МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

УКРАЇНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЗАЛІЗНИЧНОГО ТРАНСПОРТУ

М. А. Мірошник, Л. А. Клименко, Я. Ю. Корольова

ТЕХНОЛОГІЇ ТА АВТОМАТИЗАЦІЯ ПРОЄКТУВАННЯ ЦИФРОВИХ ПРИСТРОЇВ СКЛАДНИХ КОМП'ЮТЕРНИХ СИСТЕМ НА ПЛІС

Навчальний посібник

Харків 2021

УДК 004.415.03:[004.27:004.89], 681.3:324-326
М 31

Рекомендовано вченою радою Українського державного університету залізничного транспорту як навчальний посібник (витяг з протоколу № 8 від 3 грудня 2019 р.)

Рецензенти:
професори Г. Ф. Кривуля (ХНУРЕ),
М. О. Подустов (НТУ «ХПІ»)

М 31 Мірошник М. А., Клименко Л. А., Корольова Я. Ю. Технології та автоматизація проєктування цифрових пристроїв складних комп'ютерних систем на ПЛІС: Навч. посібник. – Харків: УкрДУЗТ, 2021. – 220 с., 140 рис., 13 табл.

ISBN

Викладено теоретичні основи автоматизації процесу проєктування складних комп'ютерних систем, зокрема вбудованих на ПЛІС, а також програмних засобів за допомогою мови опису апаратури VHDL та VeryLog.

Велика увага приділяється питанням автоматизації проєктування програмних засобів за допомогою спеціалізованих САПР.

Розглянуто питання, які виникають у процесі впровадження розроблених систем на залізничному транспорті.

Навчальний посібник призначено для студентів денної та заочної форм навчання для освітніх програм першого та другого рівнів, які навчаються за освітньою програмою «Спеціалізовані комп'ютерні системи», галузь знань 12 «Інформаційні технології», за спеціальностями 123 «Комп'ютерна інженерія» та 126 «Інформаційні системи і технології», а також для інших спеціальностей відповідних напрямків.

Навчальний посібник містить спеціальний і додатковий матеріал з дисциплін «Технології та автоматизація проектування цифрових пристроїв комп'ютерних систем», «Теорія і проектування комп'ютерних систем», «Системи автоматизації проектування пристроїв і систем автоматики», «Проектування цифрових пристроїв на ПЛІС», «Автоматизоване проектування програмних засобів систем ЗАТ», «GRID-технології в сучасних комп'ютерних системах», «Апаратне і програмне забезпечення комп'ютерних систем загального і спеціального призначення» і може застосовуватися при виконанні курсових і розрахунково-графічних робіт.

Навчальний посібник призначений для студентів закладів вищої освіти і фахівців у галузі комп'ютерних систем і програмування.

УДК 004.415.03:[004.27:004.89], 681.3:324-326

ISBN

ЗМІСТ

Основні скорочення 5
Вступ 9
1. Проєктування складних цифрових пристроїв 12
1.1. Етапи проєктування цифрових пристроїв 12
1.2. Технології проєктування цифрових систем 15
1.3. Програмні результати навчання за освітньою програмою СКС другого рівня магістр 17
Контрольні запитання 18
2. Технології автоматизації проєктування за допомогою комбінаційної логіки 19
2.1. Автоматизація мінімізації булевих функцій 19
2.2. VHDL-опис комбінаційних схем 26
2.3. Комбінаційні компоненти програмованих логічних схем 28
Контрольні запитання 31
3. Технології автоматизації проєктування за допомогою послідовної логіки 32
3.1. VHDL-модель тригерів 32
3.2. Регістри 38
3.3. Синтез тригерів на базі інших тригерів 39
Контрольні запитання 40
4. Абстрактний цифровий автомат 41
4.1. Визначення цифрового автомата 41
4.2. Типи цифрових автоматів 44
4.3. Способи задавання складних цифрових автоматів 46
4.4. Зв'язок між автоматами Мілі та Мура 48
Контрольні запитання 51
5. Структурний синтез складних цифрових автоматів 51
5.1. Кодування станів автоматів, що реалізуються на ПЛІС 51
5.2. VHDL-моделі керуючих автоматів 52
Контрольні запитання 67
6. Операційні автомати 68
6.1. VHDL-модель операційного автомата 68
6.2. Синтез канонічної структури операційного автомата 81

6.3. Операційний автомат типу I 90
6.4. Операційний автомат типу M 91
6.5. Операційний автомат типу IM 93
6.6. Операційний автомат типу S 96
Контрольні запитання 98
7. Основи використання мов опису апаратури 99
7.1. Мови опису апаратури 99
7.2. Ознайомлення з основами мови опису апаратури Verilog 102
Контрольні запитання 134
8. Вивчення існуючих напрямів і тенденцій у проєктуванні і виробництві сучасних інтегральних схем 135
8.1. Класифікація програмованих логічних інтегральних схем 135
8.2. Основні типи FPGA 145
8.3. Проєктування з FPGA 158
Контрольні запитання 190
9. Синтез з використанням мови опису апаратури Verilog 191
9.1. Синтез конструкцій у Verilog 191
Контрольні запитання 199
10. Імплементація пристроїв і аналіз часових параметрів 200
10.1. Методи та етапи тестопридатного проєктування обчислювальних систем і пристроїв на ПЛІС 200
10.2. Місце імплементації в процесі проєктування. Використання пакета Xilinx ISE 203
10.3. Стандарт IEEE P1076.4: Timing methodology (VITAL) (методологія часового аналізу) 207
10.4. VITAL-бібліотеки в VHDL 208
10.5. Оцінювання часових параметрів проєктованого пристрою 211
Контрольні запитання 217
Висновки 218
Бібліографічний список 219

ОСНОВНІ СКОРОЧЕННЯ

ASIC – Application Specific Integrated Circuit
ASMBL – Application Specific/(Advanced Silicon) Modular Block
ASSP – Application Specific Standart Circuit
CAD – Computer Aided Design
CMOS – Complementary metal-oxide semiconductor (комплементарний метал-оксидний напівпровідник (КМОН))
CPLD – Complex Programmable Logic Device (складний напівпровідниковий логічний пристрій)
DSL – Digital Subscriber Line (цифрова абонентська лінія)
DSP – Digital Signal Processing
EDIF – Electronic Digital Interchange Format
EPROM – Electrically Programmable Read-Only Memory
EPROM-OTP – Electrically Programmable Read-Only Memory – One Time Programmable
ES – Embedded System
FLEX – Flexible Logic Element Matrix (матриця елементів гнучкої логіки)
FPGA – Field Programmable Gate Array (програмована користувачем вентильна матриця)
GAL – Generic Array Logic (типова/узагальнена матрична логіка)
IMM – Ideal Multifunction Machine (багатофункціональна машина IDEAL)
IP – Intellectual Properties
ITM – Internal Test Mode
JTAG – Join Test Action Group
LAN – Local Areal Network (локальна мережа)
LPGA – Laser Programmable Gate Arrays
MAC – Media Access Control
MPGA – Mask Programmable Gate Arrays
MSI – Medium Scale Integration
NES – Networked Embedded System
NoC – Network on-a Chip
NRE – Non recurring engineering (cost) (одноразові витрати на проектування)
PAL – Programmable Array Logic (програмована матрична логіка)
PLA – Programmable Logic Array (програмована логічна матриця)

PLD – Programmable Logic Device (програмований логічний пристрій)
PLS – Programmable Logic Sequencer (програмований логічний секвенсер)
PRM – Programmable Routing Matrix
PROM – Programmable Read-Only Memory (перепрограмований постійний запам’ятовуючий пристрій)
PSoC – Programmable System-on-a-Chip
SoC – System on-a Chip
SoPC – System on Programmable Chip
SPLD – Standart / Simple PLD (стандартний/простий програмований логічний пристрій)
SRAM – Static RAM (Random Access Memory)
SSI – Small Scale Integration
VITAL – VHDL Initiative Towards ASIC Libraries
VHDL – VHSIC Hardware Description Language
VHSIC – Very High Speed Integrated Circuit
VLSI – Very Large Scale Integration

АСУ – автоматизована система управління
БМК – базові матричні кристали
БВН – блок вибору направлення передачі інформації
ВОС – вбудовані обчислювальні системи
ВС – вбудовані системи
ГПТП – генератор псевдовипадкових тестових послідовностей
ГСА – граф-схема алгоритму
ГТП – генератор тестових послідовностей
ГЗ – граф залежності
ГПД – графа потоку даних
ДНФ – диз’юнктивна нормальна форма
ЕОМ – електронно-обчислювальна машина
ЕП – елемент пам’яті
ЗП – запам'ятовуючий пристрій
ЗПР – задача прийняття рішення
ІМ – ієрархічна пам'ять
ІКС – інформаційно- керуюча система
ІС – інтегральна схема
К – комутатор

МВВ – модуль управління введення/виведення
МОП – модулі оперативної пам'яті
МВС – мережеві вбудовані системи
МК – мікроконтролер
МОП – модульні оператори пам’яті
МП – мікропроцесор
НВІС – надвелика інтегральна схема
ОЗП – оперативний запам’ятовуючий пристрій
ОМ – обчислювальні модулі
ОС – операційна система
ПВВ – пристрій введення/виведення
ПД – пам'ять даних
ПрД – процесор даних
ПрК – процесор команд
ПЕ – процесорні елементи
ПЗ – програмне забезпечення
ПЗЗ – процес запису вхідної інформації в регістрі пам'яті
ПЗП – постійний запам'ятовуючий пристрій
ПК – пам'ять команд
ПорК – портативний комп'ютер
ПерК – персональний комп'ютер
ПЛІС – програмовані логічні інтегральні схеми
ПЛМ – програмовані логічні модулі
ПМЛ – програмована матрична логіка
ПЛП – програмований логічний пристрій
ПОС – паралельна обчислювальна система
ППЗП – перепрограмоване ПЗП
ПП – периферійні пристрої
ПСнК – програмована система на кристалі
ПСП – псевдовипадкова послідовність
ПУ – пристрій управління
ПУВВ – пристрій управління введенням і виведенням
ПУП – пристрій управління перенумерацією
РІУС – розподілені інформаційно-керуючі системи
РОН – регістри загального призначення
РПЗП-ЕС – репрограмований постійний запам’ятовуючий пристрій з електричним стиранням

РПЗП-УФ – репрограмований постійний запам’ятовуючий пристрій з ультрафіолетовим стиранням
САПР – система автоматизованого проектування
СДНФ – створена диз’юнктивна нормальна форма
СІС – мікросхема середнього рівня інтеграції
СнК – система на кристалі
СОЗП – статичне ОЗП
СПР – система прийняття рішень
ФБ – функціональний блок
ЦАП – цифро-аналоговий перетворювач
ЦОС – цифрова обробка сигналів
ЦП – центральний процесор
ЦОС – цифрова обробка сигналів

ВСТУП

Сучасний рівень розвитку обчислювальної техніки і засобів передачі інформації відкриває широкі можливості з розроблення і впровадження технічної бази автоматизації. Наявні ж математичні засоби дають змогу формалізувати основні завдання управління.

Обчислювальна техніка зробила можливими багато наукових, комерційних і промислових досягнень і відкриттів, які могли б бути недосяжними в разі відсутності цифрових комп’ютерів. Жодна космічна програма не була б реалізована без комп'ютерного управління в реальному масштабі часу, ефективність функціонування виробничих і фінансових підприємств була б немислимою без автоматичної обробки даних. Комп'ютери використовуються в різних сферах діяльності людей. Загальною властивістю цифрового комп'ютера є його універсальність, тобто застосовність до всього. Він може виконувати послідовно команди – програму, яка обробляє дані. Користувач може задавати і змінювати програму і/або дані відповідно до специфіки завдання. Завдяки цій гнучкості цифрові комп'ютери загального призначення можуть вирішувати широкий спектр завдань обробки інформації. Цифрові комп'ютери загального призначення – найбільш відомий приклад цифрових систем разом з такими системами, як цифрові вимірювальні прилади, лічильники, електронні калькулятори, цифрові дисплеї.

Характерною особливістю цифрових систем є маніпуляція дискретними елементами інформації. Такими дискретними елементами інформації можуть бути електричні імпульси, десяткові цифри від 0 до 9, букви алфавіту, арифметичні операції, розділові знаки або будь-який інший набір значущих символів. Перші цифрові комп'ютери використовувалися переважно для чисельних обчислень. У цьому випадку як дискретний елемент використовувалася цифра. Тоді і з'явився термін *цифровий комп’ютер*. Взагалі більш прийнятною назвою для цифрового комп'ютера було б *дискретна система обробки інформації*. Дискретні елементи інформації подаються в цифрових системах за допомогою фізичних величин, так званих сигналів. Електричні сигнали, такі як напруга і струм, є найпоширенішими. Сигнали в

сучасних електронних цифрових системах мають тільки два дискретних логічних значення (0 і 1), що називаються бінарними (двійковими). Проєктувальник цифрових систем обмежується використанням двійкових сигналів через низьку надійність багатозначних електронних схем. Іншими словами, схеми з десятьма станами з використанням одного дискретного значення напруги для кожного зі станів спроєктувати можна, але вона буде мати дуже низьку надійність функціонування, на відміну від транзисторних схем, які ввімкнені або вимкнені і мають, таким чином, тільки два можливих значення сигналу. Це дозволяє створювати на їх базі гранично надійну конструкцію. До того ж і людській логіці властиво все бінаризувати.

Щоб імітувати фізичні процеси в *цифровому комп'ютері*, величини мають бути дискретизованими. Наприклад, коли змінні процесу подаються безперервними сигналами в реальному масштабі часу, вони дискретизуються аналого-цифровим перетворювачем. Фізична система, чия поведінка описується математичними рівняннями, моделюється в *цифровому комп'ютері* шляхом використання чисельних методів. А коли вирішувана проблема дискретна за своєю природою, як, наприклад, у комерційних додатках, *цифровий комп'ютер* маніпулює змінними в їхній натуральній формі. У даній дисципліні ми будемо вивчати основи логічного проєктування цифрових схем від простих комбінаційних вузлів до цифрових автоматів. Причому будуть розглядатися і класичні підходи, які дозволяли виконувати проєктування цифрових схем вручну, і сучасні методи, що дозволяють за допомогою засобів автоматизованого проєктування позбавити проєктувальника рутинної «ручної праці».

У навчальному посібнику викладено теоретичні основи технології та автоматизації проєктування цифрових пристроїв і складних комп'ютерних систем на ПЛІС. Велика увага приділяється технологіям автоматизації проєктування за допомогою комбінаційної та послідовної логіки, автоматизації проєктування і структурного синтезу абстрактних і складних цифрових автоматів, канонічному методу синтезу мікропрограмних автоматів Мілі та проєктуванню операційних автоматів за допомогою VHDL-моделей.

У посібнику розглянуто мету і завдання навчальної дисципліни, надано загальні основи проєктування цифрових пристроїв (розділ 1). Доступно викладено технології та автоматизація проєктування за допомогою комбінаційною логіки (розділ 2). Розглянуто технології та автоматизацію проєктування за допомогою послідовної логіки (розділ 3) і абстрактних цифрових автоматів (розділ 4). Розглянуто технології та автоматизацію структурного синтезу складних цифрових автоматів (розділ 5) та автоматів за допомогою VHDL-моделей (розділ 6). Розглянуто основи використання мов опису апаратури Verilog (розділ 7). Розглянуто існуючу елементну базу з вивчення існуючих напрямів і тенденцій у проєктуванні і виробництві сучасних інтегральних схем (розділ 8). Розглянуто синтез з мови Verilog, подано синтезовану підмножину мови опису апаратури Verilog (розділ 9). Розглянуто імплементацію пристроїв і аналіз часових параметрі (розділ 10).

Кожний розділ має приклади практичних завдань і контрольні запитання.

1. ПРОЄКТУВАННЯ СКЛАДНИХ ЦИФРОВИХ ПРИСТРОЇВ

1.1. Етапи проєктування цифрових пристроїв

Проєктування цифрової системи починається з розроблення технічного завдання, на основі якого потім будується функціональна схема, яка послідовно перетворюється в реальний пристрій. Моделювання та синтез доповнюють один одного процедурами, використовуваними в процесі проєктування, хоча їх точне взаємовідношення залежить від планованої реалізації. Розглянемо етапи проєктування для спеціалізованих інтегральних схем (ASIC – Application Specific Integration Circuit) (рис. 1.1). Такі замовні мікросхеми є вузькоспеціалізованими, мають високу продуктивність при виконанні будь-яких обчислень, але вони найдорожчі.

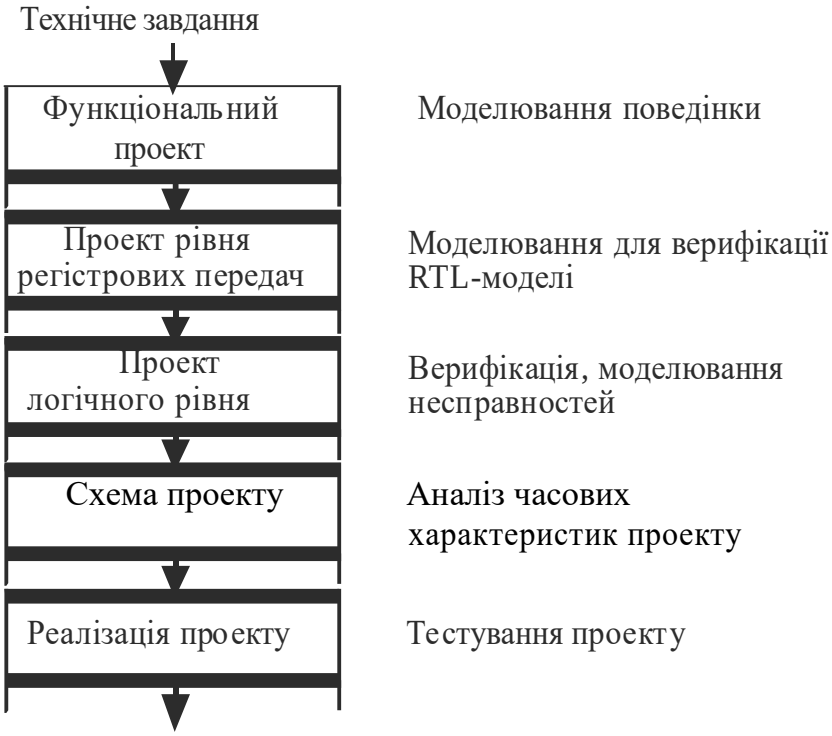


Рис. 1.1. Основні етапи спадного проєктування цифрових систем

Перший етап – створення технічного завдання, яке зазвичай включає вимоги до роботи даного продукту, опис інтерфейсу, вартісні обмеження та інші фізичні вимоги, такі як метричні характеристики системи і розсіювання потужності. На основі

технічного завдання створюється попередня високорівнева функціональна схема. Моделювання на цьому етапі використовується для приведення цієї схеми відповідно до технічного завдання. Потім вона перетворюється в модель рівня регістрових передач (RTL – register transfer level), що описує регістри, модулі пам'яті, операційні та керуючі автомати.

Наступний етап – створення логічних схем для кожного компонента. На рівні регістрових передач і логічному може бути використано моделювання для верифікації проєкту. Моделювання несправностей дозволяє отримувати вихідні реакції схеми для заданого класу дефектів, що виникають на етапах виробництва та експлуатації. Якщо існуючі статистичні дані вказують на досить високий відсоток цих дефектів, проєкт слід модифікувати з метою забезпечення тестопридатності і відновлюваності. Нарешті, опис логічного рівня перетворюється у схемне, а потім проєктується топологія кристала і обчислюються реальні фізичні властивості проєкту, такі як площа кристала і розсіювання потужності. Перевіряються проєктні норми, визначаються параметри схеми, після чого на цьому рівні може бути виконана верифікація проєкту.

На кожному кроці ієрархічного проєктування для опису структури використовуються компоненти. На вищому абстрактному рівні – це невелика кількість складних елементів, таких як суматори або пам'ять. На нижньому – величезна кількість простих елементарних компонентів, наприклад вентилів або транзисторів. Кожен рівень ієрархії проєкту є певною абстракцією, має безліч операцій і підтримує їхні інструменти проєктування, частина з яких наведена на рис. 1.1. За будь-якого рівня абстрактного подання можна передбачити поведінку об'єкта. Але фізичні властивості і можливості схеми бажано розглядати, переходячи до нижніх рівнів, хоча для них потрібен довший час аналізу і проєктування.

Виправлення помилок проєктування, виявлених на заключних етапах розроблення пристрою, є дуже дорогим процесом. Це може призвести до збільшення витрат часу на створення проєкту, а отже, до зростання вартості готового виробу, не кажучи вже про втрати прибутку через запізнення виходу на ринок. Завдяки можливості моделювання поведінки об'єкта на різних рівнях абстракції несправності можна виявляти

і виправляти значно раніше. Мови опису апаратури призначені для їх використання на всіх етапах проєктування і тестування, вони забезпечують сумісність і відповідність між різними рівнями, а також можливості діагностування помилок проєктування за наявності тестів перевірки несправностей, що належать до модельних або фізичних дефектів.

Типова процедура автоматичного синтезу залежить від обраної для імплементації схемотехніки (програмованої логіки). Розглянемо етапи проєктування пристрою для реалізації на програмованих користувачем логічних матрицях FPGA (Field Programmable Gate Arrays). У даному випадку FPGA можна розглядати як електронний пристрій з великою кількістю вентилів і тригерів, зв'язки між якими можна запрограмувати як автоматично, так і ручним способом. На рис. 1.2 показана змінена схема етапів проєктування.

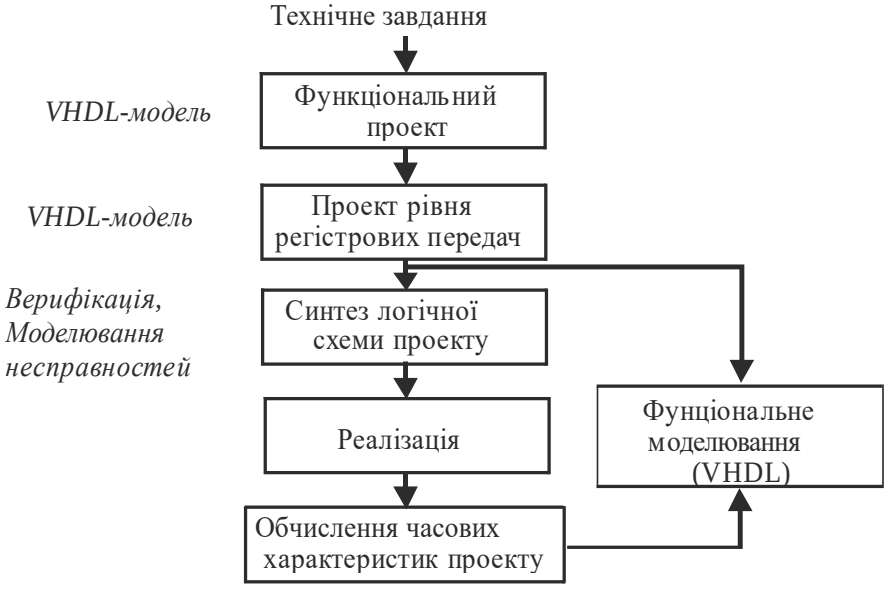


Рис. 1.2. Етапи розроблення проєкту для FPGA

На початку проєктування на основі технічного завдання створюється функціональна схема мовою VHDL і виконується її верифікація. Потім вона перетворюється до рівня регістрових передач, визначаються потоки даних і основні апаратні функціональні модулі. Після отримання VHDL-моделі рівня регістрових передач за допомогою синтезу компілятора

створюється вентильний еквівалент об'єкта. На цьому етапі логічне моделювання може бути використано для попереднього оцінювання можливостей спроєктованого пристрою. Об'єкт на логічному рівні поданий з логічних елементів і тригерів, що входять до складу FPGA. Відповідні вентилі і тригери з'єднуються за допомогою конфігурування сигналів. Такий процес називається розміщенням і маршрутизацією, після закінчення якого можна отримати точні оцінки затримок на лініях і елементах. Після цього виконується моделювання реальних часових характеристик. Узагальнена структура стадій проєктування зображена на рис. 1.3.

Рис. 1.3. Узагальнена структура стадій проєктування

1.2. Технології проєктування цифрових систем

Основу вдосконалення цифрових систем складають досягнення мікроелектроніки. Ефективність її розвитку така, що кожного тижня тактова частота універсального процесора збільшується на 5 МГц, у той час як у 80-ті роки XX століття її підвищення з 5 до 50 кГц відбувалося за три роки. Щодо динаміки вартісних параметрів, то слід зауважити, що двадцять

років тому середня ціна одного транзистора (вентиля) становила 5 дол., а сьогодні на ринку мікроелектроніки за один долар можна купити 5 мільйонів транзисторів. Спостерігається також істотне зниження споживаної потужності кристалів мікросхем і застосування субмікронних технологій при їх виробництві.

Крім того, спостерігається прогрес, пов'язаний з розширенням різноманіття пропонованих схемотехнік на ринку електронних технологій, яке можна подати як три напрями: мікропроцесори (CPU), замовні БІС (ASIC – Application Specific Integration Circuit, VLSI – Very Large Scale Integration) і програмована логіка (FPGA, CPLD). Перші два за високою швидкодією мають недоліки, пов'язані зі значними часовими (не менше 18 тижнів) і фінансовими витратами проєктування обчислювальних пристроїв. Що стосується третього типу, то сучасний рівень розвитку субмікронних технологій мікроелектроніки дозволяє створювати на одному кристалі програмованої логічної інтегральної схеми (ПЛІС) надскладні цифрові системи, що містять 5-8 мільйонів еквівалентних вентилів. Це дає можливість проєктувати складні спеціалізовані обчислювальні пристрої протягом 2-4 тижнів за допомогою засобів автоматизованого проєктування відомих фірм, таких як Aldec, Cadence, Altera, Xilinx, Synopsys, Mentor Graphics.

У наш час світовими лідерами в галузі виробництва програмованої логіки є фірми Xilinx (FPGA – Virte, Spartan), Altera (CPLD, FLEX, Lattice/Vantis (SPLD, PAL, GAL), CPLD), Actel (FPGA), Cypress, Atmel, Quicklogic. Максимальні параметри програмованої логіки: тактова частота – 500 МГц, кількість еквівалентних вентилів – 9 млн, обсяг пам'яті – 400 кбіт, підтримання Boundary Scan. Обсяг продажів програмованої логіки на ринку мікроелектроніки становить понад 60 млрд дол. на рік. Лідером є фірма Xilinx – 28 млрд дол. Спільний розподіл ринку ПЛІС між цими фірмами: Xilinx – 52 %, Altera – 34 %, Lattice – 8 %, Actel – 6 %.

Що стосується коштів автоматизованого проєктування (Electronic Design Automation), то тут лідирують фірми, що працюють у трьох напрямах: 1) введення і моделювання цифрових проєктів (Design Entry & Simulation): Aldec (20000 instalations), Mentor Graphics (12000 instalations); синтез та

імплементація – (Synthesis): Synopsys (74 % продажів на ринку САПР), Mentor Graphics (15 %), Synplicity (11 %); тестування і верифікація (Digital System Testing): Logic Vision, Mentor Graphics, Synopsys, Syntest, Simucad, Cadence.

1.3. Програмні результати навчання за освітньою програмою СКС другого рівня магістр

Згідно з освітньою програмою «Спеціалізовані комп'ютерні системи» другого рівня навчання магістр студенти після закінчення навчання мають такі програмні результати:

ПРН-1. Знання і розуміння впливу комп'ютерних технологій у суспільному, економічному, соціальному та екологічному контексті.

