СППЗП, потім ЕСППЗП а тепер, як правило, використовуються пристрої на осередках flash-пам'яті. Ці компоненти

спеціальної пам'яті містять один вивід для виходу даних, який під'єднується до контакту вводу конфігураційних даних ПЛІС (рис. 4.52) [24].

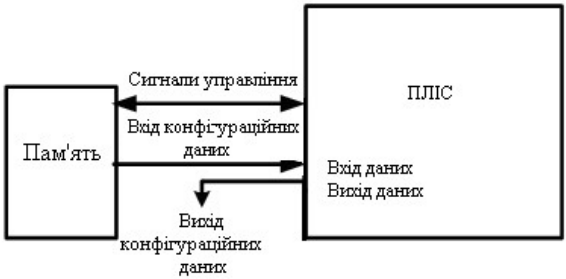


Рис. 4.52. Послідовне завантаження, ПЛІС у режимі «ведучий»

ПЛІС також використовує декілька сигналів для управління зовнішньою пам'яттю. До них відноситься сигнал скидання, який видається ПЛІС при готовності нею почати читання даних, а також синхросигнал, призначений для синхронізації конфігураційних даних.

У цьому режимі ПЛІС не потребує зовнішньої пам'яті з послідовною адресацією. При такому режимі використовуються прості імпульси сигналу скидання для передачі сигналу готовності почати читання даних з початку конфігураційної послідовності, а потім застосовується послідовність синхроімпульсів для синхронізації видачі конфігураційних даних з пам'яті.

При необхідності вихідний сигнал з ПЛІС може використовуватися для читання конфігураційних даних пристрою. Такий підхід має місце при використанні декількох мікросхем ПЛІС на одній друкарській платі. В цьому випадку кожна мікросхема може мати власний виділений пристрій зовнішньої пам'яті і конфігуруватися окремо, як показано на рис. 4.52. Згідно з іншим підходом, ПЛІС можуть з'єднуватися каскадом, тобто за схемо послідовного опитування, і використовувати спільно один пристрій зовнішньої пам'яті (рис. 4.53).

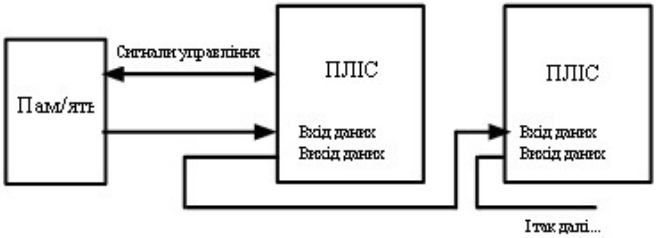


Рис. 4.53. Послідовне включення мікросхем FPGA

У схемі, показаній на рис. 4.53, перша ПЛІС в ланцюжку (вона з'єднується напряму із зовнішньою пам'яттю) конфігуруватиметься для роботи в послідовному режимі як ведуча (master). Усі подальші ПЛІС в цьому ланцюжку повинні бути конфігуровані для роботи в послідовному режимі як ведені (slave) [24].

4.6.9. Паралельне завантаження, ПЛІС у режимі «ведучий»

У багатьох відношеннях цей режим подібний до попереднього, але при цьому дані читаються 8-бітовими фрагментами з пам'яті, яка містить також вісім виводів даних. Групи з восьми бітів досить широко поширені і називаються байтом.

Окрім забезпечення управляючих сигналів, ПЛІС формує для зовнішньої пам'яті сигнали адреси, що використовуються для визначення байта конфігураційних даних, який завантажуватиметься на наступному такті (рис. 4.54).

Подібна схема працює за умови, що ПЛІС містить внутрішній лічильник, який використовується для генерації адреси для зовнішньої пам'яті. Перші ПЛІС містили 24-бітові лічильники, які дозволяли їм адресувати 16 мільйонів байтів даних. На початку конфігураційної послідовності лічильник скидався в нуль. Після того, як байт даних лічильника адреси був зчитаний, вміст лічильника збільшувався для адресації наступного байта. Цей процес тривав доти, поки конфігураційні дані повністю не завантажувалися в мікросхему [24].

Рис. 4.54. Паралельне завантаження в режимі «ведучий» (перший спосіб)

Здавалося б, що такий метод паралельного завантаження має перевагу в швидкості в порівнянні з розглянутим раніше послідовним способом. Проте спочатку це було не так. У перших мікросхемах, як тільки байт даних зчитувався ПЛІС, він, як і раніше, послідовно передавався у внутрішній

конфігураційний регістр зсуву. Ця ситуація була виправлена в більш сучасних пристроях.