ПРН-2. Вміння здійснювати пошук інформації в різних джерелах для розв'язання задач комп'ютерної інженерії.

ПРН-3. Вміння виконувати експериментальні дослідження за професійною тематикою, оцінювати отримані результати та аргументовано захищати прийняті рішення.

ПРН-4. Вміння застосовувати методи подання знань у системах штучного інтелекту при проєктуванні спеціалізованих комп'ютерних систем для залізничного транспорту, промисловості та бізнесу.

ПРН-5. Вміння розробляти програмно-апаратне забезпечення для вбудованих і розподілених і мобільних систем.

ПРН-6. Вміння створювати і використовувати системи автоматизованого проєктування та діагностування при розробленні спеціалізованих комп'ютерних систем для залізничного транспорту, промисловості та бізнесу.

ПРН-7. Вміння використовувати методи підвищення якості програмного забезпечення комп'ютерних систем переробки інформації та управління.

ПРН-8. Вміння використовувати квантові моделі для підвищення швидкодії аналізу та синтезу програмно-апаратної бази комп'ютерних систем.

ПРН-9. Вміння розробляти нові алгоритми проєктування та діагностування спеціалізованих комп'ютерних систем для залізничного транспорту, промисловості та бізнесу з використанням хмарних мережевих технологій.

ПРН-10. Вміння створювати і використовувати системи автоматизованого проєктування та діагностування спеціалізованих комп'ютерних систем для залізничного транспорту, промисловості та бізнесу.

ПРН-11. Вміння проєктувати спеціалізовані комп'ютерні системи з використанням комп’ютерів Internet jf things.

ПРН-12. Вміння застосовувати методи захисту інформації при проєктуванні та експлуатації спеціалізованих комп'ютерних систем для залізничного транспорту.

Метою вивчення цього блока дисциплін є отримання знань у галузі логічних основ проєктування цифрових схем і методів синтезу та аналізу цифрових автоматів. У результаті студенти повинні знати:

- принцип мікропрограмування, структурні автомати і метод їх канонічного синтезу;
- основи автоматизованого синтезу цифрових автоматів;

повинні вміти:

- виконувати канонічний синтез автоматів за граф-схемою алгоритму;
- складати опис автоматів мовою опису апаратури VHDL;
- користуватися середовищем Project Navigator пакета Xilinx ISE для програмування FPGA на платі Spartan-3E Starter Kit фірми Xilinx;
- синтезувати комбінаційні схеми;
- складати описи автоматів мовою опису апаратури VHDL;
- користуватися середовищем Project Navigator пакета Xilinx ISE для програмування FPGA на платі Spartan-3E Starter Kit фірми Xilinx.

Контрольні запитання

1. Які існують моделі цифрових систем?
2. Які існують технології проєктування цифрових систем?
3. Які існують мови опису апаратури?
4. Що таке VHDL і Verilog?
5. Які існують етапи проєктування цифрових пристроїв?
6. Як відбувається проєктування з FPGA?

2. ТЕХНОЛОГІЇ АВТОМАТИЗАЦІЇ ПРОЄКТУВАННЯ ЗА ДОПОМОГОЮ КОМБІНАЦІЙНОЇ ЛОГІКИ

2.1. Автоматизація мінімізації булевих функцій

Перш ніж говорити про автоматизацію проєктування, розглянемо основні типи інтегральних схем (ІС).

1. Програмовані логічні ІС (ПЛІС, PLD (Programmable Logic Device)) – це ІС з регулярною структурою (повторюваними однаковими блоками), яку можна конфігурувати (запрограмувати) у задану структуру. Вони можуть бути перепрограмованими.

2. Замовні ІС – це ІС, що мають жорстку структуру, яку змінити неможливо. ASIC (Application-Specific Integrated Circuit) проблемно-орієнтована (спеціалізована) – інтегральна мікросхема, замовна ІС, мікросхема для виконання набору спеціальних функцій (наприклад управління пристроєм), зазвичай розробляється під конкретного замовника.

3. Напівзамовні ІС – це різновид замовних ІС і програмованих замовних ІС (Programmable ASIC), функціональність яких може з часом нарощуватися. Вони поєднують у собі і ASIC-, і PLD-особливості. На рис. 2.1 наведена класифікація ПЛІС за архітектурою:

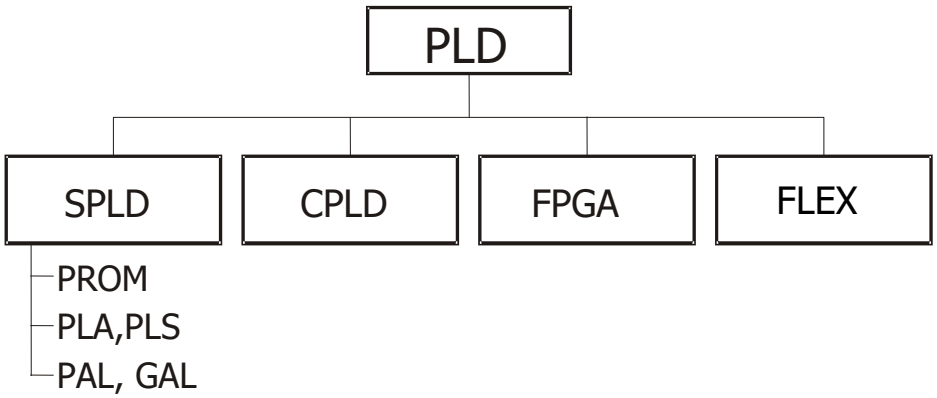


Рис. 2.1. Класифікація ПЛІС за архітектурою

– PLD (Programmable Logic Device) – програмований логічний пристрій;

– SPLD (Standart PLD або Simple PLD) – стандартний або простий програмований логічний пристрій;

– PROM (Programmable Read Only Memory) – перепрограмований постійний запам'ятовуючий пристрій;

– PLA (Programmable Logic Array) – програмована логічна матриця;

– Pal (Programmable Array Logic) – програмована матрична логіка;

– PLS (Programmable Logic Sequencer) – програмований логічний секвенсор;

– GAL (Generic Array Logic) – типова або узагальнена матрична логіка;

– CPLD (Complex PLD) – складний програмований логічний пристрій;

– FPGA (Field Programmable Gate Array) – програмована користувачем вентильна матриця (класична FPGA);

– FLEX (Flexible Logic Element Matrix) – матриця елементів гнучкої логіки (комбінована FPGA).

Перший клас **SPLD** (Standart PLD або Simple PLD) – це стандартні або прості програмовані логічні пристрої. Їх можна поділити на групи: програмовані постійні запам'ятовуючі пристрої (ППЗП), програмовані логічні матриці (ПЛМ) і програмована матрична логіка (ПМЛ). У першому випадку це таблична реалізація функцій, коли в пам'ять заносяться значення функції для вхідних наборів відповідно до адреси клітинки. Наприклад, до комірки з адресою 0000 записується значення функції чотирьох змінних x1, x2, x3, x4 для x1=0, x2=0, x3=0, x4=0.

ПЛМ і ПМЛ реалізують диз'юнктивні нормальні форми (ДНФ) перемикальних функцій за допомогою двох матриць: І і АБО. Перша матриця має m входів і формує q термів, друга матриця з q можливих термів формує n функцій. Таким чином, у ПЛМ можна реалізувати N функцій m змінних, що складаються не більше як з q термів. У ПМЛ матриця АБО є фіксованою, що набагато знижує функціональні можливості цих пристроїв, оскільки для основної маси поширених у практиці завдань великий перетин функцій з термів не є типовим.

Другий клас **CPLD** (Complex PLD) – це складні програмовані логічні пристрої з кількох блоків ПМЛ або ПЛМ,

які об'єднуються за допомогою програмованої комутаційної матриці. CPLD може складатися з сотень блоків і десятків тисяч еквівалентних матриць. Фірми-розробники CPLD: Altera, Atmel, Lattice Semiconductor, Cypress Cemiconductor, Xilinx та ін.

Третій клас **FPGA** – програмовані користувачем вентильні матриці, що складаються з великої кількості логічних блоків, конфігурованих і розташованих по рядках і стовпцях. При цьому використовується табличний спосіб реалізації функцій. Фірми-розробники FPGA: Xilinx, Actel, Agere System (раніше Lucent Technologies), Quick Logic.

Після створення архітектури CPLD і FPGA були плдані в чистому вигляді. Кожна з них має свої переваги і недоліки. Прагнення поєднувати їхні переваги в одному пристрої і зростання інтеграції призвело до появи ПЛІС із комбінованою архітектурою. Такі мікросхеми займають проміжне положення між FPGA і CPLD і поєднують властивості обох класів. Існує дуже багато таких архітектур. Але, як правило, вони мають структуру, подібну до CPLD, але при цьому використовується таблична реалізація функцій. Як приклад перших ПЛІС з комбінованою архітектурою можна навести мікросхеми серії Flex, що випускаються фірмою Altera. Розглянемо основні етапи автоматизованого проєктування пристроїв на ПЛІС на прикладі FPGA XC3S500E-5fg320 (рис. 2.2) і в системі автоматизованого проєктування (САПР) Xilinx ISE. Основні етапи автоматизованого проєктування пристроїв на ПЛІС наведені на рис. 2.3.

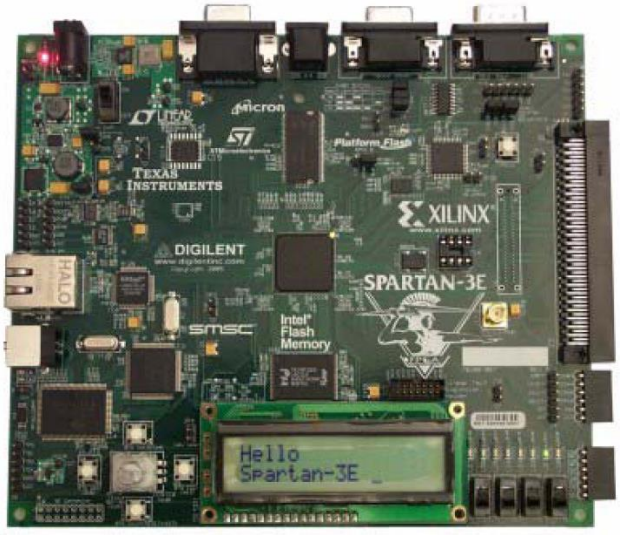


Рис. 2.2. Плата Spartan-3E Starter Kit

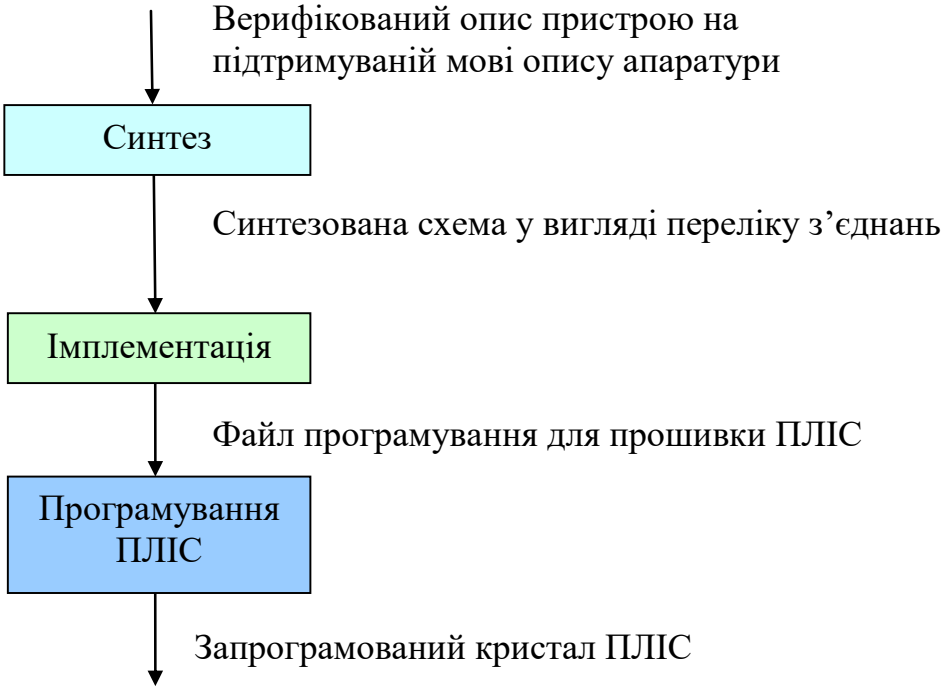


Рис. 2.3. Основні етапи автоматизованого проєктування пристроїв на ПЛІС

На вхід **системи автоматизації проєктування** надходить верифікований опис пристрою підтримуваною мовою опису апаратури, наприклад VHDL-опис (файл з розширенням vhd). Верифікація – перевірка правильності роботи моделі об'єкта. VHDL (Very high speed integrated circuit (VHSIC) Hardware Description Language) – мова опису апаратури, розроблена в 1980-ті роки на замовлення Міністерства оборони США (стандарт IEEE 1076) у рамках проєкту VHSIC зі створення високошвидкісної елементної бази.

Призначений у першу чергу для специфікації – точного опису проєктованих схем і їх моделювання на алгоритмічному, функціонально-блоковому і логічному рівнях проєктування. Має ada-подібний синтаксис, дозволяє описувати одночасні події, структуру системи, виробляти декомпозицію системи на підсистеми, моделювати роботу системи і багато іншого. Має ряд розширень для роботи з аналоговими і змішаними сигналами. Це вхідна мова для багатьох САПР електронних схем, як програмованих, так і замовних. Як приклад VHDL-опису наведемо лістинг 2.1 двовходового суматора за модулем два.

Лістинг 2.1. Двовходовий елемент XOR

```
-- Файл: two_input_xor.vhd
-- Підключення бібліотеки ieee.
library ieee;
-- Підключення пакета бібліотеки ieee.
use ieee.std_logic_1164.all;
-- Опис інтерфейсу пристрою (входи a,b, вихід з)
-- two_input_xor – ім'я інтерфейсу пристрою
entity two_input_xor is
    port ( a, b : in std_logic;
           c : out std_logic);
    end two_input_xor;
-- Опис архітектури пристрою
-- Behavioral-ім'я архітектури пристрою
architecture Behavioral of two_input_xor is
   begin
c <= a xor b;
    end Behavioral;
```

Синтез проєкту – створення схеми, тобто перехід від функціонально коректного опису проєкту мовою VHDL до списку з'єднань (netlist), тобто до машинно-орієнтованої схематики.

Імплементація проєкту – розміщення схеми на цільовій мікросхемі. Імплементація проєкту – це послідовність подій, яка переводить netlist у файл програмування для пристрою FPGA. Синтезований проєкт має ряд портів на верхньому рівні. Для імплементації необхідно знати, як відводити порти проєкту під фізичні виводи мікросхеми FPGA, яка приєднується до різних ресурсів плати Spartan-3E Starter Kit. Якщо не зробити явних призначень, засоби будуть підключені до виводів випадковим чином. Однак це поганий варіант, оскільки випадкове призначення може бути неправильним. Необхідно формувати файл, призначений для користувача обмежень, – user constraint file (UCF). Цей файл (з розширенням ucf) містить обмеження, які не були визначені в VHDL-описі, наприклад розташування виводів, обмеження на характеристики проєкту. Швидше це

зручніше робити в UCF, ніж у VHDL-описі. Наприклад, якщо ви допускаєтеся помилки в призначенні виводів, не потрібно повертатися і заново синтезувати проєкт. Як приклад UCF наведемо лістинг 2.2 для двовходового суматора за модулем два.

Лістинг 2.2. UCF для двовходового елемента XOR

```
NET "a"   LOC = "L13";
NET "b"   LOC = "L14";
NET "c"   LOC = "F12";
```

Тут a і b – вхідні порти (сигнали), c – вихідний порт (сигнал). L13 і L14 – виводи мікросхеми FPGA XC3S500E-5fg320, підключені до перемикачів, що задають вхідні впливи на платі. А F12 – вивід, підключений до світлодіода на платі, на якому можна спостерігати результат (рис. 2.4). На рис. 2.5 виводи мікросхеми, призначені в UCF для сигналів a, b і c, виділені синіми кружками. Дана мікросхема має BGA-виводи.

У результаті виконання даного етапу формується файл програмування ПЛІС. Для FPGA – це bitstream, файл з розширенням bat.

Програмування кристала – надання його структурі конфігурації, що забезпечує заданий алгоритм функціонування пристрою.

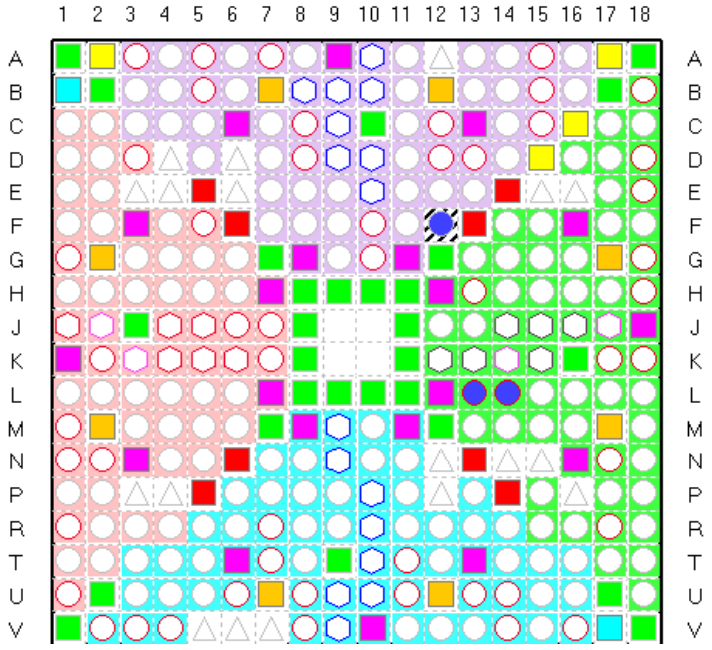


Рис. 2.4. Мікросхема FPGA XC3S500E-5fg320

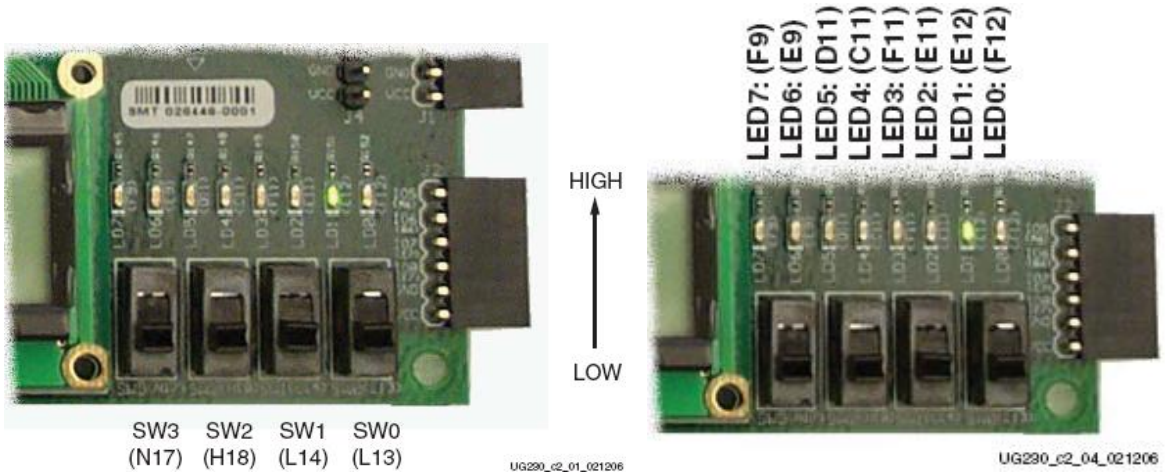


Рис. 2.5. Перемикачі вхідних впливів та індикатори вихідних значень

Програмування FPGA безпосередньо – зручний спосіб випробувати проєкт. Цей метод корисний для прототипування, коли проєкт не є остаточним, щоб упевнитися в його правильності. Складні проєкти не завжди працюють «з першої спроби». Одна з переваг FPGA перед ASIC-схемами полягає в тому, що плата за помилку при першій спробі мінімальна. Адже FPGA можна перепрограмувати, а ASIC доведеться викинути. У табл. 2.1 наведено порівняльну характеристику FPGA- і ASIC-схем.

Таблиця 2.1

Порівняльна характеристика FPGA- і ASIC-схем

Характеристика	FPGA	ASIK
Час побудови продукту до появи на ринку	невиликий	значний
Цінність модулів великого обсягу	низька	висока
Одночасні затрати на проєктування	низькі	високі
Можливість реконфігурації після побудови	висока	низька
Продуктивність	середня	дуже висока
Щільність	середня	дуже висока
Споживана потужність	висока	низька
Мінімальний розмір замовлення	низький	високий
Складніть процесу проєктування	середня	дуже висока
Складність тестування	низька	висока
Загальний час виробничого циклу	години	місяці

2.2. VHDL-опис комбінаційних схем

Пристрої в VHDL описуються за допомогою модулів, а модуль складається з інтерфейсу і архітектури.

Інтерфейс пристрою, його входи і виходи описується за допомогою конструкції entity:

```
entity entity_name is
[port(interface_signal_declaration);]
end [entity] [entity_name];
```

Список сигналів інтерфейсу: тип режиму [:= початкове значення]{ ;

Список сигналів інтерфейсу: тип режиму [:= початкове значення]}

Порти (port) – зовнішні для даного модуля вхідні та вихідні сигнали:

in – вхідний порт;

out – вихідний порт.

Архітектура пристрою реалізує його функцію. Для її запису використовується конструкція architecture:

```
architecture architecture_name of entity_name is
[declarations]
begin
architecture body
end [architecture] [architecture_name];
```

Архітектура може мати поведінковий, структурний або змішаний опис.

Структурний опис – це опис пристрою або системи у вигляді сукупності компонентів (підсхем, елементів) і зв'язків між ними.

Поведінковий опис – це опис пристрою або системи за допомогою певних процедур на рівні залежності виходів від входів. Інакше кажучи, поведінковий опис задає алгоритм реалізованою системою або пристроєм.

Змішаний опис – це опис пристрою або системи при використанні поєднання структурного і поведінкового описів.

Один інтерфейс може бути пов'язаний з декількома архітектурами.

У VHDL коментарі починаються з --.

Ідентифікатори:

- можуть використовуватися символи 'A – -' Z 'i' a ' – 'z', цифри ('0'–'9'), символ підкреслення ('_');
- мають починатися з букви;
- не можуть закінчуватися символом підкреслення;
- не можуть містити два поруч символи підкреслення.

Наприклад, правильні ідентифікатори: y, x1, Up_to, counter;

неправильні ідентифікатори: C@1, 55r, x1_, _y, bit _ _ 0.

Мова VHDL не чутлива до регістра.

Ідентифікатори мають починатися з букви і закінчуватися буквою або цифрою.

Найпростіший спосіб опису комбінаційної схеми – за допомогою оператора паралельного присвоєння '<=' і логічних операторів and, or, nand, nor, xor, xnor, not.

У лістингу 2.1 був наведений приклад VHDL-опису двовходового суматора за модулем два за допомогою оператора паралельного присвоювання і логічних операторів. Тепер розглянемо приклад опису системи булевих функцій, отриманих у прикладі спільної мінімізації системи булевих функцій. У результаті мінімізації були отримані два рівняння:

$$f_1:\ f_1 = x_1 x_2 \vee \overline{x}_1 \overline{x}_2 \overline{x}_3 \vee x_1 \overline{x}_2 x_3.\quad f_2:\ f_2 = \overline{x}_1 x_2 \vee \overline{x}_1 \overline{x}_2 \overline{x}_3 \vee x_1 \overline{x}_2 x_3.$$

У лістингу 2.3 наведено приклад VHDL-опису системи булевих функцій або комбінаційної схеми з трьома входами і двома виходами. Тут, як і в лістингу 2.1, використовується поведінковий опис схеми.

Лістинг 2.3. Приклад VHDL-опису системи булевих функцій

```
library IEEE; -- Підключення бібліотеки ieee.
use IEEE.STD_LOGIC_1164.all; -- підключення пакета бібліотеки ieee.
-- Опис інтерфейсу пристрою (входи x1,x2,x3, виходи f1, f2)
```

```
    -- BF – ім'я інтерфейсу пристрою
    entity BF is
    port(x1,x2,x3: in STD_LOGIC;
    f1, f2: out STD_LOGIC);
    end BF;
    -- Опис архітектури пристрою
    -- System-ім'я архітектури пристрою
    architecture System of BF is
    begin
    f1<=(x1 and x2) or (not x1 and not x2 and
not x2) or (x1 and not x2 and x3);
    f2<=(not x1 and x2) or (not x1 and not x2
and not x2) or (x1 and not x2 and x3);
    end System;
```

2.3. Комбінаційні компоненти програмованих логічних схем

Програмовані логічні ІС (ПЛІС, PLD) це ІС з регулярною структурою (повторюваними однаковими блоками), яку можна сконфігурувати (запрограмувати) в задану структуру. Вони можуть бути перепрограмованими. PLD (Programmable Logic Device) – програмований логічний пристрій.

2.3.1. Програмовані логічні матриці (ПЛМ або PLA)

Розвиток програмованих логічних пристроїв почався з програмованих логічних матриць (ПЛМ), призначених для реалізації комбінаційних схем. Структура матриці подана на рис. 2.6.

Вона складається з трьох блоків: буферів та інверторів, елементів і (І-матриця) і елементів або (АБО-матриця). Блок буферів та інверторів формує прямі та інверсні значення змінних. І-матриця формує значення термів функції, поданої у вигляді ДНФ (кон'юнктивних термів). АБО-матриця формує функції як диз'юнкцію кон'юнктивних термів. Кожен вентиль І в І-матриці має шість входів. Кожне з'єднання на входах вентиля є програмованим. Лінія, приєднана до входу вентиля, позначається

хвилястою лінією, лінія не приєднана до входу вентиля, позначається розірваною лінією. Схема спроєктована таким чином, що непідключені лінії не впливають на виходи вентилів. Програмовані з'єднання існують також на елементах АБО. На ПЛМ реалізовані дві функції: $f_1 = x_1x_2 \vee x_1\bar{x}_3 \vee \bar{x}_1\bar{x}_2x_3$, $f_2 = x_1x_2 \vee x_1x_3 \vee \bar{x}_1\bar{x}_2x_3$.

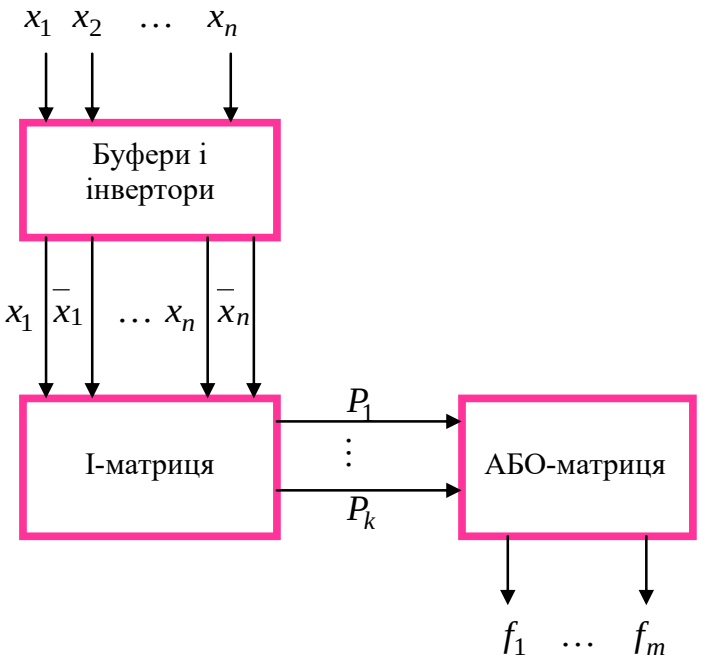


Рис. 2.6. Загальна структура ПЛМ

У матриці І реалізовані терми $P_1 = x_1x_2$, $P_2 = x_1\bar{x}_3$, $P_3 = \bar{x}_1\bar{x}_2x_3$, $P_4 = x_1x_3$. Для отримання f1 до вентиля АБО приєднуються Р1, Р2, Р3, для отримання f2 до вентиля АБО приєднуються Р1, Р3, Р4.

Розмір даної ПЛМ визначається такими параметрами: три входи, чотири терми і два виходи. У промислових масштабах використовуються ПЛМ більшого розміру, наприклад на 16 входів, 32 терми, 8 виходів. Такий спосіб подання дуже наочний і зрозумілий, але абсолютно незручний для великих ПЛМ.

Інший, більш компактний, спосіб подання схеми наведено на рис. 2.7. Замість шести входів біля вентиля І зображується один вхід у горизонтальному напрямі. Можливе з'єднання для входу вентиля І визначається перетином цього умовного входу з вертикальними лініями, на перетині ставиться хрестик. З'єднання для вентиля АБО позначається аналогічно.

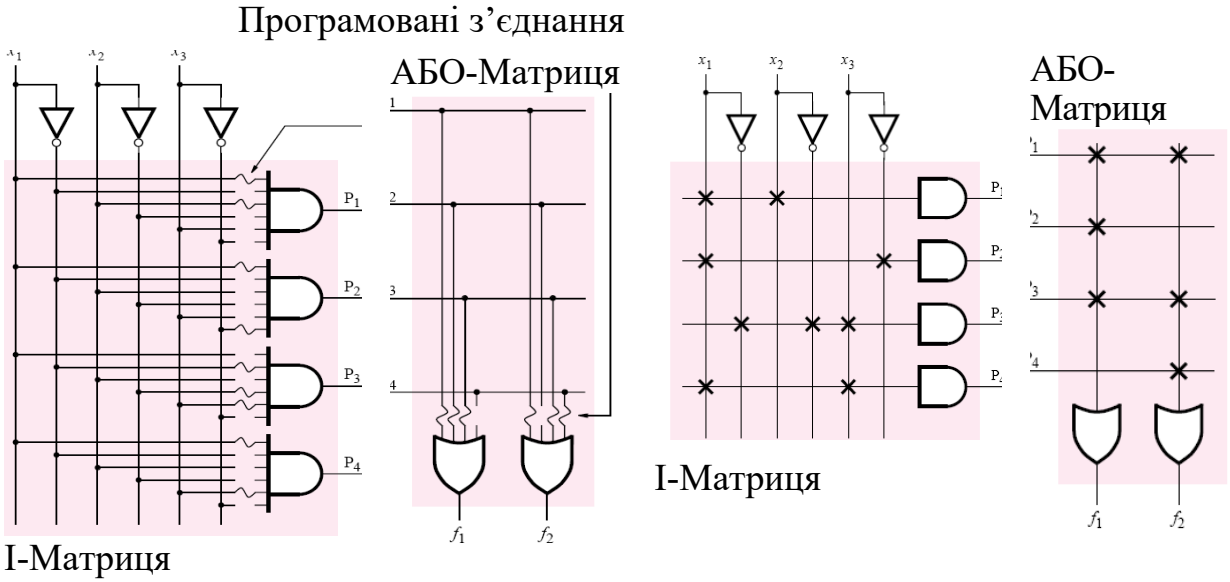


Рис. 2.7. Структура ПЛМ на вентильному рівні

ПЛМ часто використовуються в складі великих логічних схем, наприклад мікропроцесорів.

2.3.2. Програмована матрична логіка (ПМЛ або PAL)

У ПЛМ І-матриця і АБО-матриця програмовані. Програмовані перемикачі створюють дві проблеми. Перша пов'язана зі складністю їх точного виконання при виробництві. Друга пов'язана зі зменшенням швидкодії схеми. Ці проблеми призвели до реалізації програмованого пристрою з програмованою І-матрицею і фіксованою АБО-матрицею (рис. 2.8).

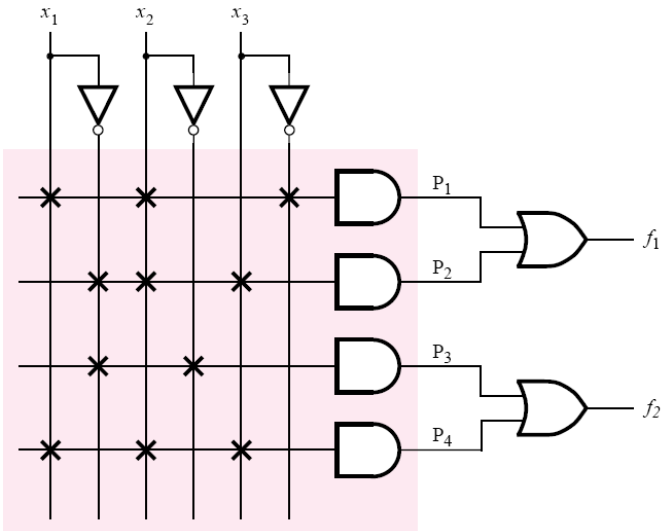


Рис. 2.8. Програмований пристрій з програмованою І-матрицею і фіксованою АБО-матрицею

Такі пристрої називаються ПМЛ-програмованою матричною логікою. Вони простіше у виробництві, дешевше, мають характеристики краще, ніж у ПЛМ, тому стали більш популярними.

На ПМЛ реалізовані дві функції:

$f_1 = x_1 x_2 \bar{x}_3 \vee \bar{x}_1 x_2 x_3$,

$f_2 = \bar{x}_1 \bar{x}_2 \vee x_1 x_2 x_3$.

ПМЛ менш гнучкі. За наявності загальних термів у різних функціях їх доводиться дублювати. У даному прикладі в ПМЛ три входи, а елементи АБО мають по два входи. У промислових схемах існують різні варіанти за кількістю входів, виходів схеми і входів вентилів І та АБО.

ПЛМ і ПМЛ належать до простих програмованих схем SPLD (Standart PLD або Simple PLD – стандартний або простий програмований логічний пристрій.

Макрокомірки на базі ПЛМ або ПЛМ разом з комутаційною матрицею утворюють складні програмовані логічні пристрої CPLD (Complex PLD).

Контрольні запитання

1. Пояснити модель булевого програмування.
2. Пояснити принцип оптимальності в задачі динамічного програмування.
3. Пояснити зв'язок задачі оптимального управління з варіаційним численням.
4. Пояснити зв'язок задач управління з задачами диференціальних ігор.
5. Що таке оптимальна стратегія в диференціальній грі?

3. ТЕХНОЛОГІЇ АВТОМАТИЗАЦІЇ ПРОЄКТУВАННЯ ЗА ДОПОМОГОЮ ПОСЛІДОВНОЇ ЛОГІКИ

3.1. VHDL-модель тригерів

3.1.1. VHDL-моделі D-тригерів

Розглянемо D-тригер (лістинг 3.1, 3.2), синхронізований рівнем 1 (рис. 3.1).

Лістинг 3.1. *D*-тригер, синхронізований рівнем 1

```
library IEEE;
use IEEE.std_logic_1164.all;
entity D_latch is
     port   (D: in STD_LOGIC;
     C: in STD_LOGIC;
     Q: out STD_LOGIC);
end D_latch;
architecture D_latch of D_latch is
   begin
process (C, D)
begin
  if C='1' then   -- синхронний запис Q за
рівнем 1
      Q <= D;
  end if;
end process;
end D_latch;
```

Розглянемо D-тригер, синхронізований рівнем 1 з асинхронним скиданням у 0 (рис. 3.2-3.4).

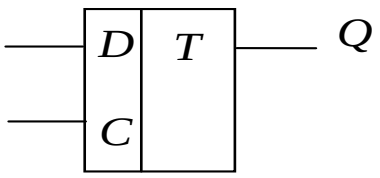


Рис. 3.1. *D*-тригер, синхронізований рівнем 1

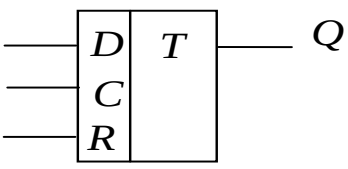


Рис. 3.2. D-тригер, синхронізований рівнем 1 з асинхронним скиданням у 0

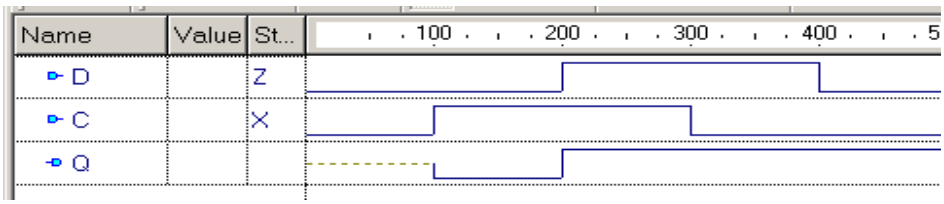


Рис. 3.3. Часова діаграма роботи D-тригера, синхронізованого рівнем 1

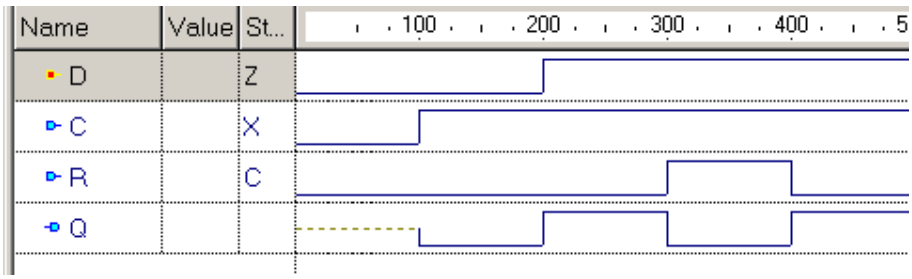


Рис. 3.4. Часова діаграма роботи D-тригера, синхронізованого рівнем 1 з асинхронним скиданням у 0

Лістинг 3.2. D-тригер, синхронізований рівнем 1 з асинхронним скиданням у 0

```
library IEEE;
use IEEE.std_logic_1164.all;
entity D_latch_R is
port   (D: in STD_LOGIC;
C: in STD_LOGIC;
R: in STD_LOGIC;
Q: out STD_LOGIC);
end D_latch_R;
architecture D_latch_R of D_latch_R is
   begin
process (C, R)
begin
  if R='1' then    --асинхронне скидання в 0
    Q <= '0';
elsif C='1' then -- синхронне скидання Q за рівнем 1
    Q <= D;
  end if;
end process;
end D_latch_R;
```

Розглянемо D-тригер, синхронізований переднім фронтом з асинхронним скиданням у 0 (лістинг 3.3, рис. 3.5, 3.6).

Лістинг 3.3. D-тригер, синхронізований переднім фронтом з асинхронним скиданням у 0

```
library IEEE;
use IEEE.std_logic_1164.all;
entity D_ff_aR is
port   (D: in STD_LOGIC;
C: in STD_LOGIC;
R: in STD_LOGIC;
Q: out STD_LOGIC);
end D_ff_aR;
architecture D_ff_aR of D_ff_aR is
   begin
process (C, R)
begin
  if R='1' then --асинхронне скидання в 0
    Q <= '0';
  elsif (C ' event and C='1') then --
синхронний запис Q по передньому
--фронт -- передній фронт описаний за
допомогою атрибута event (подія) і C= '1'
    Q <= D;
  end if;
end process;
end D_ff_aR;
```

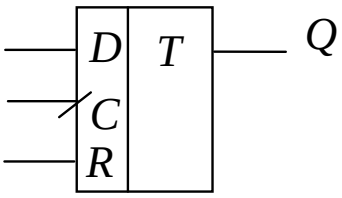


Рис. 3.5. D-тригер, синхронізований переднім фронтом з асинхронним скиданням у 0

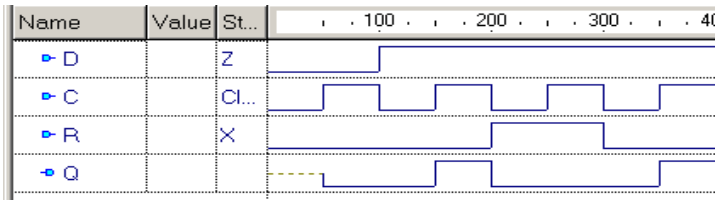


Рис. 3.6. Часова діаграма роботи D-тригера, синхронізованого переднім фронтом з асинхронним скиданням у 0

Розглянемо D-тригер, синхронізований заднім фронтом з синхронним скиданням у 0 (лістинг 3.4, рис. 3.7, 3.8).

Лістинг 3.4. D-тригер, синхронізований заднім фронтом з синхронним скиданням у 0

```
library IEEE;
use IEEE.std_logic_1164.all;
entity D_ff_swiss
port (D: in STD_LOGIC;
C: in STD_LOGIC;
До: in STD_LOGIC;
Q: out STD_LOGIC);
end D_ff_sR;
architecture D_ff_sR of D_ff_swiss
begin
process (C)
begin
if (Clk'event and C='0') then --синхронний
запис Q по задньому фронту
if R='1' then --синхронне скидання в 0
Q <= '0';
else
Q <= D;
end if;
end if;
end process;
end D_ff_sR;
```

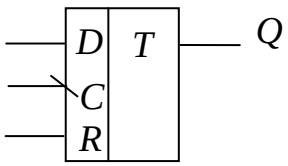


Рис. 3.7. D-тригер, синхронізований заднім фронтом з синхронним скиданням у 0

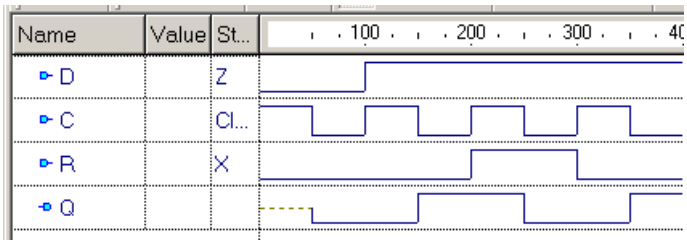


Рис. 3.8. Часова діаграма роботи D-тригера, синхронізованого заднім фронтом з синхронним скиданням у 0

3.1.2. VHDL-моделі JK-тригерів

Розглянемо JK-тригер, синхронізований заднім фронтом з асинхронним скиданням у 0 (лістинг 3.5, рис. 3.9, 3.10). Нижче наведені умовне позначення і VHDL-модель синхронного JK-тригера, керованого заднім фронтом, з асинхронною установкою в 0. Особливістю цієї моделі є те, що для реалізації операції «інверсія стану» використовується внутрішня змінна Qint, оскільки сигнал Q, задекларований в операторі port, не може стояти в правій частині оператора паралельного призначення сигналу <=.

Лістинг 3.5. JK-тригер, синхронізований заднім фронтом з асинхронним скиданням у 0

```
library IEEE;
use IEEE.std_logic_1164.all;
entity JK_tr is
port (J: in STD_LOGIC;
K: in STD_LOGIC;
C: in STD_LOGIC;
До: in STD_LOGIC;
Q: out STD_LOGIC);
утв АЛ_ек;
architecture ОЛ_ек of ОЛ_ег is
begin
process(C, R) is
variable Int: STD_LOGIC;
begin
if (R ='0') then Print:='0';
--асинхронне скидання в 0 (R - інверсний)
elsif (falling_edge(C)) then
--синхронний запис Q по задньому фронту
if (J='1' and K='1') then Int :=
not(Quint);
elsif (J='0' and K='0') then Quit:=Quit;
elsif (J='1' and K='0') then Quint:='1';
elsif (J='0' and K='1') then Qint:='0';
end if;
```

```
    -- задній фронт описаний за допомогою
функції falling_edge(C)
    end if;
    Q<=Quint;
    end process;
    end JK_tr;
```

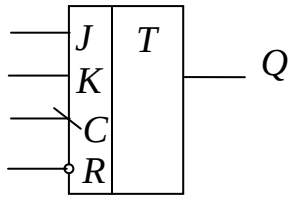


Рис. 3.9. JK-тригер,
синхронізований заднім фронтом
з асинхронним скиданням у 0

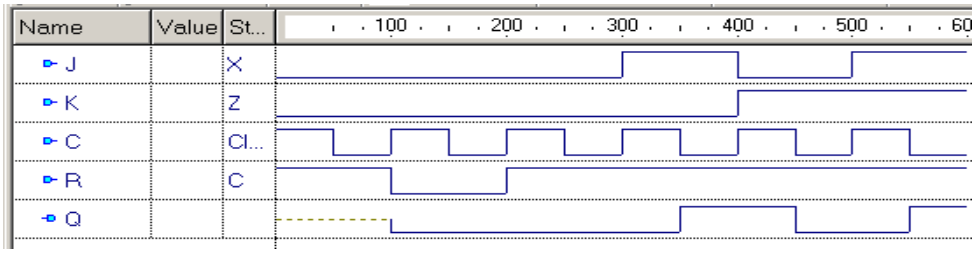


Рис. 3.10. Часова діаграма роботи JK-тригера, синхронізованого заднім фронтом з асинхронним скиданням у 0

Розглянемо JK-тригер, синхронізований переднім фронтом з синхронним скиданням у 0 (лістинг 3.6, рис. 3.11, 3.12).

Лістинг 3.6. JK-тригер, синхронізований переднім фронтом з синхронним скиданням у 0

```
    library IEEE;
    use IEEE.std_logic_1164.all;
    entity JK_tr is
    port (J, K, C, R: in STD_LOGIC;
    Q: out STD_LOGIC);
    утв АЛ_ек;
    architecture ОЛ_ек of ОЛ_ег is
    begin
    process(C, R) is
    variable Int: STD_LOGIC;
    begin
```

```
    if (rising_edge(C)) then
    --синхронний запис Qint по передньому
фронту
    if (R ='1') then Qint:='0';
    --синхронне скидання в 0 (R - прямий)
    elsif (J='1' and K='1') then Int :=
not(Qint);
    elsif (J='0' and K='0') then Quit:=Quit;
    elsif (J='1' and K='0') then Quint:='1';
    elsif (J='0' and K='1') then Qint:='0';
    end if;
    -- передній фронт описаний за допомогою
функції rising_edge(C)
    end if;
    Q<=Quint;
    end process;
    end JK_tr;
```

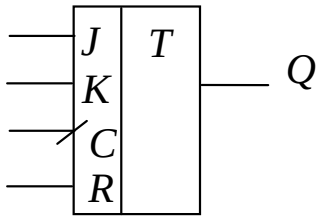


Рис. 3.11. JK-тригер, синхронізований переднім фронтом з синхронним скиданням у 0

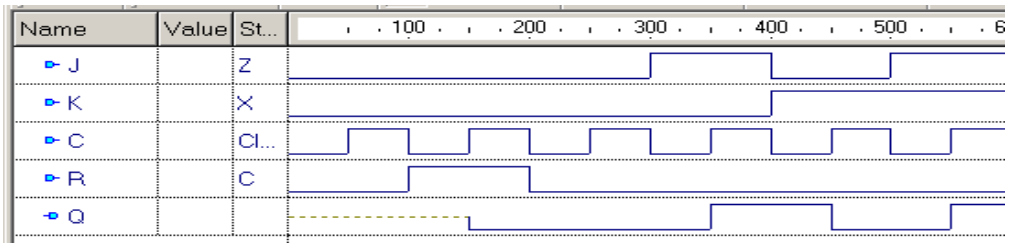


Рис. 3.12. Часова діаграма роботи JK-тригера, синхронізованого переднім фронтом з синхронним скиданням у 0

3.2. Регістри

Регістрами називають цифрові пристрої, призначені для тимчасового зберігання інформації, яка подається на них у вигляді багаторозрядних двійкових чисел. Основою будь-якого

регістра є елемент пам'яті - тригер. Кількість тригерів, розміщених паралельно або об'єднаних послідовно, визначає розрядність регістрів.

У регістрах використовуються RS-, JK- і D-тригери. Для забезпечення управління записом інформації в тригери і її зчитування використовуються комбінаційні схеми, які закладають алгоритм управління регістрами.

Регістри класифікуються за різними ознаками, але основним є спосіб введення і виведення інформації. Виходячи з цього вони поділяються:

- на паралельні (накопичувальні/запису або регістри пам'яті);
- послідовні (регістри зсуву);
- послідовно-паралельні або комбіновані регістри.

Найпростішими структурами є паралельні регістри. Тригери, які використовуються в таких регістрах, мають бути синхронними для забезпечення більш адекватного управління схемою. Одночасно в усі розряди регістра записуються дані, що надходять на його входи.

Послідовні регістри можуть бути однонаправленими або двонаправленими (реверсними). У будь-якому випадку тригери, складові регістри мають бути двоступінчастими (якщо вони статично керовані) або динамічно керованими. Інакше зрушення реалізувати не буде можливим.

Паралельно-послідовний регістр. Мультиплексори з двох в один розділяють два режими: зрушення від входу D3 до виходу Q1 (A = 1) і паралельний запис інформації з входів D1, D2, D3 на виходи Q1, Q2, Q3 відповідно (A = 0).

3.3. Синтез тригерів на базі інших тригерів

У цьому підрозділі розглянемо елементарні цифрові автомати з пам'яттю – тригери, а також їхні властивості та типи. Тригери розглядаються як елементарні послідовні автомати. У більшості серій інтегральних елементів є тригери різних типів, у тому числі й універсальні. У ряді випадків розробнику потрібен тригер зі спеціальними функціями, які не задовольняють наявні

тригери, тобто виникає завдання проєктування довільного тригерного елемента. Далі матеріал розглядатиметься з позицій проєктування з використанням так званого канонічного методу синтезу. Канонічний метод структурного синтезу застосовно до тригерів дозволяє звести завдання їх синтезу до завдання структурного синтезу комбінаційних схем. Результатом канонічного методу структурного синтезу є система логічних рівнянь, яка виражає залежність вихідних сигналів триегрів і функцій збудження елементарних запам'ятовуючих комірок (R*; S*) від сигналів на вході тригера та сигналів з виходів елементарних запам'ятовуючих комірок. Метод містить процедури етапів абстрактного та структурного синтезу, наведені в табл. 3.1.

Таблиця 3.1

Синтез тригерів на базі інших тригерів

(K)	(J)	RS	R^*S^*	JK	K^*J^*	S	S^*	R	R^*	E	E^*	R^*S	RS^*
R	S	Q^{t+1}	Q^{t+1}	Q^{t+1}	Q^{t+1}	Q^{t+1}	Q^{t+1}	Q^{t+1}	Q^{t+1}	Q^{t+1}	Q^{t+1}	Q^{t+1}	Q^{t+1}
0	0	Q^t	X	Q^t	$\overline{Q}^t$	Q^t	1	Q^t	0	Q^t	Q^t	0	1
0	1	1	0	1	0	1	0	1	0	1	0	X	Q^t
1	0	0	1	0	1	0	1	0	1	0	1	Q^t	X
1	1	X	Q^t	$\overline{Q}^t$	Q^t	1	Q^t	0	Q^t	Q^t	Q^t	1	0

Контрольні запитання

1. Що таке задача оптимізації та індивідуальна задача оптимізації?
2. Як оцінюють складність алгоритмів розв’зання оптимізаційних задач?
3. Як формалізуються задачі про найкоротший шлях?
4. Як формалізуються задачі про незалежну множину?
5. Як формалізуються задачі про фарбування графів?
6. Як формалізуються задачі про покриття?
7. Як формалізується задача комівояжера?

4. АБСТРАКТНИЙ ЦИФРОВИЙ АВТОМАТ

4.1. Визначення цифрового автомата

Математичною моделлю дискретного пристрою є абстрактний автомат, який визначається як шестикомпонентний вектор (або кортеж)

$S=<A, X, Y, \delta, \lambda, a_1>$.

1. $A = \{a_1, \ldots, a_m, \ldots a_M\}$ – множина станів (алфавіт внутрішніх станів).

2. $X = \{x_1, \ldots, x_f, \ldots x_F\}$ – множина вхідних станів (вхідний алфавіт).

3. $Y = \{y_1, \ldots, y_g, \ldots y_G\}$ – множина вихідних станів (вихідний алфавіт).

4. Функція $\delta: A \times X \rightarrow A$ – функція переходів, що реалізує сюр'єктивне відображення $\delta: A \times X$ на A. Іншими словами, функція δ деяким парам «стан - вхідний сигнал» (a_m, x_f) ставить у відповідність стан автомата $a_s = \delta(a_m, x_f)$, $a_s \in A$.

5. Функція $\lambda: A \times X \rightarrow Y$ – функція виходів, що реалізує сюр'єктивне відображення $\lambda: A \times X$ на Y. Функція λ деяким парам «стан - вхідний сигнал» (a_m, x_f) ставить у відповідність вихідні сигнали автомата $y_g = \lambda(a_m, x_f)$.

6. $a_1 \in A$ – початковий стан автомата.

Визначення. Відповідність $G \subseteq A \times B$ – це підмножина декартового добутку $A \times B$, де A – множина відправлення, а B – множина прибуття.

Відповідність називається *скрізь визначеною*, якщо область визначень збігається з множиною відправлення.

Відповідність називається *сюр'єктивною*, якщо область значень збігається з множиною прибуття.

Образом елемента a в B в разі відповідності G називається множина всіх елементів $b \in B$, відповідних елементу $a \in A$.

Прообразом елемента b в A при відповідності G називається множина всіх елементів $a \in A$, відповідних елементу $b \in B$.

Відповідність називається ін'єкційною тоді, коли прообразом будь-якого елемента з області значень є єдиний елемент з області визначень.

Відповідність називається функціональною тоді, коли образом кожного елемента з області визначень є єдиний елемент з області значень.

Бієктивна або взаємооднозначна відповідність – відповідність, яка є всюди певною, сюр'єктивною, ін'єкційною і функціональною.

Функціональна відповідність (функція) з A в B – відповідність, при якій кожному елементу з області визначень відповідає єдиний елемент з області значень. Повністю визначена функція з А в В називається відображенням А в В.

Якщо функціональна відповідність (функція) з А в В сюр'єктивна, тобто будь-який елемент з В має прообраз в А, має місце сюр'єктивне відображення, тобто відображення А на В.

Під алфавітом розуміється непорожня множина попарно різних символів. Елементи алфавіту називаються буквами.

Кінцева впорядкована послідовність літер називається словом у даному алфавіті. З розгляду не виключається і порожня послідовність букв (послідовність нульової довжини) - порожнє слово, яке будемо позначати буквою e або X_0.

Абстрактний цифровий автомат має один вхід X і один вихід Y. Автомат функціонує в дискретному часі, який набуває цілих невід'ємних значень $t = 0, 1, 2,\ldots$ У кожен момент t дискретного часу дискретний автомат (ДА) знаходиться в деякому стані $a(t)$ з множини внутрішніх станів A, причому в початковий момент $t = 0$ він завжди знаходиться в початковому стані $a(0)=a_1$. У момент часу t, будучи в стані $a(t)$, автомат здатний сприймати на вході букву вхідного алфавіту $X(t) \in X$. Відповідно до функції виходів він видає в той же момент часу t на виході букву вихідного алфавіту $y(t)=\lambda[a(t), x(t)]$ і відповідно до функції переходів δ переходить у наступний стан $a(t+1)=\delta[a(t), x(t)]$, $a(t) \in A$, $y(t) \in Y$.