Спеціальні пристрої пам'яті, що створюються для використання сумісно з ПЛІС в даний час, є відносно недорогими і виготовляються за Flash-технологіями, тобто є пристроями багатократного використання. Сучасні ПЛІС використовують нові варіанти паралельного завантаження. У цих випадках зовнішня пам'ять є спеціальним пристроєм, який не вимагає зовнішньої адресації, тобто для цих цілей ПЛІС більше не потребує внутрішнього лічильника (рис. 4.55) [24].

Рис. 4.55. Паралельне завантаження в режимі «ведучий» (сучасна реалізація)

В цьому випадку так само, як і в послідовному режимі, ПЛІС просто видає сигнал скидання зовнішньої пам'яті для визначення того, що вона готова почати читання даних з початку конфігураційної послідовності, а потім видає синхроімпульси для синхронізації конфігураційних даних, що надходять із зовнішньої пам'яті.

4.6.10. Паралельне завантаження, ПЛІС у режимі «ведений»

Розглянуті вище режими конфігурації, в яких ПЛІС виступала в ролі ведучої (master), досить привабливі завдяки своїй простоті і невибагливості, оскільки для роботи потрібна тільки ПЛІС і одна мікросхема зовнішньої пам'яті.

Проте на багатьох друкарських платах містяться мікропроцесори, які звичайно використовуються для виконання різноманітних внутрішніх задач. В цьому випадку розробники можуть вирішити використовувати мікропроцесори для завантаження конфігураційних даних у ПЛІС (рис. 4.56) [24].

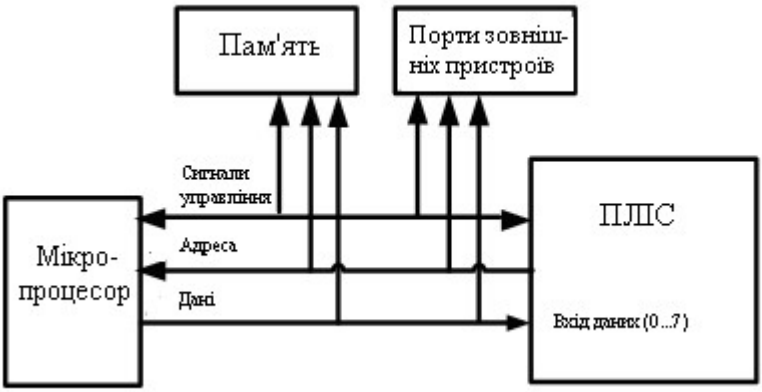


Рис. 4.56. Паралельне завантаження у режимі «ведений»

Ідея полягає у тому, що мікропроцесор є управляючим пристроєм. У цьому режимі мікропроцесор повідомляє ПЛІС, що він готовий почати процес конфігурації, після чого він здійснює читання байта конфігураційних даних з певного пристрою пам'яті або із зовнішнього пристрою, або з інших пристроїв, записує ці дані в ПЛІС, зчитує наступний байт даних з пам'яті, записує його в ПЛІС і так до закінчення процесу конфігурації.

Такий підхід має ряд переваг, не останньою з яких є те, що мікропроцесор може використовуватися для запиту устаткування, що належить сусідній системі, і обирати за певною ознакою конфігураційні дані для завантаження в ПЛІС.

4.6.11. Послідовне завантаження, ПЛІС у режимі «ведений»

Цей режим аналогічний паралельному, але при цьому для завантаження даних в ПЛІС використовується тільки один біт. Мікропроцесор, як і раніше, здійснює читання одного байта даних з пристрою пам'яті і потім перетворює цей байт в послідовність бітів для запису в ПЛІС.

Головною перевагою такого підходу є те, що його реалізація вимагає меншої кількості виводів, задіяних у ПЛІС. Іншими словами, у процесі конфігурації використовується тільки один контакт вводу/виводу, який за допомогою додаткового провідника підключається до шини даних мікропроцесора [24].

4.6.12. JTAG-порт

Багато сучасних пристроїв, у тому числі і ПЛІС, обладнано спеціальним інтерфейсом, який називається JTAG-порт. Цей порт підтримується Об'єднаною робочою групою по автоматизації тестування (Joint Test Automation Group, JTAG) і відомий як стандарт IEEE 1149.1; спочатку він розроблявся для реалізації методу периферійного сканування, який застосовувався для тестування друкарської плати та цифрових мікросхем.

У даному контексті важливо, що ПЛІС має деяку кількість виводів-контактів, які використовуються як JTAG-порт. При цьому один з цих виводів використовується для вводу даних, інший – для виводу, а інші – для зв'язку з послідовно сполученими JTAG-регістрами або тригерами.

Ідея послідовного сканування полягає в наступному: через JTAG -порт можна послідовно передавати дані в JTAG-регістри, які зв'язані з вхідними виводами мікросхеми. У результаті пристрій, в даному випадку ПЛІС, дістає можливість працювати з цими даними, зберігати результати цієї роботи в JTAG -регистрах, зв'язаних з вихідними виводами мікросхеми, і послідовно видавати ці результати назад в JTAG-порт.