Сенс поняття абстрактного цифрового автомата полягає в тому, що він реалізує деяке відображення множини слів вхідного алфавіту X в множині слів вихідного алфавіту Y, інакше, якщо на вхід автомата, встановленого в початковий стан a_1, подавати буква за буквою деяку послідовність букв вхідного алфавіту $x(0)$, $x(1)$, $x(2),\ldots$ – вхідне слово, то на виході автомата будуть послідовно з'являтися букви вихідного алфавіту – $y(0)$, $y(1)$, $y(2),\ldots$ – вихідне слово.

Таким чином, на абстрактному рівні поняття «робота автомата» розуміється як перетворення вхідних слів у вихідні слова. При цьому ми відволікаємося від розгляду внутрішньої структури абстрактного автомата, розглядаючи його як «чорний» ящик. Так називається прийнятий у технічній кібернетиці підхід, коли основна увага приділяється поведінці системи щодо зовнішнього середовища. Таке абстрагування від структури автомата дозволяє вирішити ряд складних завдань, які на структурному рівні приховуються за безліччю деталей, несуттєвих з точки зору функціонування всієї системи в цілому.

Поняття абстрактного цифрового (дискретного) автомата в теорії дискретних пристроїв втілює в собі системний підхід, при якому предмет або явище розглядається як щось ціле, і основний акцент у дослідженні робиться на виявленні різноманіття зв'язків і відносин із зовнішнім середовищем, на відміну від структурного підходу, при якому властивості об'єкта визначаються властивостями складових його елементів.

Поняття внутрішнього стану у визначенні автомата вводиться у зв'язку з тим, що часто виникає необхідність в описі поведінки систем, виходи яких залежать не тільки від станів входів у даний момент часу, але і від деякої передісторії, тобто сигналів, які надходили на входи системи раніше. Стани саме і відповідають деякій пам'яті про минуле, дозволяючи усунути час як явну змінну і виразити вихідний сигнал як функцію стану і входу в даний момент часу.

Автомати з пам'яттю залежно від кількості внутрішніх станів поділяються на *елементарні автомати* (тригери), кількість внутрішніх станів яких дорівнюють двом, і *складні цифрові автомати* (ЦА), кількість внутрішніх станів яких більше двох.

Відомий також клас автоматів, у яких вихід не залежить від передісторії і в кожен даний момент визначається лише вхідним сигналом у цей же момент часу - це так звані комбінаційні схеми або примітивні автомати без пам'яті. Примітивний автомат може бути описаний трійкою $\delta = (X, Y, \lambda)$, де X, Y – вхідний і вихідний алфавіти відповідно, а $\lambda: X \rightarrow Y$ – функція виходів.

Автомат з пам'яттю генерує вихідні сигнали, що залежать від вхідного сигналу і внутрішнього стану. На структурному рівні автомат з пам'яттю має ряд вхідних і вихідних шин.

Генерація вихідних сигналів у комбінаційному автоматі залежить тільки від вхідних сигналів у певний момент часу і не залежить від того, які вхідні сигнали впливали на автомат раніше.

4.2. Типи цифрових автоматів

4.2.1. Автомат Мілі

Два типи автоматів набули найбільшого поширення на практиці: автомат Мілі і Мура, названі так на ім'я перших дослідників цих моделей – американських вчених G. H. Mealy і E. F. Moore.

Функціонування автомата Мілі задається рівняннями

$$a(t+1)= \delta[a(t), x(t)]; \; y(t)=\lambda_1[a(t), x(t)]; \; t=0,1,2,\ldots$$

Розглянемо як приклад спрощений варіант автомата, що продає в метро жетони.

Будемо вважати, що можна класти в автомат тільки одну купюру за один сеанс видачі жетонів і використовувати купюри вартістю 2, 5 і 10 грн. Також є кнопка «пуск», яка запускає процес видачі жетонів, її повинен натискати покупець після захоплення купюри автоматом.

Тобто картина роботи автомата є такою. Автомат знаходиться в стані очікування, він готовий приймати купюри. Покупець кладе в автомат певну купюру. Натискає кнопку «пуск» для підтвердження готовності отримання жетонів. Потім автомат видає жетони і решту, після цього автомат переходить у початковий стан очікування.

Автомат переходить зі стану в стан під дією вхідних сигналів, тобто перехід до нового стану залежить від умови переходу. Вихідний сигнал буде формуватися під час переходу зі стану в момент часу t в стан у момент часу $t+1$. При переході автомата зі стану a_5 у стан a_1 буде формуватися вихідний сигнал y_1.

4.2.2. Автомат Мура

Функціонування автомата Мура задається рівняннями

$$a(t+1)= \delta[a(t), x(t)]; \; y(t)=\lambda_2[a(t)]; \; t=0,1,2,\ldots.$$

Розглянемо як приклад спрощений варіант автомата, що продає в метро жетони, який був розглянутий раніше. Стани, вхідні і вихідні сигнали ті самі. Але вихідні сигнали прописані у вершинах. Це означає, що вихідний сигнал буде формуватися не під час переходу зі стану в момент часу t в стан у момент часу t+1, а в момент, коли автомат перейде в стан у момент часу t. Наприклад, як тільки автомат перейде в стан a_5, буде формуватися вихідний сигнал y_1.

Автомат Мура відрізняється від автомата Мілі. Його функція виходів залежить тільки від внутрішнього стану. Тому, як тільки автомат потрапляє в певний стан, тут же формується вихідний сигнал, який відповідає цьому стану. На графі це зазначається розташуванням вихідних сигналів у вершинах станів після похилої або дробової риски. Вихідна функція автомата Мілі залежить як від внутрішнього стану, так і вхідного сигналу.

4.2.3. С-автомат

Під абстрактним С-автоматом розуміється математична модель цифрового пристрою, що задається вектором S_C=<A, X, Y, U, δ, λ_1, λ_2, a_1>.

У цьому векторі A позначає внутрішній алфавіт; X – вхідний алфавіт; Y – вихідний алфавіт типу 1 (як в автоматі Мілі); U={u_1,…u_r,…u_R} позначає вихідний алфавіт типу 2 (як в автоматі Мура); δ: $A\times X\to A$ – функція переходів, що реалізує сюр'єктивне відображення δ: $A\times X$ на A; λ_1: $A\times X\to Y$ – функція виходів, що реалізує сюр'єктивне відображення λ_1: $A\times X$ на Y; λ_2: $A\to U$ – функція виходів, що реалізує сюр'єктивне відображення λ_2: $A\to U$; a_1 – позначає початковий стан автомату.

С-автомат містить вихідні функції двох типів: типу 1 і типу 2. Тобто він є поєднанням елементів автоматів і Мілі, і Мура.

Функціонування С-автомата задається рівняннями

$$a(t+1)=\delta[a(t), x(t)];\ y(t)=\lambda_1[a(t), x(t)];\ u(t)=\lambda_2[a(t)];\ t=0,1,2,\ldots$$

Розглянемо як приклад спрощений варіант автомата, що продає в метро жетони, який був розглянутий раніше. Стани і вхідні сигнали залишаються тими самими, але додається ще один вихідний сигнал: u_1 – загоряється кнопка «пуск».

Тут за основу взято автомат Мілі і додався вихідний сигнал за типом автомата Мура u_1. Як тільки автомат потрапляє в стан a_2 (або a_3, або a_4), має загорятися кнопка «пуск». А коли кнопка «пуск» буде натиснута, відбуватиметься перехід і стан a_1 і видача жетонів з рештою.

Автомати називаються *дискретними* або *цифровими*, оскільки вони функціонують у часі дискретно або безперервно (за тактами).

Існують автомати *синхронні* і *асинхронні*.

Інтервали часу в синхронних автоматах фіксовані і генеруються спеціальним генератором синхроімпульсів. В асинхронних автоматах моменти дискретного часу визначаються зміною стану пам'яті або вхідних сигналів. У комп'ютерах застосовуються синхронні цифрові автомати.

Автомат називається *кінцевим* (Finite state machine, FSM), якщо кінцеві множини A, X та Y. Надалі будуть розглядатися тільки кінцеві автомати, і термін «кінцевий» може пропускатися.

Автомат називається повністю визначеним, якщо область визначення функцій переходів δ і функцій виходу λ $=D_\delta=D_\lambda=A\times X$. Іншими словами, область визначення δ і λ збігаються з множиною $A\times X$ – множина будь-яких пар (a_m, x_f). У повному обсязі певного або часткового автомата функції δ і λ визначені не для всіх пар $(a_m, x_f) \in A\times X$.

У даній дисципліні будуть вивчатися лише детерміновані автомати, у яких виконана умова однозначності переходів: автомат, що знаходиться в деякому стані, під дією будь-якого вхідного сигналу не може перейти в більш ніж в один стан.

4.3. Способи задавання складних цифрових автоматів

Існує кілька способів задавання цифрових автоматів.

Словесний спосіб являє собою словесний опис поведінки автомата (було розглянуто в попередньому пункті).

Задавання автомата графом переходів. Подання автоматів графом переходів називається також графічним поданням. При цьому способі автомат задається графом переходів (ГП). Граф переходів – орієнтований граф, вершини якого відповідають станам автомата, а дуги – переходам. Дві вершини a_m і a_s

з'єднуються дугою, якщо в автоматі є перехід з a_m в a_s. Дуга відзначається вхідним сигналом x_f і вихідним сигналом y_g.

Якщо вихідний сигнал не визначений, ставиться прочерк (тире). Максимальна кількість дуг, що виходять з вершини графа, дорівнює кількості букв вхідного алфавіту.

Якщо перехід зі стану a_m в стан a_s викликається багатьма вхідними сигналами, то дуги відзначаються всіма цими сигналами.

Автомат Мура задається графом переходів, у якому вихідний сигнал записується всередині вершини або поруч з нею.

Табличний спосіб задавання автоматів. Автомат Мілі задається таблицею переходів (ТП) і таблицею виходів (ТВ). У разі не повністю певного автомата таблиця переходів або таблиця виходів у повному обсязі заповнені, і в них є порожні позиції. Таблиці побудовані на підставі графа переходів автомата.

На перетині стовпця a_i і рядка x_j в таблиці переходів записується стан переходу $a_s(t+1)=\delta[a_m(t), x_f(t)]$, у який автомат переходить зі стану a_m під дією сигналу x_f, а в таблиці виходів – відповідний цьому переходу вихідний сигнал $y(t)=\lambda[a_m(t), x_f(t)]$.

Як нам відомо, вихідний сигнал автомата Мура залежить тільки від внутрішнього стану. Тому автомат Мура може бути заданий однією зазначеною таблицею переходів. Таблиця побудована на підставі графа переходів автомата.

С-автомат задається двома таблицями: таблицею переходів і зазначеною таблицею виходів або, навпаки, зазначеною таблицею переходів і таблицею виходів. Таблиці побудовані на підставі графа переходів автомата.

Автоматна стрічка. Автомат може бути заданий за допомогою автоматної стрічки або стрічки Тьюринга.

Тьюринг (англійський учений, який займався дослідженнями кінцевих автоматів) показав, що будь-якому кінцевому автомату відповідає еквівалентна йому машина Тьюринга, функціонування якої можна задавати стрічкою Тьюринга.

Задавання автомата деревом функціонування. Автомат може бути заданий модифікацією графа переходів, так званого дерева функціонування. Цей спосіб задавання має ту перевагу, що дозволяє проаналізувати роботу автомата такт за тактом. Вершини дерева відзначені станом автомата.

Автомат починає функціонувати зі стану a_1 і далі переходить до інших можливих станів під дією вхідного сигналу x_j. Дерево побудовано за автоматною стрічкою.

Матричний спосіб подання. Будь-який автомат може бути заданий матрицею з'єднань. Матриця з'єднань C має M рядків і M стовпців. Кожен рядок відповідає вихідному стану $a_m(t)$, кожен стовпець – стану переходу $a_s(t+1)$. На перетині рядка i та стовпця j записується елемент C_{ij}. Цей елемент включає чисельник і знаменник.

Чисельник відображує умову переходу (вхідний сигнал x_f) автомата зі стану a_m в стан a_s. Знаменник відображує вихідний сигнал $y_g(t)$, що генерується на переході. Складемо матрицю з'єднань C для автомата Мілі заданого таблицею переходів і таблицею виходів.

Цей спосіб не зручний при зростанні кількості внутрішніх станів, кількість нульових елементів матриці зростає (причиною цього є слабка взаємопов'язаність внутрішніх станів графів переходів цифрових автоматів), що веде до збільшення витрат пам'яті в комп'ютерах.

4.4. Зв'язок між автоматами Мілі та Мура

4.4.1. Алгоритм трансформації автомата Мура в автомат Мілі

Два автомати S_A і S_B з однаковими вхідними та вихідними алфавітами називаються еквівалентними, якщо після встановлення їх у початковий стан їхня реакція (вихідне слово) на будь-яке вхідне слово збігається.

З порівняння стрічок двох автоматів Мілі і Мура видно, що їхні реакції (реакції автомата Мілі і зсунутої на один такт реакції автомата Мура) збігаються.

Вихідним сигналом автомата Мура в такт $t = 0$ нехтуємо, оскільки він визначається не вхідним сигналом автомата в цей момент часу, а виключно станом.

З порівняння двох стрічок, як правило, не слід робити висновок, що обидва автомати S_A і S_B еквівалентні, оскільки дослідження проведено тільки для одного вхідного слова, а не

для всіх допустимих (дозволених) вхідних слів. Відомо спочатку, що вони еквівалентні.

При розгляді алгоритмів взаємної трансформації одного типу в інший будемо (відповідно до викладеного вище) нехтувати в автоматах Мура вихідним сигналом $\lambda(a_1)$, пов'язаним з початковим станом.

Розглянемо спочатку перетворення автомата Мура в автомат Мілі. Нехай дано автомат Мура $S_A=\{A_A, X_A, Y_A, \delta_A, \lambda_A, a_{1A}\}$, у якого

$A_A=\{a_1,\ldots,a_m\}$; $X_A=\{x_1,\ldots,x_F\}$; $Y_A=\{u_1,\ldots,u_G\}$; δ_A: $A_A\times X_A\to A_A$; λ_A: $A_A\to Y_A$, $a_{1A}=a_1$ – початковий стан.

Побудуємо автомат Мілі $S_B=\{A_B, X_B, Y_B, \delta_B, \lambda_B, a_{1B}\}$, у якого $A_B=A_A$; $X_B=X_A$; $Y_B=Y_A$; $\delta_B=\delta_A$; $a_{1B}=a_{1A}=a_1$. Функцію виходів λ_B: $A_B\times X_B\to Y_B$ визначимо таким чином: якщо в автоматі Мура $\delta_A(a_m, x_f)=a_s$ і $\lambda_A(a_s)=u_g$, то в автоматі Мілі $\lambda_B(a_m,x_f)=u_g$.

При переході від автомата Мура до автомата Мілі вихідний сигнал u_g, записаний поруч з вершиною a_s, переноситься на всі дуги, що входять у цю вершину.

При табличному способі задавання автомата Мура (зазначеною таблицею переходів) таблиця переходів автомата Мілі співпадає з таблицею переходів автомата Мура. У таблиці виходів знімається тільки відмітка станів автомата Мілі шляхом заміни символу стану переходу a_s символом вихідного сигналу u_g, який відзначає стовпець a_s в таблиці переходів автомата Мура.

З самого способу побудови автомата Мілі S_B випливає, що він еквівалентний автомату Мура S_A. Розглянемо приклад переходу від ОТП автомата Мура до автомата Мілі, представленому таблицею переходів і таблицею виходів.

4.4.2. Алгоритм переходу від автомата Мілі до автомата Мура

Перш ніж розглянути трансформацію автомата Мілі в автомат Мура, накладемо на автомат Мілі наступне обмеження: у ньому не повинно бути тимчасових станів. Під тимчасовим будемо розуміти стан, у якому при поданні автомата графом переходів не входить жодна дуга, але який має принаймні одну вихідну дугу.

Отже, нехай задано автомат Мілі $S_A=\{A_A, X_A, Y_A, \delta_A, \lambda_м, a_{1A}\}$, де $A_A=\{a_1,\ldots,a_m\}$; $X_A=\{x_1,\ldots,x_F\}$; $Y_A=\{u_1,\ldots,u_G\}$; δ_A: $A_A\times X_A\to A_A$; λ_A: $A_A\times X_A\to Y_A$, $a_{1A}=a_1$ – початковий стан.

Побудуємо автомат Мура $S_B=\{A_B, X_B, Y_B, \delta_B, \lambda_B, a_{1B}\}$, у якого $X_B=X_A$; $Y_B=Y_A$.

Для визначення A_B кожному стану $a_s\in A_A$ поставимо у відповідність множину A_S можливих пар вигляду (a_s, u_g), де u_g – вихідний сигнал u_g, приписаний дузі A_S, що входить в a_s дуги $A_S=\{(a_S, u_g)/\delta_A(a_m, x_f)=a_s \text{ і } \lambda_A(a_m,x_f)=u_g\}$.

Кількість елементів у множині A_S дорівнює кількості різних вихідних сигналів на дугах автомата S_A, що входять до стану a_s.

Множина станів автомата A_S отримаємо як об'єднання множин A_S (s=1, …, M): $A_B=\bigcup_{s=1}^{M} A_s$.

Функції виходів λ_B і переходів δ_B визначимо таким чином. Кожному стану автомата Мура S_B, який являє собою пару вигляду (a_s, u_g), поставимо у відповідність вихідний сигнал u_g. Якщо в автоматі Мілі S_A був перехід $\delta_A(a_m, x_f)=a_s$ і при цьому видавався вихідний сигнал $\lambda_A(a_m, x_f)=u_k$, то в S_B буде перехід з множини станів λ_m, породжуваних a_m, у стан (a_s, u_k) під дією того самого вхідного сигналу.

Як початковий стан a_{1B} можна взяти будь-який із станів множини A_1, що породжується початковим станом a_1 автомата A_A. Нагадаємо, що при порівнянні реакцій автоматів S_A і S_B (або A_A і A_B) на будь-які вхідні слова не повинен враховуватися вихідний сигнал автомата Мура в момент t=0, пов'язаний зі станом a_{1B} автомата A_B.

Викладені методи взаємної трансформації моделей Мілі і Мура показують, що при переході від автомата Мура до автомата Мілі кількість станів автомата не змінюється, тоді як при зворотному переході кількість станів в автоматі Мура, як правило, зростає.

Вона не зростає, якщо кожен стан автомата Мілі породжує тільки один стан автомата Мура.

Контрольні запитання

1. Як пов’язана структура алгоритму з архітектурою паралельної обчислювальної системи?
2. Які основні завдання вирішуються при створенні паралельної обчислювальної системи?
3. Що таке асимптотика і як її використовують для визначення складності алгоритмів?
4. Як класифікуються паралельні обчислювальні системи?
5. Які типи абстрактних машин виділяються в паралельних обчислювальних системах?

5. СТРУКТУРНИЙ СИНТЕЗ СКЛАДНИХ ЦИФРОВИХ АВТОМАТІВ

5.1. Кодування станів автоматів, що реалізуються на ПЛІС

Якщо реалізацію КЧ виконувати на ПЛІС, то в цьому випадку систему булевих функцій слід подати у вигляді найкоротшої диз'юнктивної нормальної форми (КДНФ), оскільки площа кристала для реалізації матриці в цьому випадку буде мінімальною. Слід зазначити, що особливості застосовуваних матриць іноді вимагають використання одиничного позиційного кодування для кодування ВС (одна одиниця в n можливих позиціях).

Наприклад, для автомата на рис. 5.1 коди станів можуть мати вигляд

$K(a_5)$ = 10000; $K(a_4)$ = 01000; $K(a_3)$ = 00100; $K(a_2)$ = 00010; $K(a_1)$ = 00001.

При створенні проєктів для FPGA слід пам'ятати, що кожна логічна комірка містить два тригери. Це означає, що мінімізація кількості використовуваних тригерів не має значення. Замість цього потрібно спробувати скоротити загальну кількість використовуваних комірок і з'єднань між ними. Для того щоб проєктувати більш швидкі схеми, слід зменшувати кількість

комірок, необхідних для реалізації кожного рівняння. Застосування унарного кодування часто допомагає цьому. На відміну від замовних схем, програмовані схеми мають набагато меншу швидкодію. Тому при проєктуванні схем на базі програмованих кристалів боротьба за більшу швидкодію є більш актуальною, ніж боротьба за мінімальні апаратурні витрати.

Для реалізації рівняння з меншою кількістю змінних зазвичай використовується невелика кількість логічних комірок. Якщо, наприклад, функція збудження тригерів містить чотири змінних, для її реалізації знадобиться рівно один LUT. Якщо містить, наприклад, шість змінних, необхідно буде два LUT, з'єднаних каскадно. Чим більше LUT у каскаді, тим більше затримка розповсюдження сигналу і тим менше швидкодія пристрою.

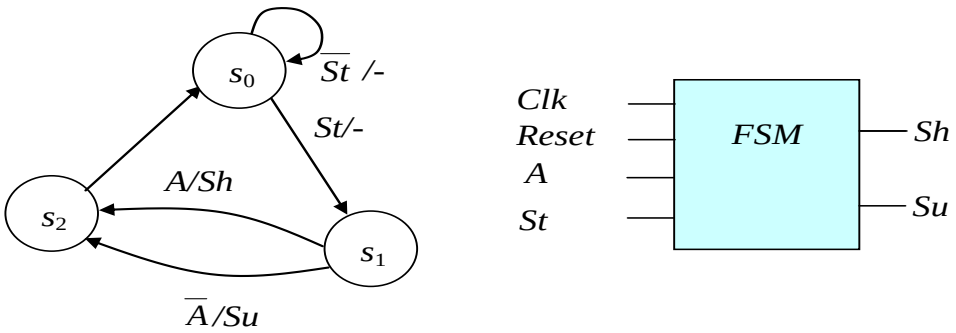


Рис. 5.1. Граф переходів автомата Мілі

5.2. VHDL-моделі керуючих автоматів

Розглянемо VHDL-модель для автомата Мілі, граф і загальний вигляд якого подано на рис. 5.1 (лістинг 5.1), а часові діаграми – на рис. 5.2.

Лістинг 5.1. VHDL-модель керуючого автомата Мілі

```
-- Підключення бібліотеки ieee.
    library IEEE;
    use IEEE.std_logic_1164.all;
-- Опис інтерфейсу пристрою
    entity FSM is
      port ( Clk: in STD_LOGIC;
        Reset: in STD_LOGIC;
```

```
        A,St : in STD_LOGIC;
        Sh, Su: out STD_LOGIC);
    end;
-- Опис архітектури пристрою
    architecture FSM of FSM is
      type State_type is (S0, S1, S2);
      signal State, NextState: State_type;
    begin
--  Блок для формирования последовательностной
части
    Sreg0_CurrentState: process (Clk, Reset)
    begin
      if Reset='1' then
        State <= S0;
      elsif Clk'event and Clk = '1' then
        State <= NextState;
      end if;
    end process;
-- Блок для формування комбінаційної частини
-- Опис за умовами переходів станів і вихідних
сигналів
    Sreg0_NextState: process (State, A, St)
    begin
      Su<='0';
      Sh<='0';
      case State is
        when S0=> if St='1' then NextState <=
S0;
                            else NextState <= S1;
                  end if;
        when S1=> if A='1' then NextState <=
S2;  Sh<='1';
                       else  NextState  <=  S2;
Su<='1';
                            end if;
        when S2=> NextState <= S0;
        when others => NextState <= S0;
      end case;
    end process;
    end;
```

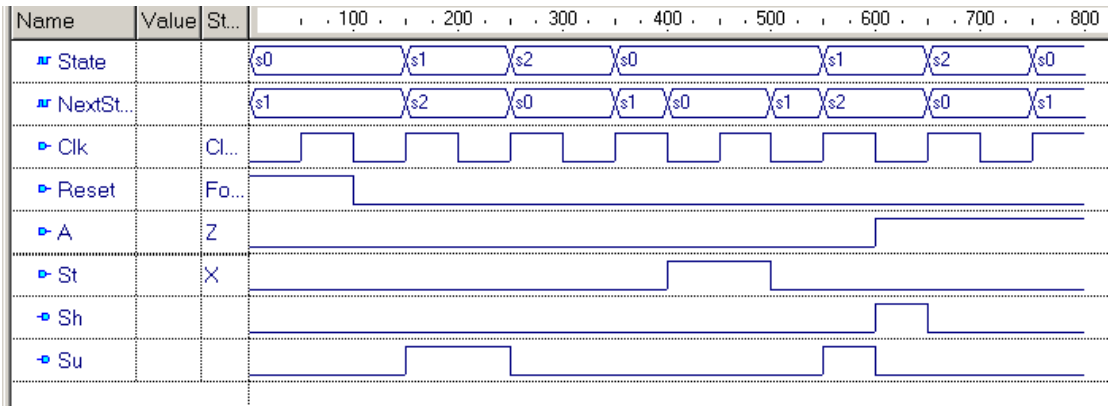


Рис. 5.2. Часові діаграми роботи автомата Мілі

Розглянемо VHDL-модель для автомата Мура, граф і загальний вигляд якого подано на рис. 5.3 (лістинг 5.2), а часові діаграми – на рис. 5.4.

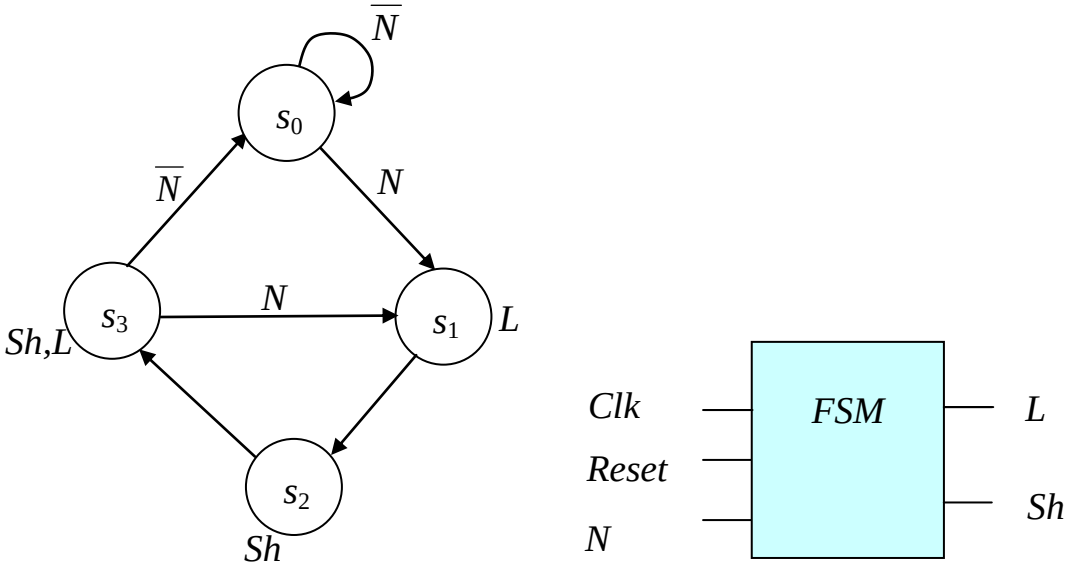


Рис. 5.3. Граф переходів автомата Мура

Лістинг 5.2. VHDL- модель керуючого автомата Мура

```
-- Підключення бібліотеки ieee.
    library IEEE;
    use IEEE.std_logic_1164.all;
-- Опис інтерфейсу пристрою
    entity FSM is
      port ( Clk: in STD_LOGIC;
       Reset: in STD_LOGIC;
```

```
       N: in STD_LOGIC;
       Sh, L: out STD_LOGIC);
    end;
-- Опис архітектури пристрою
    architecture FSM of FSM is
      type State_type is (S0, S1, S2, S3);
      signal State, NextState: State_type;
    begin
-- Блок для формування послідовних частин
    Sreg0_CurrentState: process (Clk, reset)
    begin
      if Reset='1' then
       State <= S0;
      elsif Clk'event and Clk = '1' then
       State <= NextState;
      end if;
    end process;
-- Блок для формування комбінаційної частини
-- Опис переходів станів за умовами
    Sreg0_NextState: process (State, N)
    begin
      case State is
       when S0=> if N='1' then    NextState   <=
      S1;
           else
               NextState <= S0;
          end if;
       when S1=> NextState <= S2;
       when S2=> NextState <= S3;
       when S3=> if N='1' then
                   NextState <= S1;
                 else
                   NextState <= S0;
                 end if;
       when others => NextState <= S0;
      end case;
    end process;
-- Опис вихідних сигналів
    Sh<='1' when State=S2 or State=S3 else
```

```
  '0';
L<='1' when State=S1 or State=S3 else
  '0';
end;
```

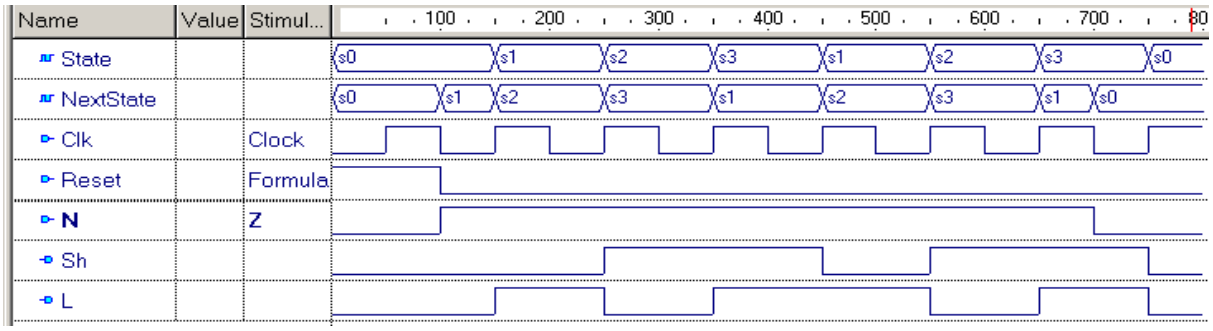


Рис. 5.4. Часові діаграми роботи автомата Мура

Плата Spartan-3E Starter Kit для програмування FPGA XC3S500E-5fg320 використовується як макет, який ілюструє процеси, що відбуваються у схемі. Для подачі вхідних впливів використовуються кнопки (рис. 5.5) і перемикачі для спостереження вихідних сигналів - світлодіоди (рис. 5.6).

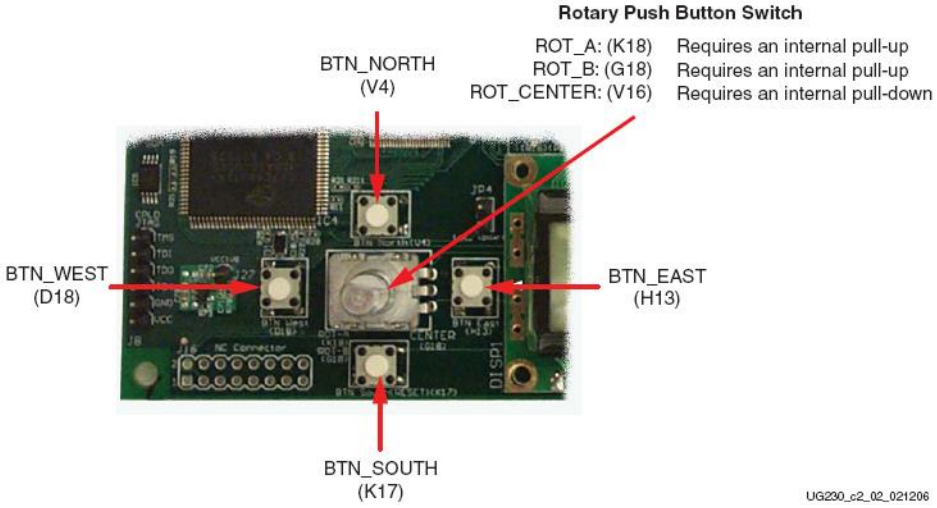


Рис. 5.5. Кнопки для задавання вхідних впливів

Для генерації синхросигналу використовується вбудований генератор з частотою 50 МГц (періодом 20 нс) (рис. 5.6). Також плата дає можливість підключати зовнішні генератори частоти - на платі є два рознімачі для цього (рис. 5.6).

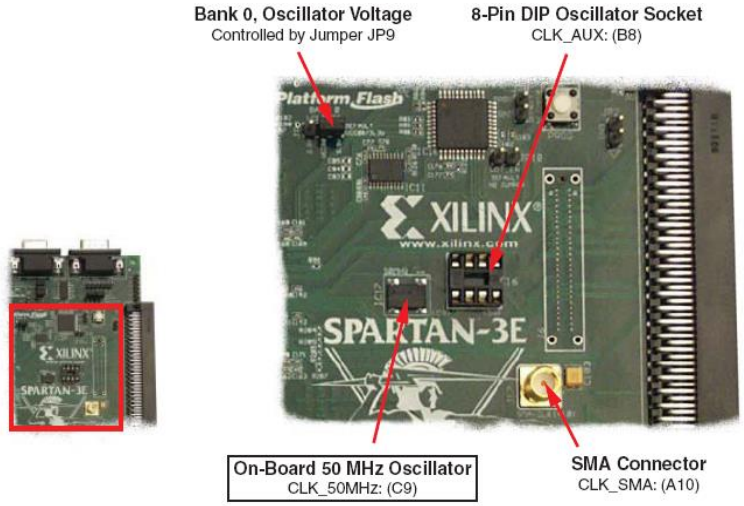


Рис. 5.6. Вбудований генератор синхроімпульсів і рознімачі для підключення зовнішніх генераторів

Якщо при реалізації автомата інформаційні сигнали подавати з перемикачів, а синхросигнал отримувати з вбудованого генератора, спостерігати покрокову роботу автомата буде неможливо через високу швидкодію схеми, що визначається частотою синхросигналу.

Якщо синхросигнал подавати з перемикача або кнопки, ми теж не зможемо спостерігати покрокову роботу автомата, оскільки кнопки і перемикач дають на вхід схеми не чистий сигнал, а сигнал з брязкотом (точніше сигнал, що викликається брязкотом механічних контактів), тобто з багаторазовою зміною сигналу протягом короткого часу (рис. 5.7). Ця багаторазова зміна сигналу утворює безліч фронтів, за якими автомат зможе переключатися неодноразово.

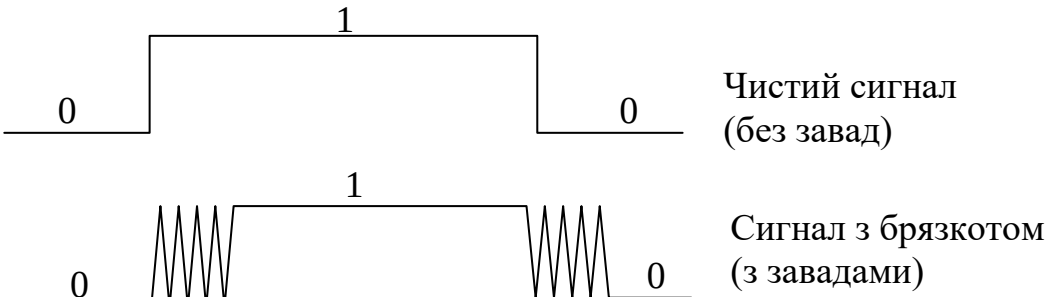


Рис. 5.7. Брязкіт механічних перемикачів

Для виходу з положення необхідно використовувати стробувальний сигнал, що дозволяє синхронізацію. Його потрібно подавати з перемикача або кнопки, а синхросигнал підключати з генератора. Оскільки період синхросигналу значно менше часу натискання кнопки або перемикача, то за одне натискання (подачу сигналу дозволу синхронізації) або перемикання автомат може перейти в t + 1 стан багато разів, і ми не зможемо спостерігати його покрокову роботу. Тому потрібна схема, що формує на виході одиничний імпульс протягом одного синхротакту для вирішення синхронізації (внутрішній стрибок). Ця схема має також враховувати, що зовнішній стрибок сигналу, що подається з перемикача або кнопки, теж має брязкіт. Таким чином, необхідна антибрязкова схема для формування одиничного імпульсу (внутрішнього стрибка), що подається на автомат після натискання кнопки або перемикача (зовнішнього стрибка). На рис. 5.8 подана часова діаграма, що ілюструє цю ідею. Тут AD – антибрязкова схема, En – зовнішній стрибок, EnFSM – внутрішній стрибок.

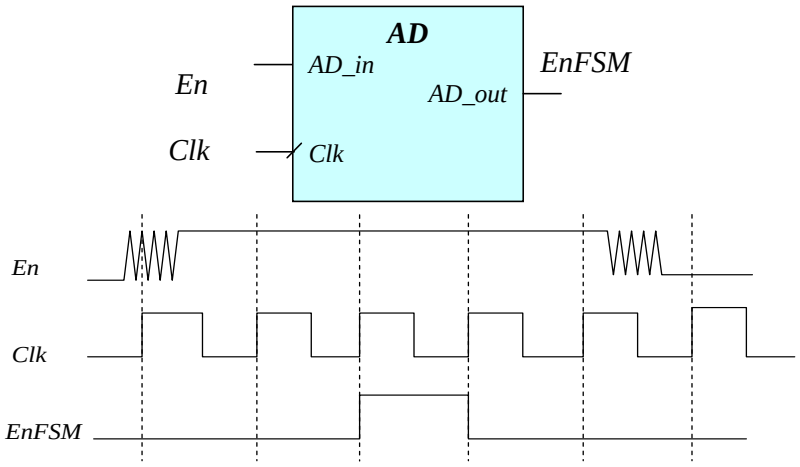


Рис. 5.8. Формування внутрішнього стрибка

Структура анибрязкової схеми і пояснюючі її роботу часові діаграми подані на рис. 5.9. Часові діаграми наведені з урахуванням затримок спрацьовування елементів.

Період синхроімпульсу має бути більше часу брязкоту. Якщо передній фронт синхроімпульсу Clk надійде в момент брязкоту, значення 0 або 1 буде збережено в тригері в момент t_1.

За наявності 0 одиниця буде отримана при наступному активному фронті синхроімпульсу (t_2). Тому сигнал S1 є безперешкодною і синхронізованою версією сигналу AD_in.

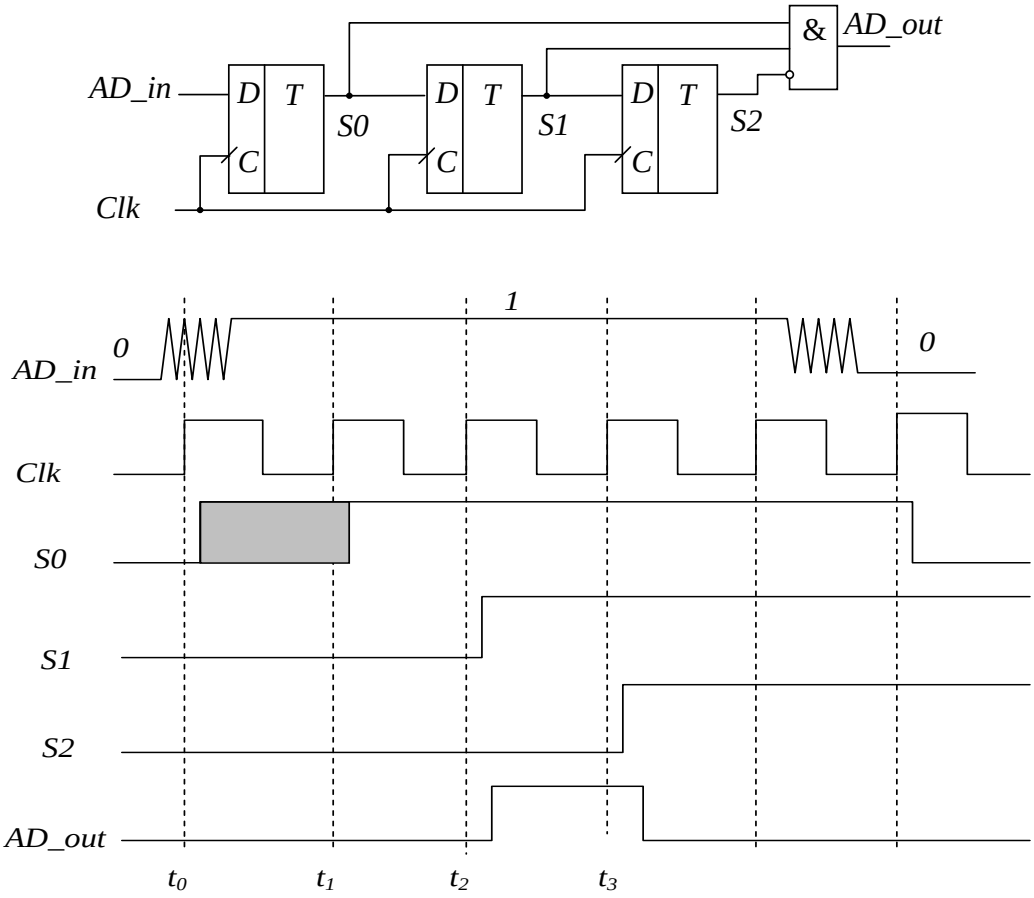


Рис. 5.9. Антибрязкова схема та її часові діаграми з урахуванням затримок

Проте можливість помилки існує, якщо зміна сигналу AD_in буде близькою до активного фронту синхроімпульсу, що призведе до порушення часу установлення і зберігання. У такому випадку вихід тригера може почати генерувати або видати неправильний результат. Хоча така ситуація зустрічається рідко, краще виключити її, додавши ще один тригер S2, який зберігає затримане на один такт значення вхідного сигналу і дозволяє виявити його зміну. Сигнал AD_out протягом одного синхротакту дорівнює 1, якщо на вході AD_in відбулася зміна значення з 0 на 1. На діаграмі сигнали S1 і S2 наведені для випадку, коли на S0 захоплюється 0 під час брязкоту.

VHDL-модель антибрязкової схеми подана в лістингу 5.3. На рис. 5.10 подано результат моделювання антибрязкової схеми.

Лістинг 5.3. VHDL-модель антибрязкової схеми

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity AD is
    Port (AD_in, clk: in  STD_LOGIC;
          AD_out: out STD_LOGIC);
end AD;
architecture AD of AD is
signal S0, S1, S2: STD_LOGIC;
begin
  PR1: process (clk)
  begin
    if clk = '1' and clk'event then
        S0 <= AD_in;
        S1 <= S0;
        S2 <= S1;
  end if;
  end process;
AD_out <=S0 and S1 and not S2;
-- формування керуючого імпульсу;
end AD;
```

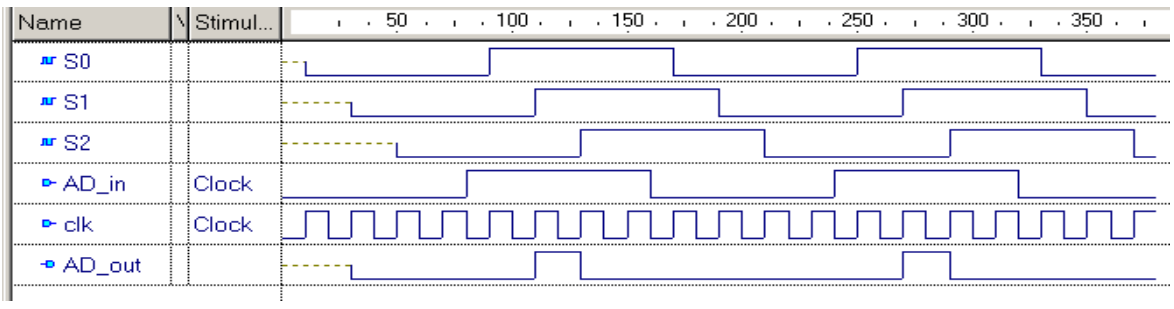


Рис. 5.10. Результат моделювання антибрязкової схеми

Наявність антибрязкової схеми не буде достатньою для моделювання роботи автомата на макеті. Це пов'язано з тим, що період синхроімпульсу набагато коротше брязкоту перемикача. Тому необхідно збільшити період синхроімпульсу (зменшити

частоту синхроімпульсу) до величини, яка відповідає тривалості брязкоту.

Перемикачі на платі Spartan-3E Starter Kit мають тривалість брязкіту $t_d = 2$ мс $= 2 \cdot 10^{-3}$ с, а частота синхроімпульсу, що генерується вбудованим у плату генератором, становить $h_{clk} = 50$ МГц $= 50 \cdot 10^6$ Гц. Період синхроімпульсу визначається як $1/ h_{clk} = 1/(50 \cdot 10^6) = 0{,}02 \cdot 10^{-6}$ с $= 20 \cdot 10^{-9}$ с $= 20$ нс.

Щоб визначити, у скільки разів необхідно збільшити період *clk*, необхідно розділити t_d / $t_{clk} = (2 \cdot 10^{-3})/(20 \cdot 10^{-9}) = 10^5$. Тобто в 100000 раз чи більше. Для реалізації збільшення періоду (зменшення частоти) *clk* необхідно скористатися подільником частоти, виконаному на лічильнику (рис. 5.11).

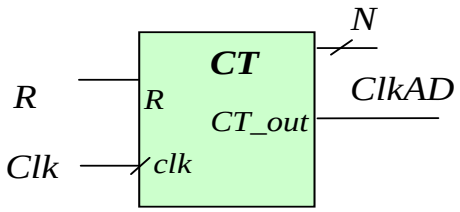


Рис. 5.11. Подільник частоти на лічильнику

Визначимо розрядність лічильника *N*, якщо коефіцієнт перерахунку лічильника має бути більше 100000. $2^{17} = 131072$, $2^{16} = 65536$, отже, $N = 17$. У такому випадку період буде дорівнювати $t_{clk} \approx 2{,}62$ мс, що більше $t_d = 2$ мс.

VHDL-модель подільника частоти подана в лістингу 5.4.

Лістинг 5.4. VHDL-модель подільника частоти

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
entity CT is
    Port (R, clk: in  STD_LOGIC;
  CT_out: out STD_LOGIC);
end CT;
architecture CT of CT is
constant N: INTEGER := 17;
signal    COUNT_INT:    STD_LOGIC_VECTOR(N-1
downto 0);
```

```
    begin
    process (CLK, R)
    begin
      if R = '1' then
        COUNT_INT <= (others => '0');
      elsif CLK'event and CLK='1' then
        COUNT_INT <= COUNT_INT + 1;
       end if;
    end process;
    CT_out<='1' when COUNT_INT(N-1)= '1' else
'0';
    end CT;
```

Замість `CT_out<='1' when COUNT_INT(N-1)= '1' else '0';` можна було написати `CT_out<= COUNT_INT(N-1).`

На рис. 5.12 подано результати моделювання подільника частоти для $N = 4$ замість $N = 17$ для наочності.

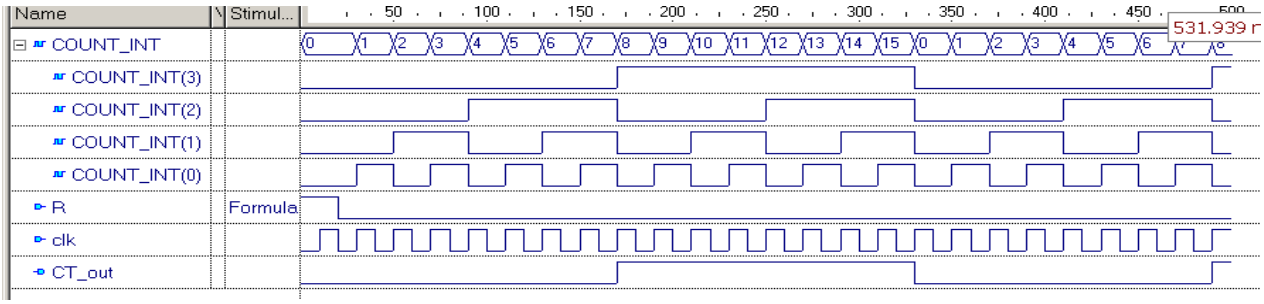


Рис. 5.12 Часові діаграми для подільника частоти з перерахунковим коефіцієнтом 16

Для того щоб скористатися антибрязковою схемою і подільником частоти при створенні проєкту автомата, необхідно передбачити в блоці, що описує власне автомат, ще один керуючий вхід *En*. Розглянемо як приклад автомат Мура (рис. 5.13). VHDL-модель автомата Мура подана в лістингу 5.5. Керуючий вхід *En* описується в блоці формування послідовнісної частини. Синхронізується автомат заднім фронтом.

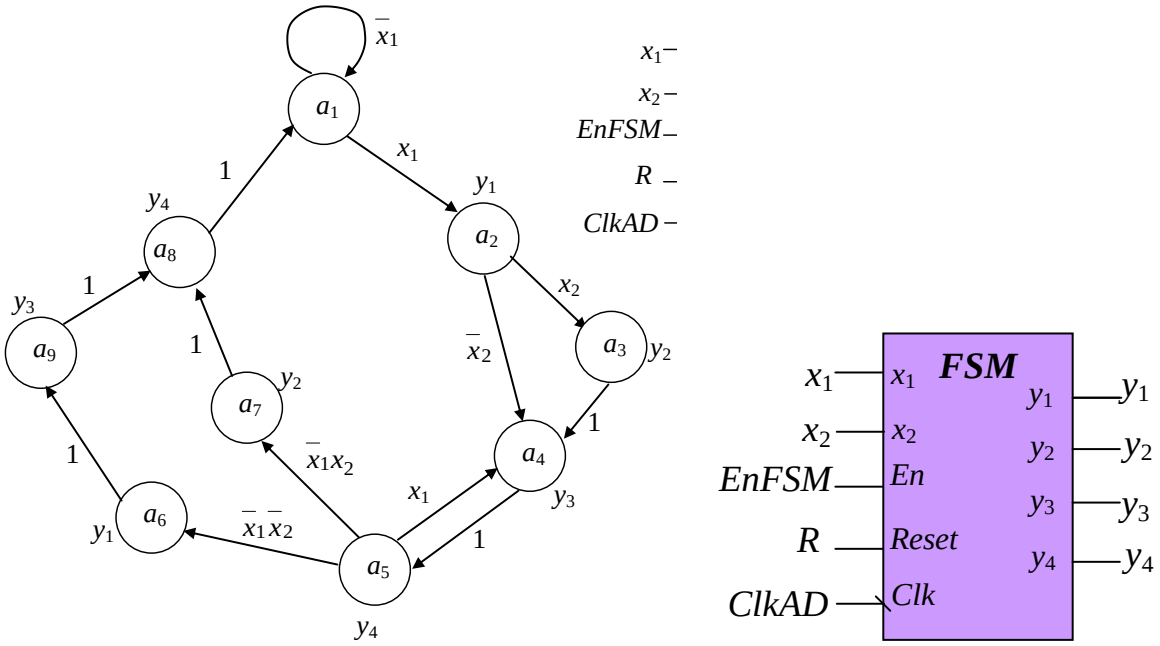


Рис. 5.13. Граф переходів автомата Мура

Лістинг 5.5. VHDL-модель автомата Мура

```
    library IEEE;
    use IEEE.std_logic_1164.all;
-- Опис інтерфейсу пристрою
    entity FSM is
      port ( Clk: in STD_LOGIC;
        Reset, En: in STD_LOGIC;
        x1, x2: in STD_LOGIC;
        y1, y2, y3, y4: out STD_LOGIC);
    end;
-- Опис архітектури пристрою
    architecture FSM of FSM is
      type State_type is (a1, a2, a3, a4, a5, a6,
a7, a8, a9);
      signal State, NextState: State_type;
    begin
-- Блок для формування послідовних частини
    Sreg0_CurrentState: process (Clk, reset)
    begin
      if Reset='1' then
        State <= a1;
      elsif Clk'event and Clk = '0' then
```

```
        if En='1'   then
        State <= NextState;
        end if;
      end if;
    end process;
-- Блок для формування комбінаційної частини
-- Опис переходів станів за умовами
    Sreg0_NextState: process (State, x1, x2)
    begin
      case State is
        when  a1=>  if  x1='1'  then  NextState  <=
a2;
                           else NextState <= a1;
                  end if;
        when a2=> if x2='1' then  NextState   <=
a3;
                            elseNextState <= a4;
                     end if;
        when a3=> NextState <= a4;
        when a4=> NextState <= a5;
        when  a5=>  if  x1='1'  then  NextState  <=
a4;
                elsif x2='1' then  NextState   <=
a7;
                  else  NextState <= a6;
                  end if;
          when a6=> NextState <= a9;
          when a7=> NextState <= a8;
            when a8=> NextState <= a1;
            when a9=> NextState <= a8;
        when others => NextState <= a1;
      end case;
    end process;
-- Опис вихідних сигналів
    y1<='1' when State=a2 or State=a6 else
      '0';
    y2<='1' when State=a3 or State=a7 else
      '0';
    y3<='1'  when  State=a4  or  State=a9  else
      '0';
```

```
y4<='1'  when  State=a5  or  State=a8  else
  '0';
end;
```

На рис. 5.14 подано часові діаграми роботи автомата Мура.

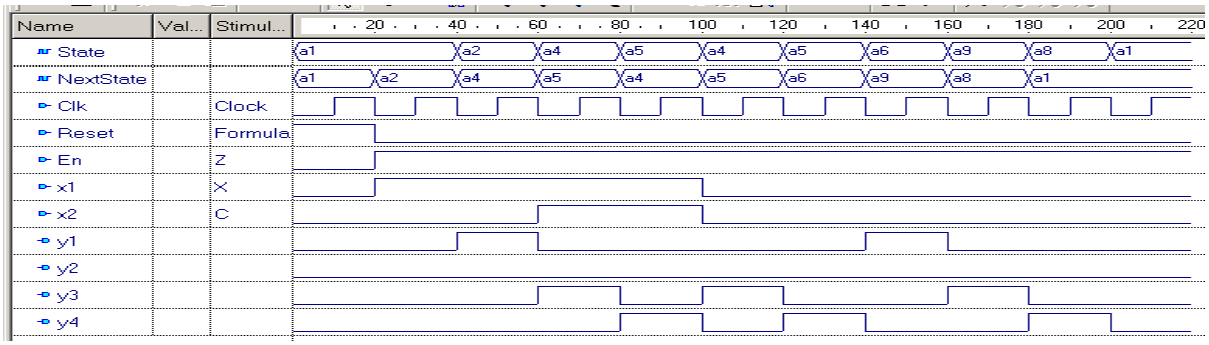


Рис. 5.14. Часові діаграми роботи автомата Мура

Повністю проєкт автомата Мура буде виглядати як на рис. 5.15.

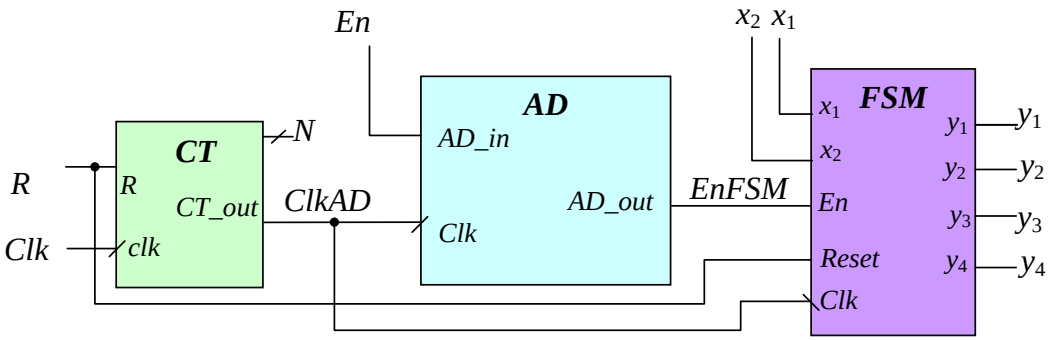


Рис. 5.15. Структурна схема для автомата Мура

Збільшена схема для автомата Мура подана на рис. 5.16, VHDL-модель збільшеної схеми автомата Мура подана в лістингу 5.6.