JTAG-пристрої містять різну додаткову управляючу логіку. У випадку ПЛІС JTAG-порт, окрім операцій послідовного сканування, може виконувати і інші функції. Наприклад, він може видавати спеціальні команди, які завантажуються в спеціальний командний JTAG-регістр через вхід JTAG-порту. Одна така команда указує ПЛІС під'єднати свій внутрішній конфігураційний регістр зсуву до скануючого ланцюжка JTAG-порту. В цьому випадку JTAG-порт може використовуватися для програмування ПЛІС. Таким чином, сучасні ПЛІС підтримують п'ять різних режимів програмування, і, отже, для визначення того, який з них використовуватиметься, потрібно вже три виводи, як показано у табл. 4.4 (в майбутньому також можуть з'явитися і додаткові режими).

Таблиця 4.4 – П'ять сучасних конфігураційних режимів

Виводи режиму установлення	Найменування режиму
000	Послідовне завантаження, ПЛІС у режимі «ведучий»
001	Послідовне завантаження, ПЛІС у режимі «ведений»
010	Паралельне завантаження, ПЛІС у режимі «ведучий»
011	Паралельне завантаження, ПЛІС у режимі «ведений»
1xx	Використовується тільки JTAG-порт

Можна відмітити, що JTAG-порт завжди доступний до використання, тобто пристрій може бути спочатку конфігурований через звичний конфігураційний порт, шляхом установлення одного із стандартних конфігураційних режимів, а потім, при необхідності, може бути

реконфігурований за допомогою JTAG-порту. Також існує можливість програмувати пристрій за допомогою тільки JTAG -порта [24].

Контрольні питання

1. Що таке ПЛІС?
2. Що таке замовні інтегральні схеми ASIC і ASSP?
3. Сфери застосування ПЛІС.
4. У чому суть методів плавких та нарощуваних перемичок?
5. Структура комірки ППЗП на основі транзистора з плавкою перемичкою.
6. Що таке ППЗП з електричним стиранням?
7. Класифікація архітектурних особливостей ПЛП.
8. Основні фірми-виробники та постачальники ПЛІС.
9. Класифікація спеціалізованих замовних ВІС (ASIC).
10. Що таке вентильна матриця?
11. Що таке канальна логічна матриця?
12. З чого складається типова спеціалізована структурована ІС?
13. З чого складається простий програмований логічний блок?
14. Структура логічного блока на мультиплексорах.
15. Що являє собою ланцюжок конфігураційних комірок?
16. Архітектура базисного модуля, що конфігурується, (логічна комірка фірми Xilinx).
17. Що таке логічний блок, що конфігурується (КЛБ)?
18. Що таке дерево синхронізації ПЛІС?
19. Які функції виконують диспетчери синхронізації?
20. Що таке банки вводу/виводу?
21. Що таке конфігураційний файл?
22. Що таке конфігураційна комірка?
23. Структура ПЛІС на нарощуваних перемичках.
24. Структура ПЛІС на комірках статичного ОЗП.
25. Суть паралельного та послідовного завантаження ПЛІС у режимі «ведучий»?
26. Суть паралельного та послідовного завантаження ПЛІС у режимі «ведений»?
27. Що таке JTAG-порт?
28. Перелік конфігураційних режимів ПЛІС.