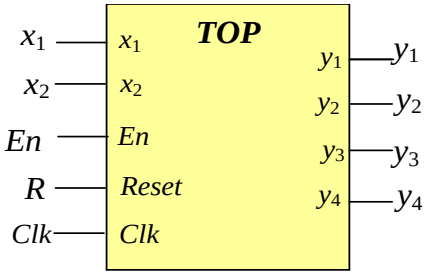


Рис. 5.16. Збільшена схема для автомата Мура

Лістинг5.6. VHDL-модель збільшеної схеми автомата Мура

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity top is
   port(
     R : in STD_LOGIC;
     CLK : in STD_LOGIC;
     En : in STD_LOGIC;
     x1 : in STD_LOGIC;
     x2 : in STD_LOGIC;
     y1 : out STD_LOGIC;
     y2 : out STD_LOGIC;
     y3 : out STD_LOGIC;
     y4 : out STD_LOGIC);
end top;
architecture top of top is
component CT
    Port (R, clk: in  STD_LOGIC;
  CT_out: out STD_LOGIC);
end component;
component AD
    Port (AD_in, clk: in  STD_LOGIC;
  AD_out: out STD_LOGIC);
end component;
component FSM
  port ( Clk: in STD_LOGIC;
    Reset, En: in STD_LOGIC;
    x1, x2: in STD_LOGIC;
    y1, y2, y3, y4: out STD_LOGIC);
end component;
signal EnFSM: STD_LOGIC;
signal ClkAD: STD_LOGIC;
begin
U1:CT
     port     map     (R=>R,     clk=>CLK,
CT_out=>ClkAD );
U2:AD
   port   map   (AD_in=>En,   clk=>ClkAD,
```

```
AD_out=>EnFSM );
    U3:FSM
        port    map    (Clk=>ClkAD,    Reset=>R,
En=>EnFSM, x1=>x1,
             x2=>x2,  y1=>y1,  y2=>y2,  y3=>y3,
y4=>y4);
    end top;
```

У процесі синтезу проєкту внутрішні стани були закодовані:

```
    Analyzing FSM <FSM_0> for best encoding.
    Optimizing  FSM  <U3/State/FSM>  on  signal
<State[1:4]> with gray encoding.
    -------------------
    State | Encoding
    -------------------
    a1    | 0000   a2    | 0001
    a3    | 0011   a4    | 0010
    a5    | 0110   a6    | 0101
    a7    | 0111   a8    | 1100
    a9    | 0100
    -------------------
```

Контрольні запитання

1. Що таке ПЛІС? У чому її відмінності від звичайних інтегральних схем?
2. Яка архітектура типової багаторівневої системи управління?
3. У чому відмінність програмувального мікроконтролера від комп'ютера загального призначення та промислового комп'ютера?
4. Чим відрізняється процесор від мікропроцесора?
5. Наведіть основні принципи архітектури фон Неймана.
6. У чому полягає принципова відмінність гарвардської архітектури від архітектури фон Неймана?
7. Наведіть переваги та недоліки архітектур фон Неймана та гарвардської.

6. ОПЕРАЦІЙНІ АВТОМАТИ

6.1. VHDL-модель операційного автомата

Перш ніж розглянути VHDL-модель операційного автомата, згадаємо алгоритм додавання цілих двійкових чисел A + B = C в додаткових кодах. Доповнимо змістовну ДСА цього алгоритму (рис. 5.5) однією операторною вершиною в кінці (C: = Sm), як показано на рис. 6.1. ДСА, подана тут, розмічена на кшталт автомата Мура.

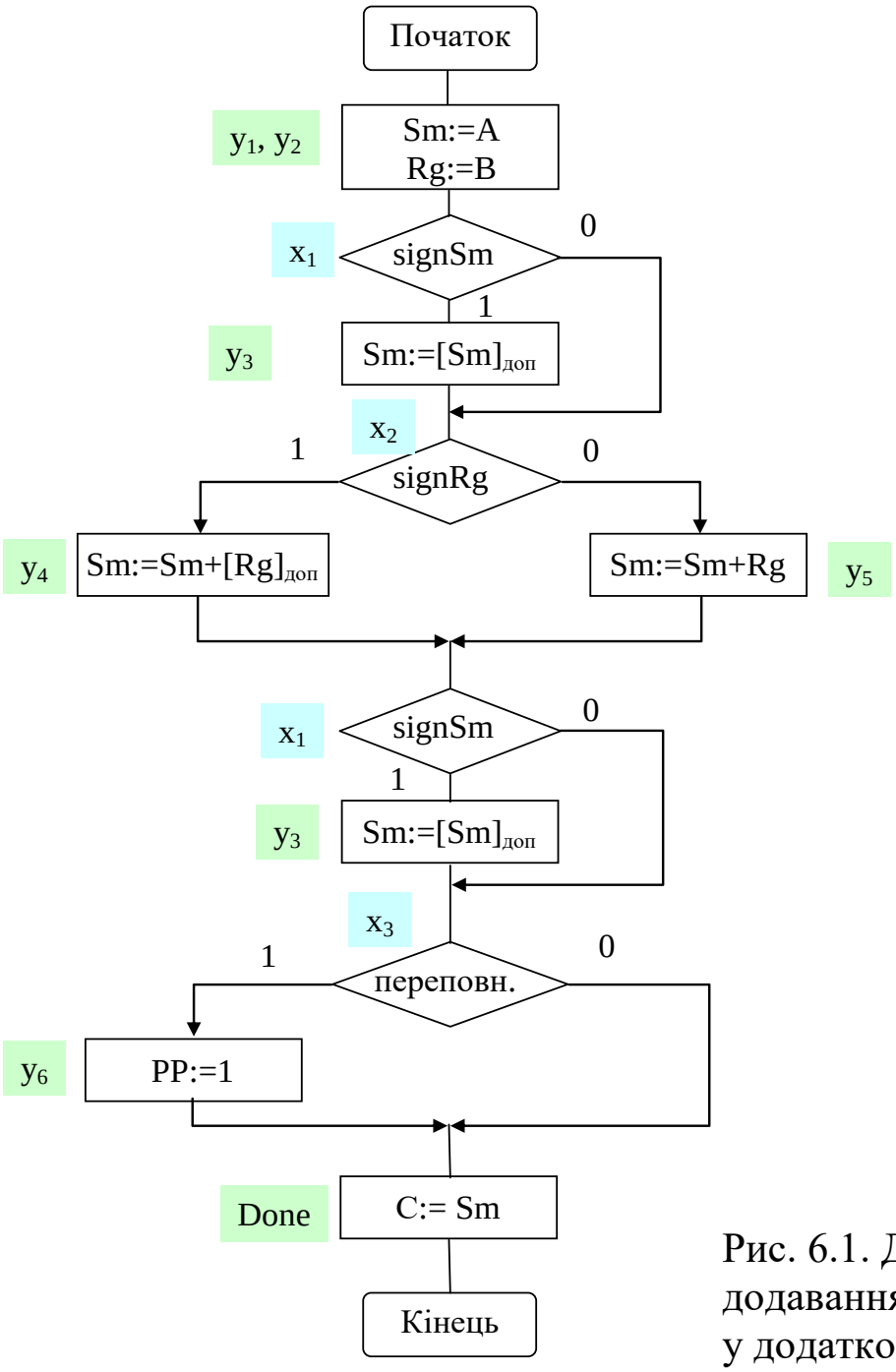


Рис. 6.1. ДСА алгоритму додавання цілих чисел у додаткових кодах

Для додавання використовуються шестирозрядні модифіковані коди, у яких два лівих розряди кодують знак. У табл. 6.1 наведено варіанти знаків, поданих у модифікованому вигляді. У табл. 6.2 подано приклади додавання двійкових цілих чисел для різних випадків переповнення і без переповнення.

Таблиця 6.1

Варіанти знаків, поданих у модифікованому вигляді

Знакові розряди	Сенс	Примітка
00	+	Додатний знак
01	ПП+	Переповнення додатне
10	ПП–	Переповнення від'ємне
11	–	Від'ємний знак

Таблиця 6.2

Приклади додавання двійкових цілих чисел у додаткових кодах

Десяткові доданки	Доданки в прямих кодах	Доданки в зворотних кодах	Додавання в додаткових кодах	Результат у прямих кодах
-8 -9	11 1000 11 1001	11 0111 11 0110	11 1000 + 11 0111 1|10 1111	ПП–
+9 +12	00 1001 00 1100	00 1001 00 1100	00 1001 + 00 1100 01 0101	ПП+
+7 +3	00 0111 00 0011	00 0111 00 0011	00 0111 + 00 0011 00 1010	+10
-2 -3	11 0010 11 0011	11 1101 11 1100	11 1110 + 11 1101 1|11 1011	11 0100 обр. + 1 11 0101 прям. -5
+5 -8	00 0101 11 1000	00 0101 11 0111	00 0101 + 11 1000 11 1101	11 0010 обр. + 1 11 0011 прям. -3

На рис. 6.2 зображена узагальнена схема операційного модуля даного пристрою. На рис. 6.3 зображений граф переходів керуючого автомата Мура для даного пристрою.

A(5:0) і *B*(5:0) – доданки, *C*(5:0) – сума, *Done* – ознака кінця обчислення суми (=1), *pp* – ознака переповнення (=1).

У лістингу 6.1 наведена VHDL-модель операційного модуля (автомата).

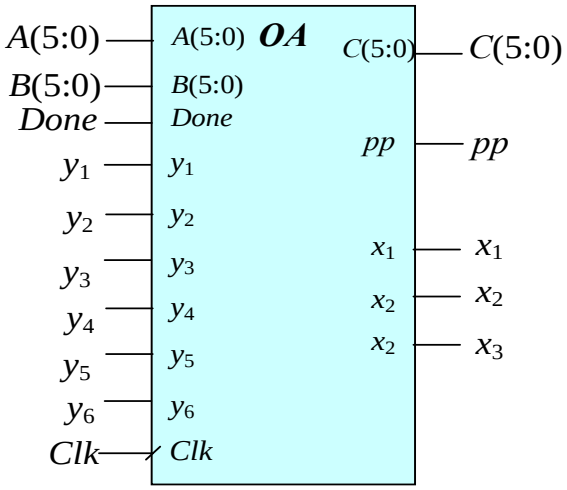


Рис. 6.2. Операційний модуль додавання цілих чисел у додаткових кодах

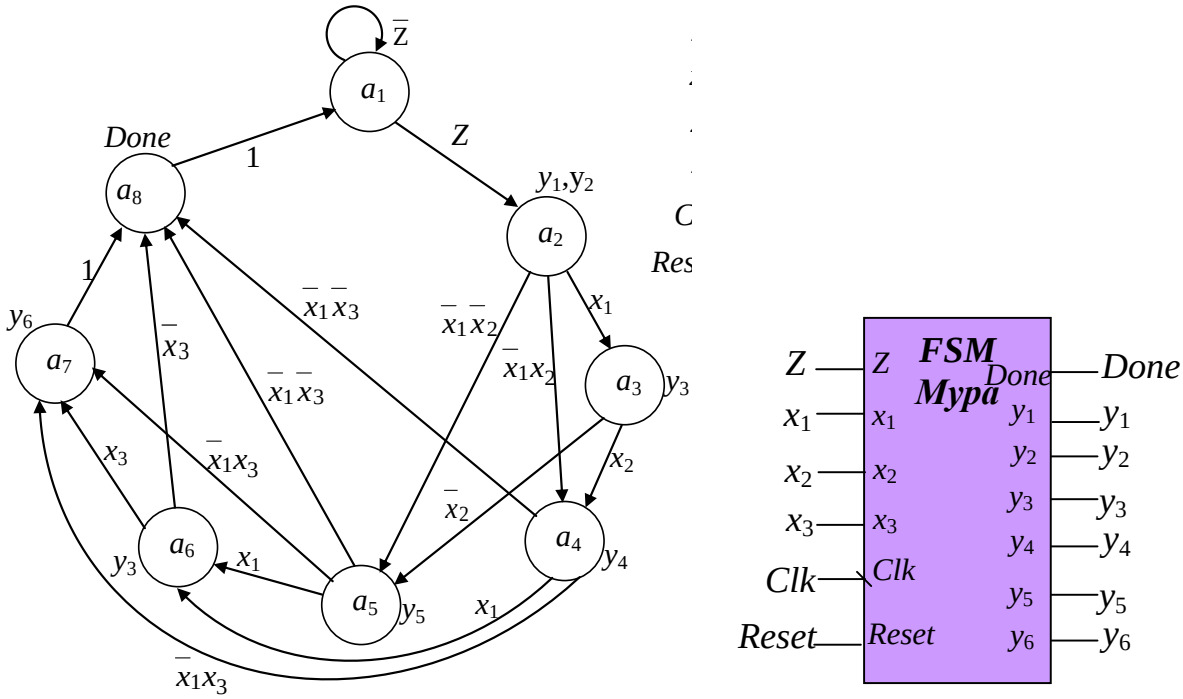


Рис. 6.3. Керуючий автомат Мура пристрою додавання цілих чисел у додаткових кодах

Лістинг 6.1. VHDL-модель операційного модуля

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_unsigned.all;
entity OA is
   port(
    Done, y1, y2, y3, y4, y5, y6, CLK: in
STD_LOGIC;
 -- control signals
     A : in STD_LOGIC_VECTOR(5 downto 0);
--  summand
     B : in STD_LOGIC_VECTOR(5 downto 0);
-- summand
     C : out STD_LOGIC_VECTOR(5 downto 0);
-- sum
    x1, x2, x3 : out STD_LOGIC; --
informative signals
     pp: out STD_LOGIC    -- overflow    );
end OA;
--}} End of automatically maintained
section
architecture OA of OA is
   -- enter your statements here --
signal RG:STD_LOGIC_VECTOR(5 downto 0); --
   - register
signal SM:STD_LOGIC_VECTOR(5 downto 0); --
register-adder
begin
   process (CLK, y1, y2, y3, y4, y5, y6)
   begin
   if (CLK'event and CLK='1')then
     If (y1 = '1' and y2 = '1') then
       RG <=B;
       SM <=A;
     elsif (y3 = '1')then
        M <=SM(5 downto 4) & ((not SM(3
downto 0))+1);
     elsif (y4 = '1')then
     SM <= SM + (RG(5 downto 4) & ((not
RG(3 downto 0))+1));
```

```
            elsif (y5 = '1')then
               SM <= SM + RG;
            elsif (y6 = '1')then
               pp <= '1';
              else pp <= '0';
            end if;
         end if;
    end process;
         C <= SM when Done = '1' else (others
=>'0');
         x1<= '1'when SM(5 downto 4)="11" else
'0';
         x2<= '1'when RG(5 downto 4)="11" else
'0';
         x3<= '1'when SM(5 downto 4)="01" or
SM(5 downto 4)="10" else '0';
    end OA;
```

На рис. 6.4 наведено часові діаграми роботи операційного автомата. У лістингу 6.2 наведена VHDL-модель керуючого автомата (рис. 6.3) Мура.

На рис. 6.5 наведено часові діаграми роботи операційного автомата.

Загальна схема пристрою наведена на рис. 6.6.

Збільшена схема пристрою подана на рис. 6.7.

У лістингу 6.3 наведена VHDL-модель пристрою додавання (рис. 6.7, 6.8).

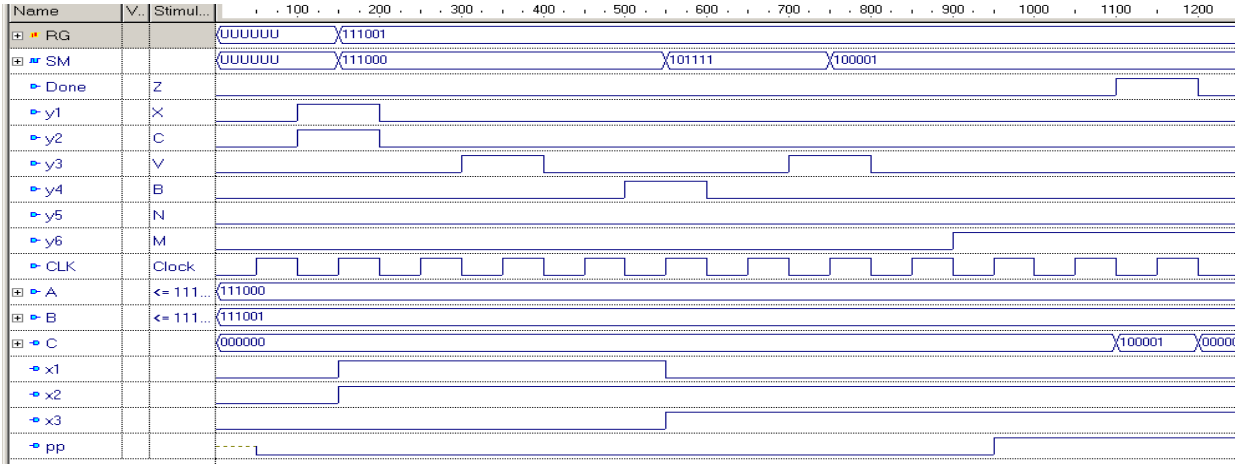


Рис. 6.4. Часові діаграми роботи операційного автомата

Лістинг 6.2. VHDL-модель керуючого автомата Мура

```
    library IEEE;
    use IEEE.std_logic_1164.all;
    -- Опис інтерфейсу пристрою
    entity FSM is
      port ( Clk, Reset, Z: in STD_LOGIC;
        x1, x2, x3: in STD_LOGIC;
        Done,  y1,  y2,  y3,  y4,y5,  y6:  out
STD_LOGIC);
    end;
    -- Опис архітектури пристрою
    architecture FSM of FSM is
      type State_type is (a1, a2, a3, a4, a5,
a6, a7,a8);
      signal State, NextState: State_type;
    begin
    -- Блок для формування послідовних частини
    Sreg0_CurrentState: process (Clk, reset)
    begin
      if Reset='1' then
        State <= a1;
      elsif Clk'event and Clk = '0' then
        State <= NextState;
      end if;
    end process;
    --  Блок  для  формування  комбінаційної
частини
    -- Опис переходів станів за умовамим
    Sreg0_NextState:  process  (State,  x1,  x2,
x3, Z)
    begin
      case State is
        when a1=> if Z='1' then  NextState  <=
a2;
                          else NextState <= a1;
                end if;
        when a2=> if x1='1' then  NextState  <=
a3;
```

```
                    elsif x2='1' then   NextState
<= a4;
                  else  NextState <= a5;
                  end if;
        when a3=> if x2='1' then  NextState   <=
a4;
                           elseNextState <= a5;
                    end if;
        when a4=> if x1='1' then  NextState   <=
a6;
                    elsif x3='1' then   NextState
<= a7;
                  else  NextState <= a8;
                  end if;
        when a5=> if x1='1' then  NextState   <=
a6;
                    elsif x3='1' then   NextState
<= a7;
                  else  NextState <= a8;
                  end if;
        when a6=> if x3='1' then  NextState   <=
a7;
                           elseNextState <= a8;
                    end if;
        when a7=> NextState <= a8;
        when a8=> NextState <= a1;
        when others => NextState <= a1;
      end case;
    end process;
    -- Опис вихідних сигналів
    y1<='1' when State=a2 else '0';
    y2<='1' when State=a2  else   '0';
    y3<='1' when State=a3 or State=a6 else '0';
    y4<='1' when State=a4   else '0';
    y5<='1' when State=a5  else   '0';
    y6<='1' when State=a7   else '0';
    Done<='1' when State=a8  else '0';
    end;
```

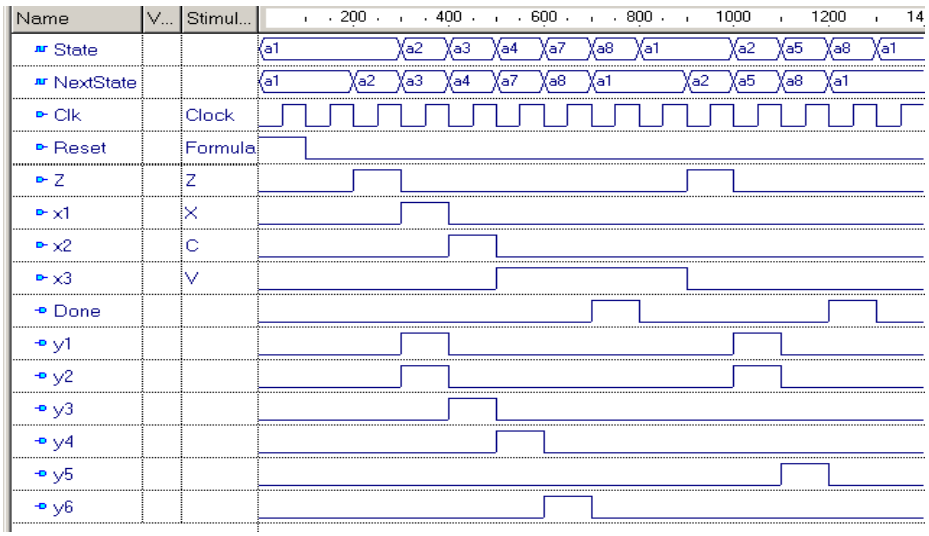


Рис. 6.5. Часові діаграми роботи операційного автомата Мура

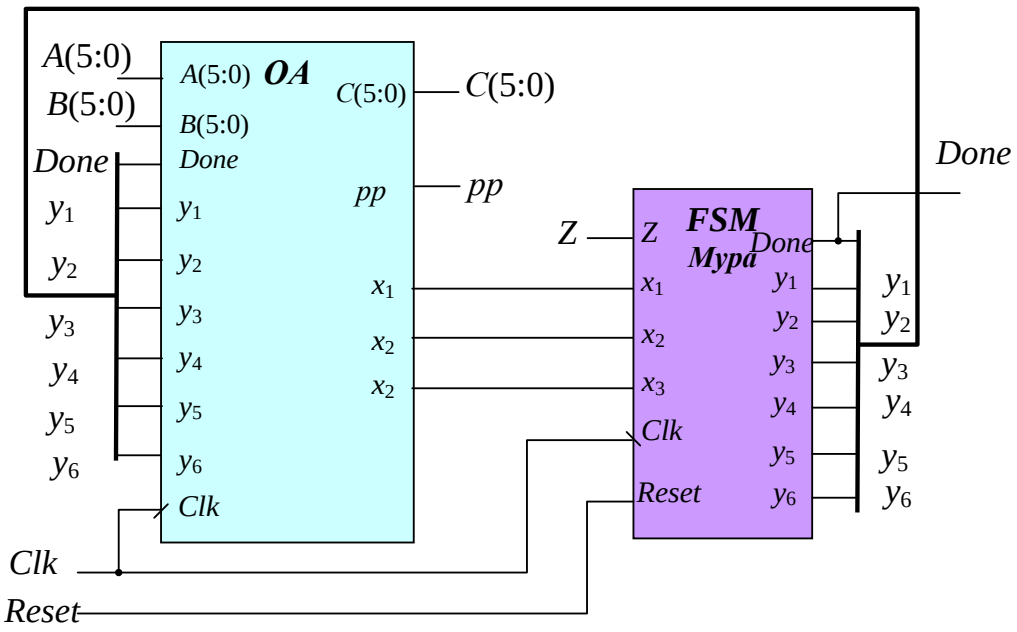


Рис. 6.6. Загальна схема пристрою додавання з УА Мура

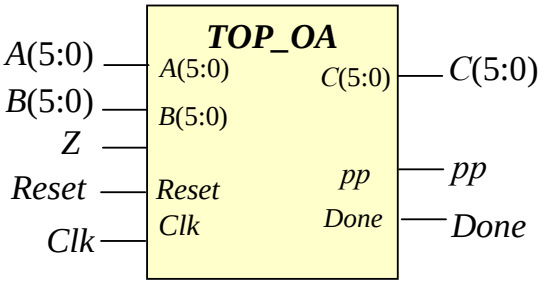


Рис. 6.7. Збільшена схема пристрою додавання

Лістинг 6.3. VHDL-модель пристрою додавання

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity TOP_OA is
   port(
    Clk, Reset, Z : in STD_LOGIC;
    A , B: in STD_LOGIC_VECTOR(5 downto 0);
    C  :  out  STD_LOGIC_VECTOR(5  downto  0);

    Done, pp : out STD_LOGIC );
end TOP_OA;
architecture TOP_OA of TOP_OA is
component OA is
   port(Done, y1, y2, y3, y4, y5, y6, CLK:
in STD_LOGIC; -- control signals
     A : in STD_LOGIC_VECTOR(5 downto 0); -
-  summand
     B  :  in  STD_LOGIC_VECTOR(5  downto  0);
-- summand
     C : out STD_LOGIC_VECTOR(5 downto 0);
-- sum
     x1,   x2,   x3:   out   STD_LOGIC;   --
informative signals
     pp:  out STD_LOGIC);
end component;
component FSM is
  port ( Clk, Reset, Z: in STD_LOGIC;
    x1, x2, x3: in STD_LOGIC;
    Done,  y1,  y2,  y3,  y4,y5,  y6:  out
STD_LOGIC);
end component;
signal x1,x2,x3,y1,y2,y3,y4,y5,y6, Don_int:
STD_LOGIC;
begin
U1: OA port map (Done=> Don_int, y1=>y1,
y2=>y2, y3=>y3, y4=>y4, y5=>y5,
y6=>y6, CLK=>CLK, A=>A, B=>B, C=>C, x1=>x1,
x2=>x2, x3=>x3, pp=>pp);
```

```
    U2: FSMport map( Clk=> Clk, Reset=>Reset,
Z=>Z, x1=>x1, x2=>x2, x3=>x3, Done=> Don_int,
y1=>y1,   y2=>y2,   y3=>y3,   y4=>y4,   y5=>y5,
y6=>y6);
    Done <= Don_int;
    end TOP_OA;
```

На рис. 6.8 наведено часові діаграми роботи пристрою додавання.

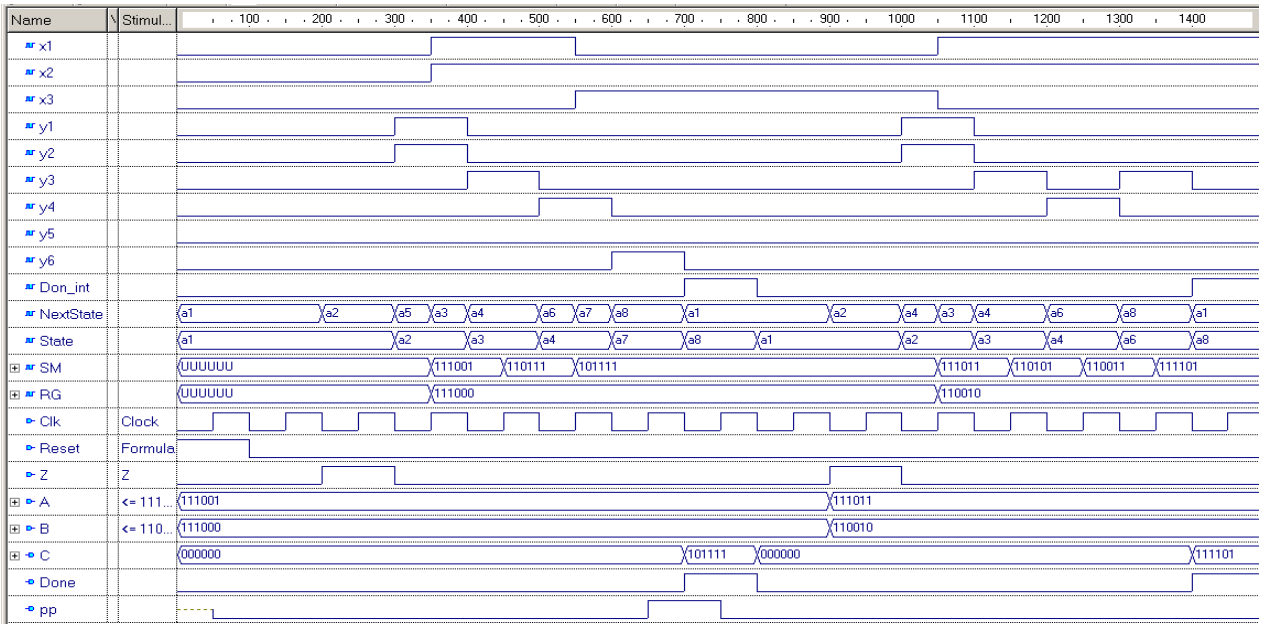


Рис. 6.8. Часові діаграми роботи пристрою додавання з УА Мура

У цьому прикладі керуючий автомат працює по задньому фронту, а операційний автомат по передньому. Таким чином, повний цикл взаємодії УА з ОА відбувається за один такт синхросигналу *T* (рис. 6.9). Коли керуючий автомат працює по задньому фронту, а операційний автомат по передньому, це не завжди добре, оскільки може знижуватися частота синхросигналу.

Щоб даний пристрій спрацьовував по одному фронту, необхідно використовувати стробувальний сигнал *En*, частота якого буде у два рази менше частоти синхросигналу. Тоді один такт працює УА, а ОА чекає, інший такт, навпаки, працює ОА, а УА чекає. Таким чином, повний цикл взаємодії УА з ОА відбувається за два такти синхросигналу (рис. 6.10).

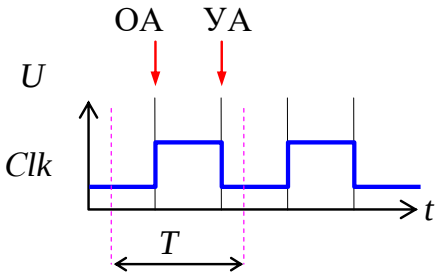


Рис. 6.9. Повний цикл взаємодії УА з ОА для автомата Мура

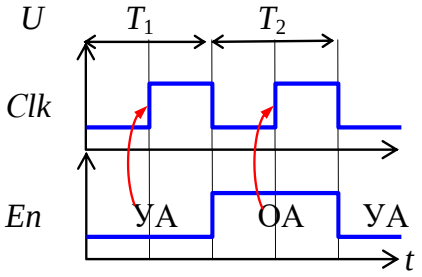


Рис. 6.10. Часові діаграми при використанні сигналу *En*

Якщо сигнал *En* описується в ОА по 1, то в УА він має описуватися по 0. Тобто в ОА використовується конструкція

```
elsif Clk'event and Clk = '1' then
  if En='1'   then
```

В УА використовується конструкція

```
elsif Clk'event and Clk = '1' then
  if En='0'   then
```

Щоб повний цикл взаємодії УА з ОА відбувався за один такт синхросигналу, і вони обидва спрацьовували по одному фронту, необхідно як керуючий автомат використовувати автомат Мілі. На рис. 6.11 подано граф переходів автомата Мілі для пристрою додавання в додаткових кодах (рис. 6.1). У лістингу 6.4 наведена VHDL-модель керуючого автомата (рис. 6.11) Мілі.

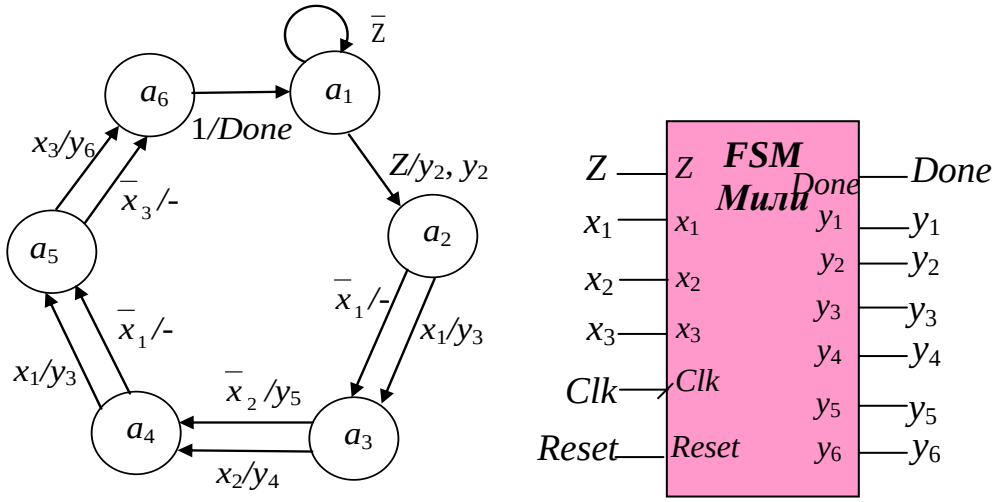


Рис. 6.11. Керуючий автомат Мілі пристрою додавання цілих чисел у додаткових кодах

Лістинг 6.4. VHDL-модель керуючого автомата Мілі

```
    library IEEE;
    use IEEE.std_logic_1164.all;
    -- Опис интерфейсу пристрою
    entity FSM is
      port ( Clk, Reset, Z: in STD_LOGIC;
        x1, x2, x3: in STD_LOGIC;
        Done,  y1,  y2,  y3,  y4,y5,  y6:  out
STD_LOGIC);
    end;
    -- Опис архітектури пристрою
    architecture FSM of FSM is
      type State_type is (a1, a2, a3, a4, a5,
a6);
      signal State, NextState: State_type;
    begin
    Sreg0_Cu
    -- Блок для формування послідовних частин
    rrentState: process (Clk, reset)
    begin
      if Reset='1' then
        State <= a1;
      elsif Clk'event and Clk = '1' then
        State <= NextState;
      end if;
    end process;
    --  Блок  для  формування  комбінаційної
частини
    --  Опис  переходів  станів  за  умовами  і
вихідними сигналами
    Sreg0_NextState:  process  (State,  x1,  x2,
x3, Z)
    begin
y1<='0';  y2<='0';  y3<='0';  y4<='0';  y5<='0';
y6<='0'; Done <='0';
      case State is
      when a1=> if Z='1' then NextState <= a2;
y1<='1'; y2<='1';
                          else NextState <= a1;
                end if;
```