СПИСОК ЛІТЕРАТУРИ

1. Бондіна Н. М. Комп'ютеризація спеціалізованих середовищ : навч. посіб. / Н. М. Бондіна, А. І. Поворознюк, О. М. Шеін. – Х. : "НТМТ", 2013. – 378 с.
2. Локазюк В. М. Інтелектуальне діагностування мікропроцесорних пристроїв та систем : навч. посіб. для вузів / В. М. Локазюк, О. В. Поморова, А. О. Домінов. – Хмельницький, 2001. – 286 с.
3. Шлезингерг М. И. Десять лекций по статистическому и структурному распознаванию / М. И. Шлезингерг, В. Главач. – К. : НТУУ “КПИ”, 2004. – 545 с.
4. Козіна О. А. Комп’ютерні системи медичної діагностики : навч. посіб. Ч.1 / О. А. Козіна, А. І. Поворознюк, Г. Є. Філатова. – Х. : НТУ "ХПІ", 2007. – 224 с.
5. Поворознюк А. И. Системы поддержки принятия решений в медицинской диагностике / А. И. Поворознюк. – Saarbrücken Germany: LAP LAMBERT Academic Publishing GmbH & Co. KG, 2011. – 314 с.
6. Максюта Н. В. Алгоритмы и методы снижения размерности пространства диагностических признаков / Н. В. Максюта, А. И. Поворознюк // Вісник НТУ "ХПІ". – Х. : НТУ"ХПІ", 2005. – №46. – С. 126–132.
7. Методы математической биологии : Учеб. пособ. для вузов в 8 кн. Кн. 2. Методы синтеза алгебраических и вероятностных моделей биологических систем. – К. : Вища школа, 1981. – 312 с.
8. Ивахненко А. Г. Моделирование сложных систем по экспериментальным данным / А. Г. Ивахненко, Ю. П. Юрачковский. – М. : Радио и связь, 1987. – 120 с.
9. Кузьмин И. В. Основы теории информации и кодирования / И. В. Кузьмин, В. А. Кедрус. – К. : Вища школа, 1986. – 268 с.
10. Васильев В. И. Распознающие системы : справочник / В. И. Васильев. – К. : Наук. думка, 1983. – 422 с.
11. Журавлев Ю. И. Алгоритмы вычисления оценок и их применение / Ю. И. Журавлев, М. М. Камилов, Ш. Е. Тулягинов. – Ташкент: Фан, 1974. – 344 с.
12. Ротштейн А. П. Интеллектуальные технологии идентификации : нечеткая логика, генетические алгоритмы, нейронные сети / А. П. Ротштейн – Винница : Универсум-Винница, 1999. – 320 с.
13. Генетические алгоритмы, искусственные нейронные сети и проблемы виртуальной реальности / [Г. К. Вороновский, К. В. Махотило, С. Н. Петрашев, С. А. Сергеев] – Х. : Основа, 1997. – 112 с.
14. Будник М. М. Новий підхід до оцінки діагностичних тестів у медичній діагностиці / М. М. Будник // Управляющие системы и машины.– 2006. – №5. – С. 48–55.

15. Файнзильберг Л. С. Гарантированная оценка эффективности диагностических тестов на основе усиленного ROC-анализа / Л. С. Файнзильберг, Т. Н. Жук // Управляющие системы и машины. – 2009. – №5. – С. 3–13.

16. Методы математической биологии : Учеб. пособ. для вузов в 8 кн. Кн. 8. – К. : Вища школа, 1984. – 344 с.

17. Колесин И. Д. Математическая модель саморегулируемой паразитарной системы / И. Д. Колесин // Биофизика. – 1993. – Т. 38, № 5. – С. 892–894.

18. Колесин И. Д. Анализ непостоянства эпидемических вспышек в ритмике сезонных подъемов заболеваемости / И. Д. Колесин // Биофизика. – 1995. – Т. 40, вып.1. – С. 126–130.

19. Антомонов Ю. Г. Математическая теория системы сахара крови / Ю. Г. Антомонов, С. И. Кифоренко. – К.: Наук. думка, 1971. – 83 с.

20. Древаль А. В. Проверка некоторых гипотез патогенеза сахарного диабета путем математического моделирования / А. В. Древаль // Биофизика. – 1983. – Т. 28, вып. 5. – С. 853–858.

21. Васиева О. О. Моделирование автоволновой агрегации DICTYOSTELIUM DISCOIDEUM / О. О. Васиева, В. Н. Васиев, В. А. Карпов, А. Н. Заикин // Биофизика. – 1995. – Т. 40, вып. 2. – С. 393–411.

22. Богач П. Г. Алгоритмы и автоматные модели деятельности гладкомышечных мышц / П. Г. Богач, А. В. Решодько. – К. : Наук. думка, 1979. – 348 с.

23. Бондарев В. Н. Цифровая обработка сигналов: методы и средства. Учеб. пособ. для вузов / Бондарев В.Н., Трестер Г., Чернега В.С. – Севастополь : Изд-во СевГТУ, 1999. – 398 с.

24. Maxfield Clive FPGAs World Class Designs. – Newnes, Burlington, USA, 2009 – 527 p.

Навчальне видання

ПОВОРОЗНЮК Анатолій Іванович
ПОВОРОЗНЮК Оксана Анатоліївна

МЕТОДИ ТА ЗАСОБИ МОДЕЛЮВАННЯ СКЛАДНИХ ДИНАМІЧНИХ ОБ'ЄКТІВ

Навчальний посібник
для магістрів спеціальності 123 «Комп'ютерна інженерія»

Відповідальний за випуск проф. Заковоротний О.Ю.

Роботу до видання рекомендував проф. Заполовський М.Й.

В авторській редакції

План 2024 р., поз. 94

Підп. до друку_____ Гарнітура Times New Roman.
Видавничий центр НТУ «ХПІ», вул. Кирпичова, 2, м. Харків, 61002
Свідоцтво про державну реєстрацію ДК № 5478 від 21.08.2017 р.
Електронне видання