```
        when a2=> if x1='1' then  NextState   <=
a3; y3<='1';
                   else  NextState <= a3;
                  end if;
        when a3=> if x2='1' then  NextState   <=
a4; y4<='1';
                   else NextState <= a4; y5<='1';
                     end if;
        when a4=> if x1='1' then  NextState   <=
a5; y3<='1';
                  else  NextState <= a5;
                  end if;
        when a5=> if x3='1' then  NextState   <=
a6; y6<='1';
                  else  NextState <= a6;
                  end if;
        when a6=> NextState <= a1; Done<='1';
        when others => NextState <= a1;
      end case;
    end process;
    end;
```

На рис. 6.12 наведено часові діаграми роботи пристрою додавання з УА Мілі. Загальна схема пристрою складання подана на рис. 6.13.

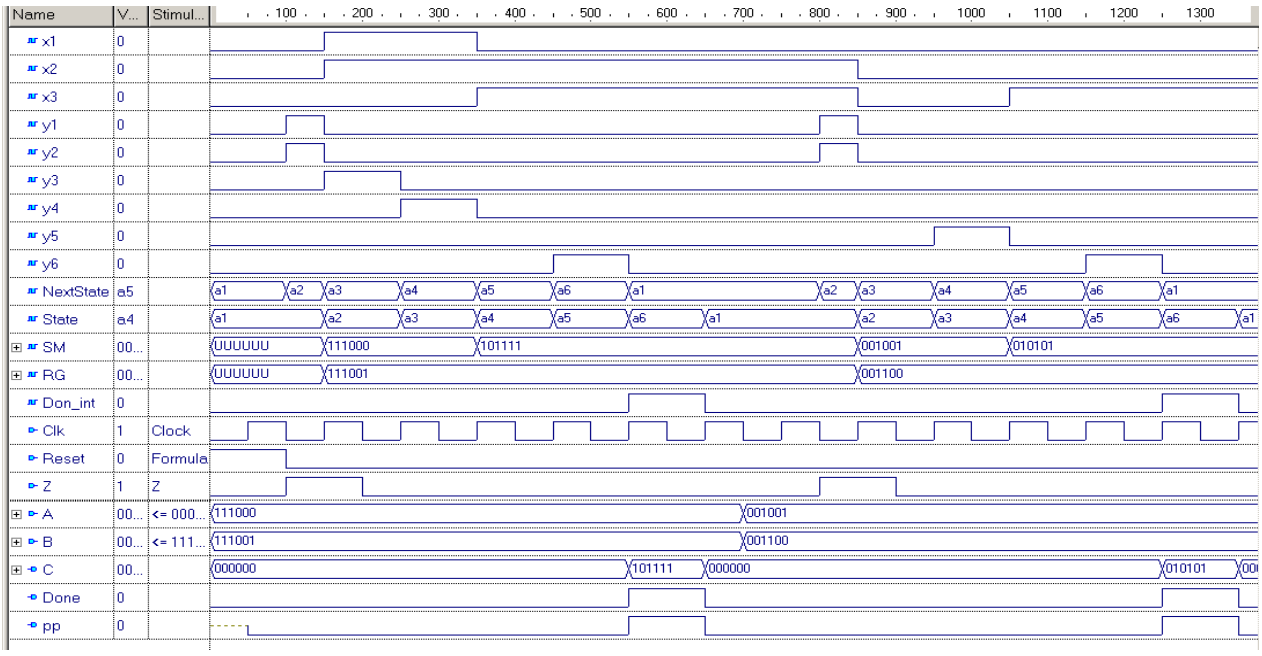


Рис. 6.12. Часові діаграми роботи пристрою додавання з УА Мілі

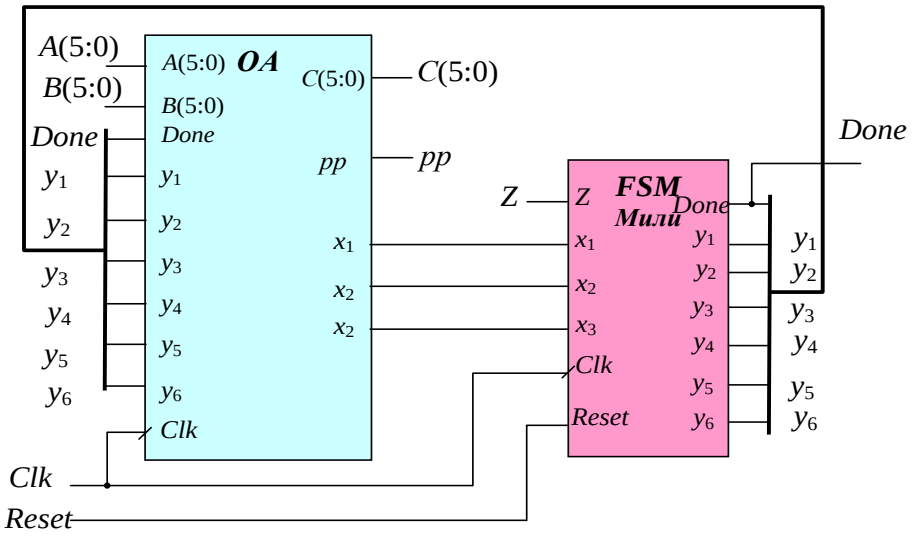


Рис. 6.13. Загальна схема пристрою додавання з УА Мілі

Повний цикл взаємодії УА з ОА відбувається за один такт синхросигналу, обидва автомати спрацьовують по передньому фронту (рис. 6.14).

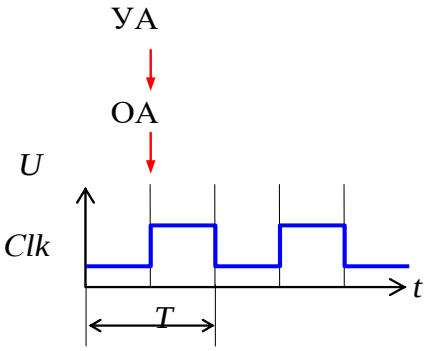


Рис. 6.14. Повний цикл взаємодії УА з ОА для автомата Мілі

6.2. Синтез канонічної структури операційного автомата

Структуру операційного автомата можна синтезувати безпосередньо за функцією, заданою:

1) множиною слів $S = \{s_1, \ldots, s_m, \ldots s_M\}$, які можуть бути вхідними, вихідними і внутрішніми.

Для прикладу на рис. 6.1 множина вхідних слів складається з $A(5:0)$ і $B(5:0)$, множина вихідних слів складається з $C(5:0)$, множина внутрішніх слів складається з $Sm(5:0)$, $Rg(5:0)$ і pp;

2) множиною мікрооперацій $Y = \{y_1, \ldots, y_g, \ldots y_G\}$.

Для прикладу на рис. 6.1 множина мікрооперацій складається з y_1, y_2, y_3, y_4, y_5, y_6, Done;

3) множиною логічних умов $X = \{x_1, \ldots, x_f, \ldots x_F\}$. Для прикладу на рис. 6.1 множина мікрооперацій складається з логічних умов x_1, x_2, x_3.

Структура автомата синтезується так.

1. Словам, описаним як внутрішні, ставляться у відповідність регістри з довжинами, рівними довжинами слів.

Sm(5: 0) – шестирозрядний регістр, *Rg* (5: 0) – шестирозрядний регістр, *pp* - тригер.

2. Словам, описаним як вхідні, ставляться у відповідність зовнішні входи схеми, які з'єднуються з регістрами шиною, що виходить з входів.

3. Словам, описаним як вихідні, ставляться у відповідність зовнішні виходи схеми, які з'єднуються з регістрами шиною, що входить у виходи.

4. Кожній мікрооперації $y_i \in Y$, описуваній оператором присвоєння (: =), ставиться у відповідність комбінаційна схема $\varphi_i \in \varphi$ (рис. 5.2), входи якої підключаються до регістрів і виходи якої з'єднуються з керованою шиною регістра. Керована шина позначається сигналом y_i, який ініціює мікрооперацію – присвоєння слову значення деякої функції. Для виконання мікрооперації передачі не потрібна комбінаційна схема, що обчислює значення деякої функції. Тому структурна реалізація мікрооперації передачі забезпечується керованою шиною, що з'єднує регістр з регістром, і позначеною відповідним керуючим сигналом. Аналогічно мікрооперації установлення (*S*:=const) реалізується керованою шиною, початок якої відмічається константою const і відповідним керуючим сигналом.

5. Кожній логічній умові x_f ставиться у відповідність комбінаційна схема $\psi_f \in \psi$, входи якої з'єднуються з регістром, а виходи відзначаються інформаційним сигналом x_f. Якщо ψ_f тривіальна функція, яка, наприклад, показує вміст розряду (розрядів) регістра, то логічна умова зображується шиною. Вихід шини відповідає інформаційному сигналу.

Канонічна структура має вигляд як на рис. 6.15.

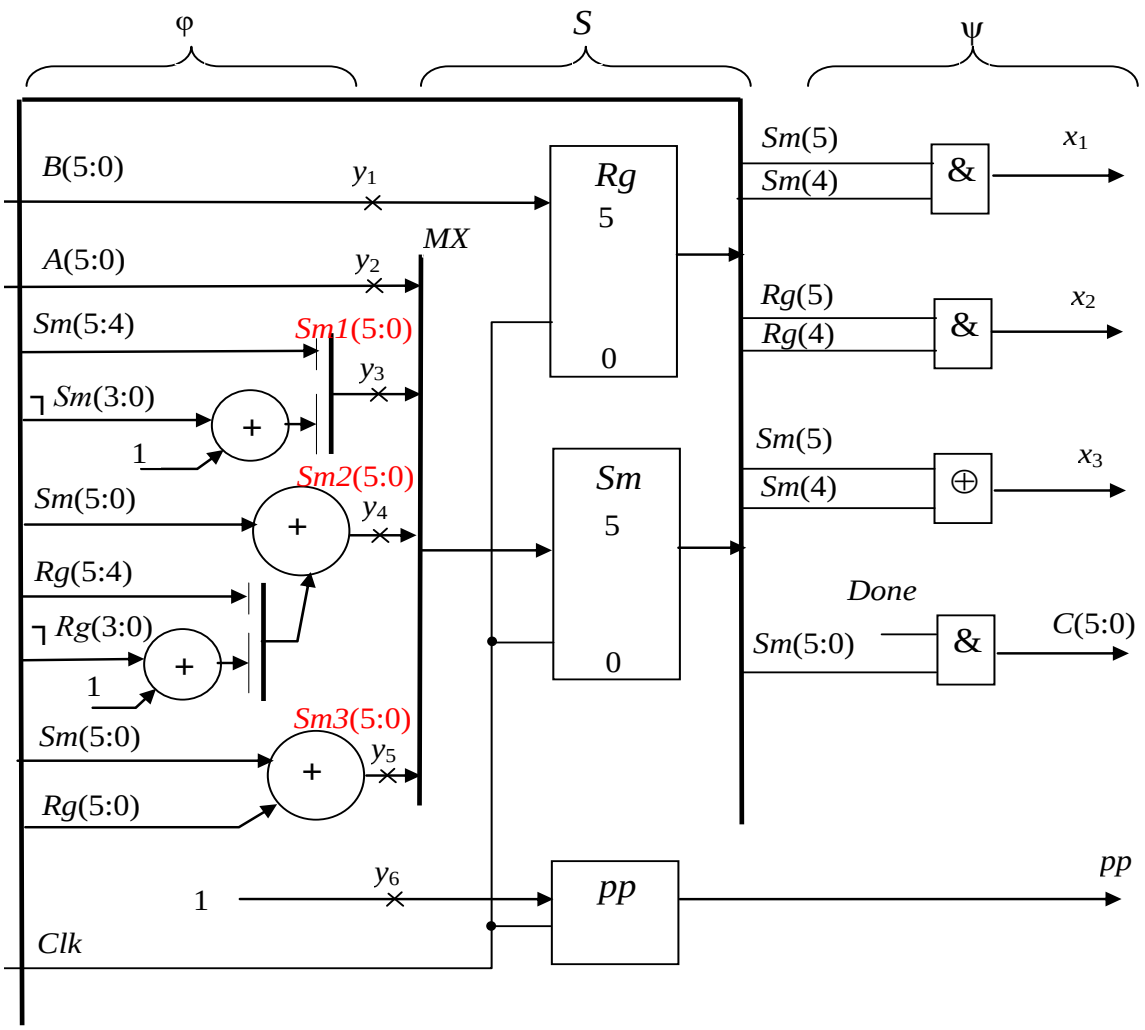


Рис. 6.15. Канонічна структура ОА

Відповідно до даної моделі операційний автомат поділяється на три частини: пам'ять S; комбінаційну схему φ, що реалізує функції φ, пов'язані з виконанням мікрооперацій Y; комбінаційну схему ψ, яка обчислює значення логічних умов X.

Функціонування операційних автоматів. Операційний автомат функціонує в часі в порядку, визначеному керуючим автоматом. Процес функціонування поділяється на інтервали часу, так звані *такти*.

Протягом такту виконуються такі дії:

1) керуючий автомат формує набір з 1, 2 або більше сигналів $y_a, ..., y_w$;

2) під впливом керуючих сигналів $y_a, ..., y_w$ в комбінаційних схемах φ_i автомата обчислюються значення двійкових виразів $\varphi_i(S), ..., \varphi_i(S)$;

3) відповідні регістри перемикаються в нові стани, які визначаються значеннями $\varphi_i(S), \ldots, \varphi_i(S)$;

4) комбінаційні схеми ψ обчислюють нові значення інформаційних сигналів X, відповідні новому стану пам'яті S, тобто новим значенням слів $s_1, \ldots, s_m$. Тривалість такту T визначається структурою схем і швидкодією логічних і запам'ятовуючих елементів, які використовуються у схемах керуючого і операційного автоматів.

Таким чином, такт – це проміжок часу між сусідніми моментами перемикання стану пам'яті. Протягом такту пам'ять автомата перемикається в новий стан, відмінний від попереднього.

Операційний автомат можна розглядати як самостійний об'єкт. У такому випадку термін «такт» умовно застосовується тільки для операційного автомату. *Такт операційного автомата* T – це проміжок часу, необхідний для виконання заданого набору мікрооперацій і обчислення значень логічних умов і рівний інтервалу часу від моменту надходження на вхід автомата керуючих сигналів до моменту вироблення значень інформаційних сигналів, відповідних стану пам'яті автомата.

Зазвичай розподіл дій у часі проводиться на основі синхронного принципу. При цьому тривалість такту T визначається максимальним значенням, необхідним для виконання будь-якої мікрооперації і обчислення значення будь-якої логічної умови. Хід часу відзначається за допомогою синхронізуючих сигналів, що виробляються генератором синхронізуючих імпульсів, які слідують з періодом, рівним тривалості такту T. Ці сигнали використовуються для синхронізації моментів перемикання тригерів.

На рис. 6.16 подано фрагмент канонічної структури ОА – реалізація мультиплексорів МХ.

6.2.1. Характеристики операційного автомата

Основні характеристики операційного автомата – продуктивність, швидкодія і витрати обладнання (вартість).

Продуктивність операційного автомата визначається кількістю мікрооперацій, виконуваних автоматом за один такт.

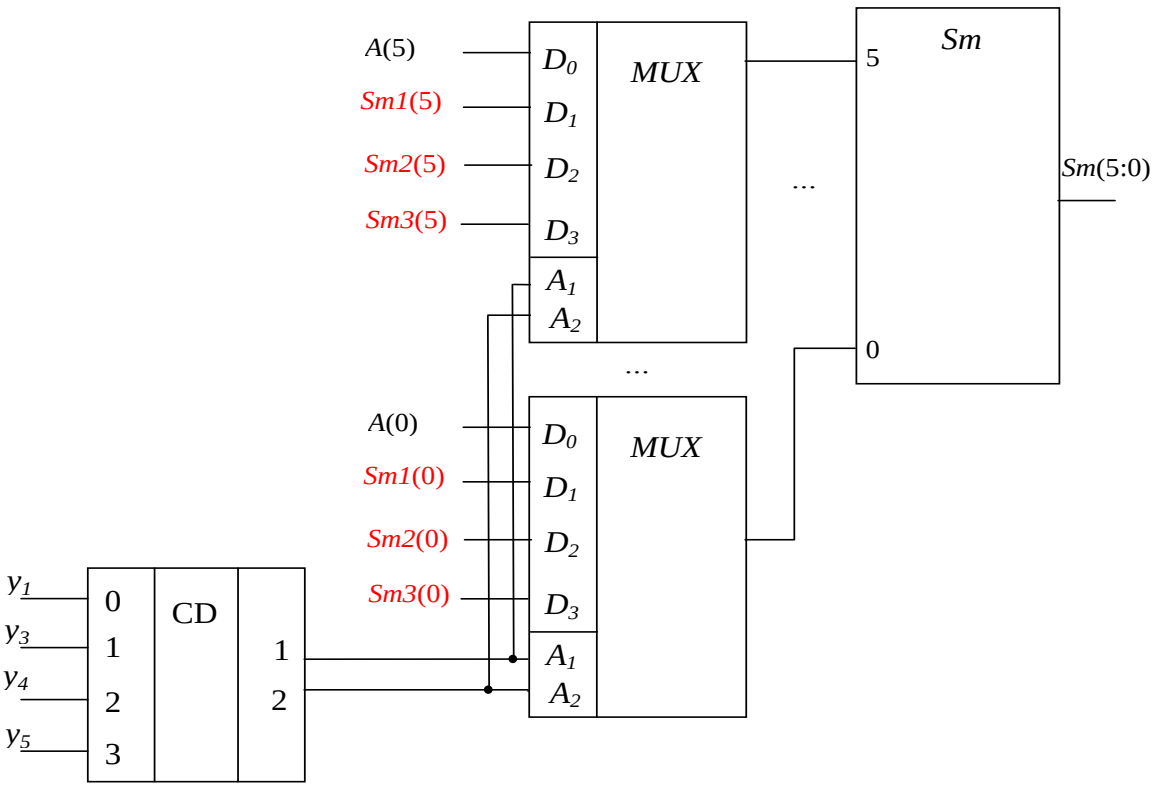


Рис. 6.16. Фрагмент канонічої структури ОА

В одному такті можуть виконуватися тільки сумісні мікрооперації, які фігурують в одному операторі мікропрограми, і їх кількість може змінюватися від такту до такту. Отже, кількість мікрооперацій, виконуваних за один такт, можна розглядати як дискретну випадкову величину. Фактор випадковості вноситься початковими даними D, залежно від значень яких процес виконання мікропрограми може розвиватися різними шляхами, що визначаються графом мікропрограми. З огляду на це продуктивність оцінюють або максимальним значенням, рівним максимальній кількості мікрооперацій, який може виконати автомат протягом такту, або середнім значенням. Середня оцінка продуктивності обчислюється таким чином. Мікропрограма складається з K операторів, кожен з яких містить m_k мікрооперацій (κ=1, ..., K), що виконуються спільно за такт часу. При одній реалізації мікропрограми кожен оператор виконується в середньому q_k раз. Виходячи з цих значень середня кількість мікрооперацій, виконуваних автоматом у такті, оцінюється величиною

$$V = \sum_{k=1}^{K} q_k m_k \,/\, \sum_{k=1}^{K} q_k \ .$$

Середня продуктивність V залежить від властивостей мікропрограми (кількості сумісних мікрооперацій в операторах і частоти використання операторів) і структури автомата, яка може накладати обмеження на сумісність мікрооперацій.

Швидкодія операційного автомата характеризується тривалістю такту T операційного автомата. Чим менше тривалість такту, тим вище швидкодія автомата. Швидкодія залежить в основному від внутрішньої структури комбінаційних схем φ і ψ і швидкісних характеристик логічних і запам'ятовуючих елементів, які використовуються в комбінаційних схемах і пам'яті автомата.

Витрати обладнання в операційному автоматі визначаються сумою витрат обладнання в пам'яті автомата S, комбінаційних схемах φ, що реалізують мікрооперації, і комбінаційних схемах ψ, що обчислюють значення логічних умов.

Канонічна структура має максимальну продуктивність порівняно з іншими варіантами структур, що реалізують один і той самий алгоритм. Це пояснюється тим, що канонічна структура не вносить обмежень на сумісність мікрооперацій: усі функціонально сумісні мікрооперації можуть виконуватися паралельно в одному такті. Тому витрати часу на виконання алгоритму з використанням канонічної структури мінімальні порівняно з іншими варіантами структур операційних автоматів. Менших витрат часу можна досягти, змінивши алгоритм виконання операцій у пристрої.

Швидкодія різних структур операційних автоматів, побудованих на одній і тій самій елементній базі, майже не відрізняється. Але все-таки канонічна структура має найвищу швидкодію (їй властива найменша тривалість такту) порівняно з іншими варіантами структур.

У більшості випадків канонічна структура не є мінімальною за кількістю використовуваного обладнання, що викликано такими причинами:

1) пам'ять автомата може бути надмірною відносно розглянутого алгоритму;

2) множина схем $Z = \{\varphi_1, \ldots, \varphi_n, \psi_1, \ldots, \psi_m\}$ може містити схеми, еквівалентні за реалізованими функціями;

3) множині схем Z може відповідати інша множина Z', еквівалентна за своїми функціями, але яка породжує менші витрати обладнання.

Зазначені причини усуваються так.

Для мінімізації пам'яті автомата розроблений ряд формальних методів, що дозволяють перетворити алгоритм таким чином, щоб мінімізувати кількість слів, які використовуються в алгоритмі для подання даних. Надмірність комбінаційних схем $Z = \{\varphi_1, \ldots, \varphi_n, \psi_1, \ldots \psi_m\}$ викликана таким. У схемі (рис. 6.15) для виконання мікрооперацій y_4, y_5 використовуються два суматори, еквівалентні з точки зору реалізованих ними функцій. Витрати обладнання зменшаться, якщо для виконання двох мікрооперацій використовувати один суматор. Таким чином, кілька комбінаційних схем $\varphi_1, \ldots, \varphi_n$, що реалізують однакові функції, можна замінити однією схемою, що дозволяє зменшити кількість обладнання в автоматі. Нарешті, шляхом глибоких перетворень алгоритму, що призводять до зміни набору мікрооперацій і логічних умов, можна створити набір, для реалізації якого будуть потрібні комбінаційні схеми Z', що призводять до менших витрат обладнання порівняно зі схемами Z.

Таким чином, канонічна структура, яка реалізує задану функціональну мікропрограму (алгоритм), має максимально можливу при даній мікропрограмі швидкодію. Канонічна структура синтезується прямо за функціональною мікропрограмою без використання будь-яких процедур мінімізації витрат обладнання. Тому в загальному випадку канонічна структура є надмірною за кількістю використовуваного обладнання.

6.2.2. Еквівалентні операції та узагальнений оператор

Розглянемо більш докладно, як можна зменшити апаратурні витрати. Для зменшення комбінаційної схеми можна виділяти еквівалентні операції і об'єднувати їх в узагальнений оператор. Як приклад розглянемо дві мікрооперації y1: $s_4 := s_1 + s_2$; y_2: $s_5 := s_1 + \lceil s_3 + 1$, де y_1 і y_2 – керуючі сигнали, що реалізують одну й

ту саму функцію додавання над різними словами. У канонічній структурі для реалізації цих мікрооперацій необхідні дві комбінаційні схеми – два суматори (рис. 6.17, а). Щоб зменшити витрати обладнання в операційному автоматі, можна використовувати тільки один суматор (рис. 6.17, б). У цьому випадку акумулятор підключається до регістрів s_2 s_3 керованими шинами, за допомогою яких вхід A_1 суматора з'єднується з регістром s_2 або s_3. Коли виконується мікрооперація y_1, на вхід A_1 надходить значення s_2, а при виконанні мікрооперації y_2 – значення s_3. Аналогічно константа 1 надходить на вхід A_2 суматора тільки при виконанні мікрооперації y_2.

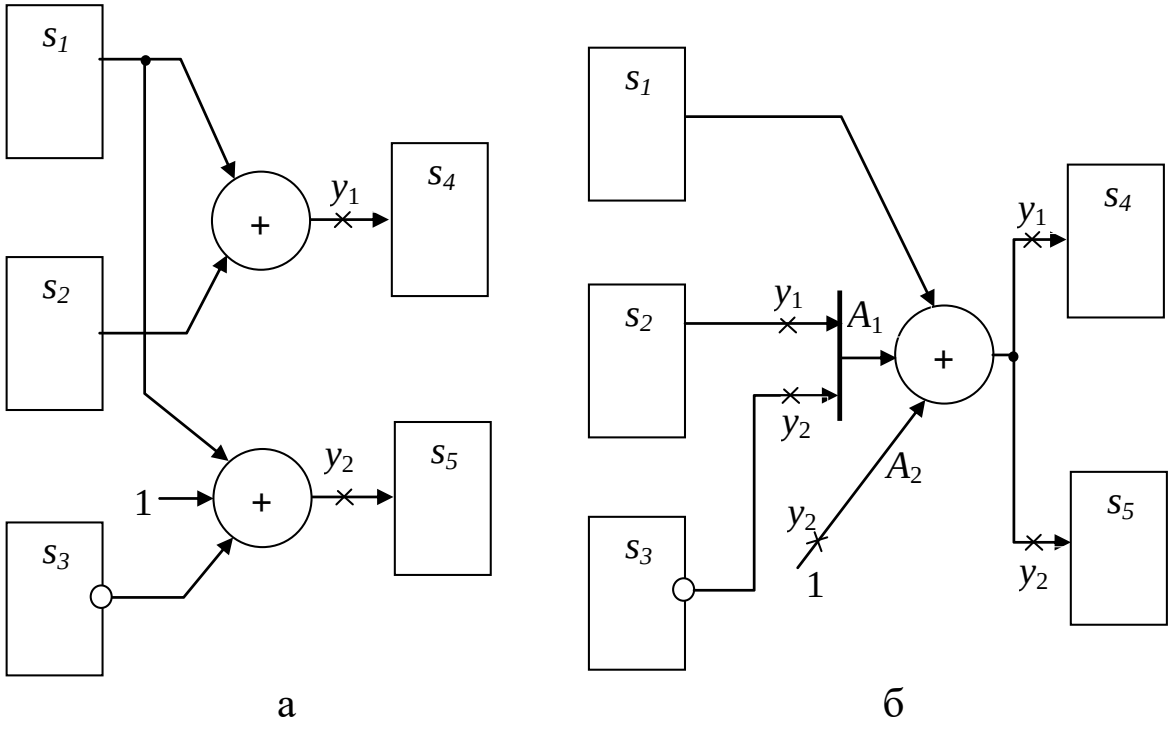


Рис. 6.17. Реалізація еквівалентних мікрооперацій

Таким чином, одна комбінаційна схема може використовуватися для виконання декількох мікрооперацій, еквівалентних у сенсі реалізованих функцій. За рахунок цього можуть бути зменшені витрати обладнання в операційному автоматі. Використання однієї комбінаційної схеми для виконання декількох мікрооперацій виключає сумісність цих мікрооперацій. Так, функціонально сумісні мікрооперації в структурі (рис. 6.17, а) можуть виконуватися в одному такті і

стають несумісними в структурі (рис. 6.17, б), тобто можуть виконуватися тільки в різних тактах. Якщо у функціональній мікропрограмі мікрооперації y_1 і y_2 використовувалися спільно в одній операторній вершині, то при використанні структури (рис. 6.17, б) час операції збільшується через структурні обмеження на сумісність цієї пари мікрооперацій. Таким чином, економія обладнання може спричинити збільшення часу виконання операцій.

Мікрооперації $S_{\alpha 1}:=\varphi_m(S_{\alpha 2},\ldots,S_{\alpha\beta})$ та $S_{\beta I}:=\varphi_m(S_{\beta 2},\ldots,S_{\beta q})$ вважаються еквівалентними, коли оператори містять одну і ту саму функцію φ_m, тобто функції в операторах мають однакові імена.

Еквівалентність встановлюється таким способом.

Двійкові вирази $C_{\alpha 1}*C_{\alpha 2}*\ldots*C_{\alpha p}$ і $C_{\beta 1}*C_{\beta 2}*\ldots*C_{\beta q}$, де C_α, C_β – аргументи, що подаються словами, їхніми інверсіями і константами, * – знаки двійкових операцій, називаються еквівалентними, якщо один з двійкових виразів може бути приведено до іншого шляхом:

1) заміни слова C_α словом C_β або інверсією TC_β;

2) заміни слова C_α константою (у тому числі і нулем) і навпаки;

3) заміни одних констант іншими, у тому числі і нульовими;

4) рівносильними перетвореннями виразів $C_{\alpha 1}*C_{\alpha 2}*\ldots*C_{\alpha p}$.

Еквівалентним мікрооперaціям y_1 і y_2 відповідає узагальнений оператор

$$S:= S_1 + A_1 + A_2,$$

де

$$A_1 = \begin{cases} S_2, & y_1 = 1, \\ \neg S_3, & y_2 = 1 \\ 0 \text{ в решті випадках} \end{cases} \qquad A_2 = \begin{cases} 1, & y_2 = 1, \\ 0 \text{ в решті випадках} \end{cases}.$$

6.2.3. Явище гонок в операційних автоматах

Як і в керуючих автоматах, в операційних автоматах можуть відбуватися гонки. Явище гонок полягає в такому. У момент надходження керуючого сигналу, який ініціює мікрооперацію, у

схемі починається процес передачі сигналів з виходів регістрів через комбінаційну схему до входів тригерів регістра, якому присвоюється нове значення, що обчислюється комбінаційною схемою. Під впливом вхідних сигналів тригери регістра перемикаються в нові стани.

Через відмінності часових характеристик ланцюгів, якими поширюються сигнали, може припинитися вироблення одних сигналів збудження під впливом інших, що мають менший час поширення, у результаті чого деякі запам'ятовуючі елементи не встигають перейти в необхідні стани. Таке явище називається гонками сигналів. Результат гонок – спотворення функцій, виконання яких покладено на схему.

Для запобігання гонок необхідно використовувати тригери з динамічним управлінням і коректно розрахувати період синхросигналу.

6.3. Операційний автомат типу I

Проєктований операційний автомат має забезпечувати задану продуктивність. Необхідна продуктивність може бути забезпечена тільки в тому випадку, якщо синтезована структура не вносить обмежень на сумісність мікрооперацій, тобто забезпечує можливість одночасного виконання всіх функціонально сумісних мікрооперацій.

Для мінімізації витрат апаратури необхідно узагальнювати комбінаційні схеми, що виконують мікрооперації з безлічі $Y = \{S_n := \varphi_{m(S)}\}$, $n = 1,\ldots, N$ і обчислюють інформаційні сигнали з множини $X = \{x_e = \psi_e (S_{\beta 1},\ldots,S_{\beta k})\}$.

Операційні автомати, структура яких забезпечує одночасне виконання всіх еквівалентних і функціонально сумісних мікрооперацій при використанні мінімально можливої кількості комбінаційних схем, належать до класу I-автоматів.

Структура I-автомата синтезується в такий спосіб:

1. Множина мікрооперацій $Y = \{y_1,\ldots,y_m\}$ розбивається на підмножини $Y_1,\ldots,Y_N$, відповідні внутрішнім регістрам $S_1,\ldots,S_N$.

2. На підмножинах Y_n, $n=1,\ldots, N$ виділяються класи еквівалентних мікрооперацій K_{nj}, $j = 1,\ldots, J_n$.

3. Для кожного класу K_{nj}, що містить не менше двох еквівалентних мікрооперацій, будується узагальнений оператор.

Якщо клас K_{nj} містить тільки одну мікрооперацію, то узагальненим оператором даного класу є сама мікрооперація.

4. Виходячи з опису слів, узагальнених операторів і логічних умов будується структура схеми І-автомата.

Структури ОА типу І наведені на рис. 6.18.

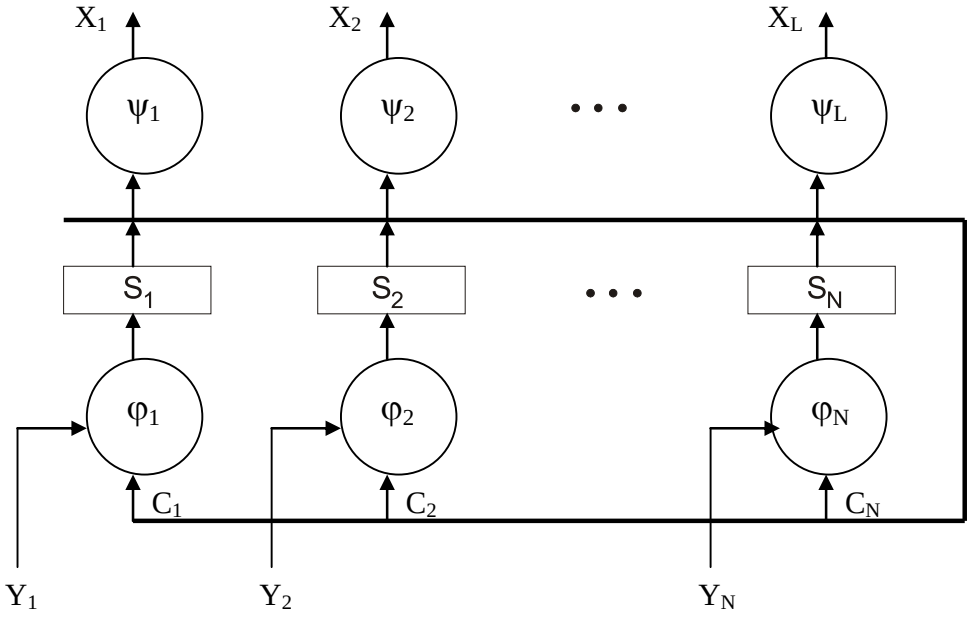


Рис. 6.18. Структура ОА типу І

Для І-автоматів характерно те, що кожен регістр обслуговується своєю комбінаційною схемою, засобами якої реалізуються мікрооперації, що обчислюють значення слова s_n.

6.4. Операційний автомат типу М

У структурі І-автомата можуть міститися комбінаційні схеми, які використовуються для обслуговування різних регістрів і виконують однакові еквівалентні операції.

Для заданого алгоритму функціонування апаратурні витрати можна мінімізувати, якщо комбінаційні схеми узагальнити відносно всіх регістрів S_1 ,…, S_n. Операційні автомати, синтезовані на основі принципу узагальнення комбінаційних схем, називаються М-автоматами. Структурна організація М-автомата показана на рис. 6.19.

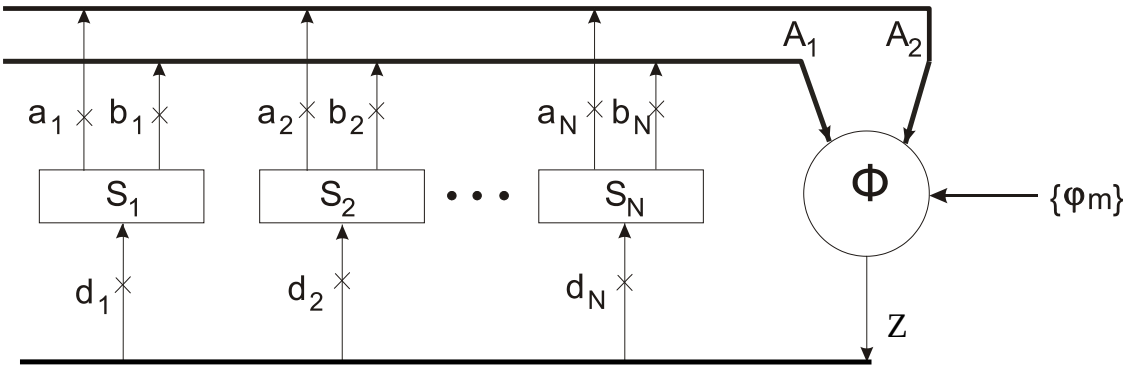


Рис. 6.19. Структурна організація М-автомата

У структурі М-автомата регістри S_1 ,…, S_N використовуються для зберігання слів. Узагальнена комбінаційна схема Ф використовується для обчислення значень $\varphi_m(S_{p1}$,…, $S_{pk})$, виконуваних мікрооперацією. Дані (операнди) надходять на входи схеми Ф через шини A_1 , A_2. Для вибору слова на шину A_1 використовуються сигнали a_1 ,…, a_n, для вибору слова на шину A_2 використовуються сигнали b_1 ,…, b_n. Сигнал α_I ініціює присвоєння $A_1:=S_i$, сигнал b_i ініціює присвоєння $A_2:= S_j$.

Схема Ф налаштовується на виконання мікрооперації $Z:= \varphi_m(A_1, A_2)$ за допомогою керуючого сигналу φ_m. Завантаження результату Z у будь-який регістр S_k ініціюється керуючим сигналом d_k .

Як зрозуміло з рис. 6.19, М-автомат виконує одну мікрооперацію за такт. Таким чином, продуктивність М-автомата мінімальна.

Швидкодія М-автомата незначно відрізняється від швидкодії І-автомата. Тривалість такту трохи збільшується за рахунок введення у схему мультиплексорів A_1, A_2.

Витрати обладнання М-автомата мінімальні, оскільки комбінаційна схема Ф використовується для виконання всіх еквівалентних мікрооперацій з множини Y.

М-автомат породжує специфічний набір керуючих сигналів $\{a_i\}$, $\{b_j\}$,$\{\varphi_m\}$,$\{d_k\}$. Кожен сигнал ініціює певну дію, яку можна розглядати як мікрооперацію.

6.5. Операційний автомат типу ІМ

Структура І-автомата базується на принципі закріплення комбінаційних схем за кожним з регістрів (принцип індивідуального закріплення). Структура М-автоматів базується на усуспільненні (узагальненні) комбінаційних схем до всіх регістрів. Тому І-автоматам притаманні максимальна продуктивність, але і максимальні витрати обладнання. У разі М-автомата продуктивність мінімальна при найменших витратах обладнання.

Операційний автомат зі структурою, що носить обмеження на сумісність мікрооперації і одночасно з цим забезпечує виконання за один такт більше однієї МО, називається ІМ-автоматами.

ІМ-автомати займають проміжне місце між М-автоматами і І-автоматами: продуктивність їх вище, ніж у М-автоматів, а витрати обладнання менше, ніж в І-автоматів.

Існує два типи ІМ-автомата: з паралельною комбінаційною частиною і послідовною комбінаційною частиною.

6.5.1. Операційний автомат типу ІМ з паралельною комбінаційною частиною

Регістри $S_1,\ldots,S_2$ ІМ-автомата, наведеного на рис. 6.20, зберігають слова даних. Комбінаційні схеми $Ф_1$ і $Ф_2$ обчислюють булеві вирази, які реалізують функції $\{f_c\}$ і $\{f_m\}$ відповідно. У цьому випадку схема $Ф_1$ виконує бінарну операцію $Z_1:= f_c (A_1,A_2)$ (додавання, кон'юнкцію, диз'юнкцію, складання за модулем) над змінними A_1 і A_2. Значення необхідних змінних вибираються на шини A_1 і A_2 з регістрів S_i, S_j за допомогою сигналів a_i, b_j.

Регистри $S_1,\ldots,S_2$ ІМ-автомата, наведеного на рис. 6.20, зберігають слова даних. Комбинаційні схеми $Ф_1$ і $Ф_2$ розраховують булеві вирази, що реалізують функції $\{f_c\}$ і $\{f_m\}$ відповідно. У цьому випадку схема $Ф_1$ виконує бінарну операцію $Z_1:= f_c (A_1,A_2)$ (додавання, кон’юнкція, диз’юнкція, додавання за модулем) над змінними A_1 і A_2 . Значення необхідних змінних выбираются на шини A_1 і A_2 із регистрів S_i , S_j за допомогою сигналів a_i , b_j.

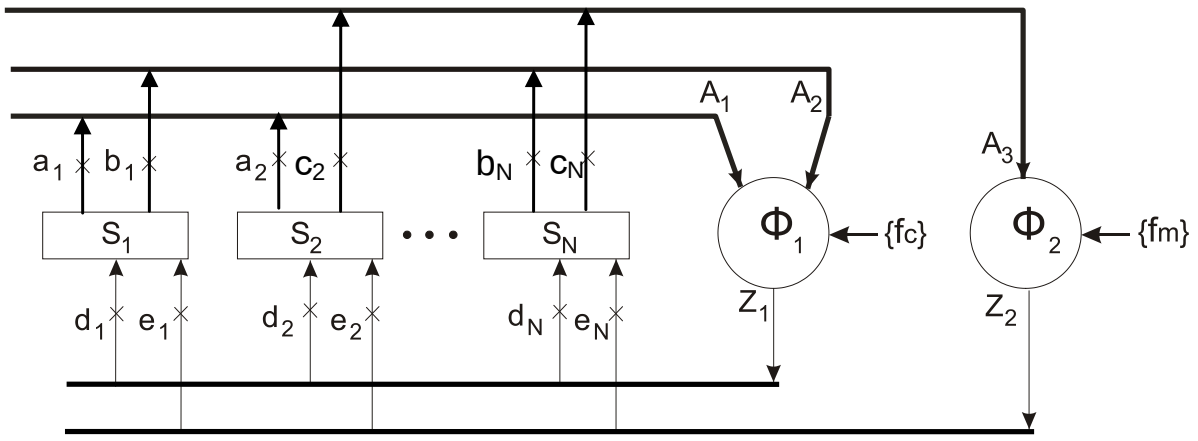


Рис. 6.20. Структура ІМ-автомата з паралельною комбінаційною частиною

Схема Ф2 виконує унарну мікрооперацію $Z_2:= f_m (A_3)$ (передачу, інвертування, зсув, формування констант і т. п. над допоміжною змінною A_3) керуючим сигналом c_k. Результати мікрооперації Z_1, Z_2 завантажуються в регістри S_p, S_r, що визначаються керуючими сигналами d_p, e_r.

Автомат виконує дві мікрооперації $S_p:= f_c (S_i, S_j)$ і $S_r:= f_m (S_k)$ за один такт за допомогою керуючих сигналів (a_i, b_j, c_k, f_c, f_m, d_p, e_r).

Ці сигнали ініціюють такі мікрооперації:

a_i: $A_2:= S_i$; b_j: $A_1:= S_j$; c_k: $A_3:= S_k$; d_p: $S_p:= Z_1$; e_r: $S_r:=Z_2$;

f_c: $Z_1:= f_c (A_1,A_2)$; f_m: $Z_2:= f_m (A_3)$.

Максимальна продуктивність ІМ-автоматів з В паралельними комбінаційними схемами $Ф_1,…, Ф_B$ $1<B<N$ дорівнює В мікрооперацій за такт і збільшується зі збільшенням кількості комбінаційних схем. Таким чином, кількість комбінаційних схем визначається вимогами до швидкодії автомата.

ІМ-автомат може розглядатися як композиція М-автоматів, що мають загальну пам'ять $S_1,…, S_n$. Виходячи з цього синтез ІМ-автомата з паралельними комбінаційними здебільшого зводиться до розбиття множини мікрооперацій $Y=\{y_1,…,y_m\}$ на В підмножин і синтезу В автоматів, які реалізують зазначені підмножини.

6.5.2. Операційний автомат типу ІМ з послідовною комбінаційною частиною

ІМ-автомат з послідовною комбінаційною частиною, структуру якої наведено на рис. 6.21, містить комбінаційні схеми

Φ_1, Φ_2, Φ_3, реалізує мікрооперації з множин $\{f_k\}$, $\{g_e\}$, $\{h_m\}$ відповідно. Операції виконуються з операндами на шинах A_1 і A_2. Зокрема сигнали a_i, b_j ініціюють мікрооперації $A_2:= a_i$; $A_1:= a_j$, комбінаційні схеми Φ_1, Φ_2, Φ_3 виконують мікрооперації $A_3= f_h(A_3)$; A_4: $y_e(A_1,A_2)$; $Z = h_m(A_4)$ під дією сигналів f_h, g_e, h_m відповідно. Результат Z записується в пам'ять $S_n:= Z$ сигналом d_n.

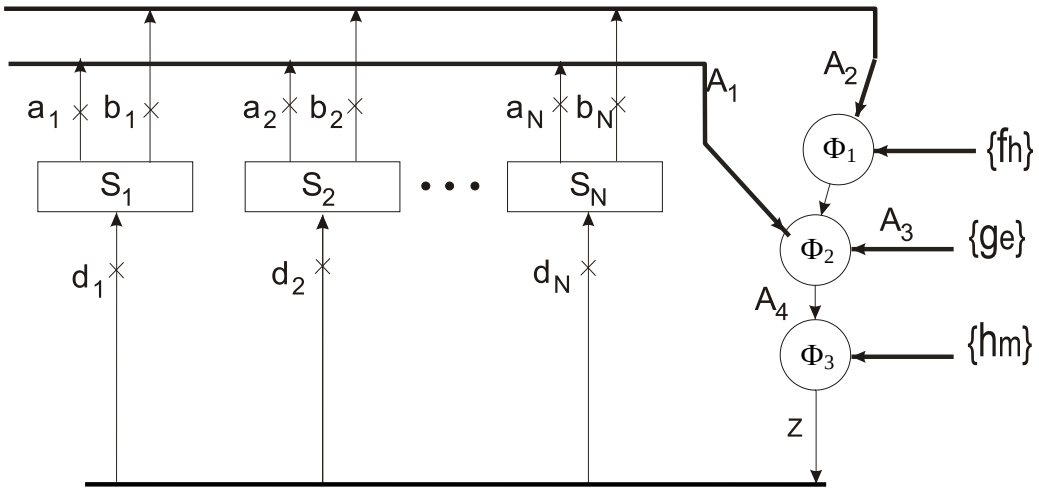


Рис. 6.21. Структура ІМ-автомата з послідовною комбінаційною частиною

Таким чином, набір керуючих сигналів (a_i, b_j, f_h, g_e, h_m, d_n) ініціює виконання мікрооперацій f_h, g_e, h_m над словами S_i, S_j і записи результату в регістр S_n, тобто реалізується перетворення $S_n:= h_m (g_e (S_j, f_h (S_j)))$.

Цей оператор еквівалентний трьом послідовно виконуваним мікрооперaціям. Це означає, що продуктивність даного ІМ-автомата в три рази вище продуктивності М-автомата. Для забезпечення можливості виконання мікрооперацій f_h, g_e, h_m у будь-якому поєднанні кожна зі схем Φ_1, Φ_2, Φ_3 має виконувати такі мікрооперації:

$A_3:= A_2$; $A_4:= A_1$; $A_4:= A_3$; $Z:= A_4$.

Таким чином, комбінаційна частина, крім перетворення $S_n:=h_m (g_e (S_j, f_h (S_j)))$, має виконувати такі мікрооперації:

$Z:= A_1$; $Z:= A_3$; $Z:= f_h (H_2)$; $Z:=g_e(A_1,A_2)$; $Z:= h_m (H_1)$; $Z:=g_e (H_1, f_h (A_2))$ і т. д.

При використанні ІМ-автоматів як ОА процесорів схеми Φ_1, Φ_2, Φ_3 зазвичай виконують такі функції. Схема Φ_1 реалізує

інверсію, генерацію констант, формування полів. Вона називається формувачем кодів. До таких мікрооперацій формувачі кодів належать

f_0: $A_3:=A_2$; f_1: $A_3:=1$; f_2: $A_3:=15_{10}$; f_3: $A_3:=1010$;

f_4: $A_3:=0*A_2(1:n)$; f_5:$A_3:=1*A_2(1:n)$;

f_6: $A_3:=1*\rceil A_2(1:n)$; f_7: $A_3:=A_2(0:7)$, 00…0.

Мікрооперація f_0 забезпечує передачу слова A_2 без перетворення мікрооперацій f_1, f_2, f_3 значення констант. Мікрооперація f_4 служить для передачі модуля операнда, мікрооперація f_5 для передачі числа з протилежним знаком. Мікрооперація f_6 формує зворотний код слова A_2, а мікрооперація f_7 виділяє перший байт операнда. Схема F_2 виконує бінарні мікрооперації , +, $\vee$, $\wedge$, $\oplus$. Тому Φ_2 часто називається суматором.

g_0: $A_4:=A_1$; g_1: $A_4:=A_3$; g_2: $A_4:=A_1+A_3$; g_3: $A_4 \oplus A_3$; g_4: $A_4:=A_1 \wedge A_3$; g_5: $A_1 \vee A_3$.

Схема ϕ використовується для виконання зсуву і називається С-двигуном.

С-двигун реалізує такі мікрооперації:

h_0: $Z:=A_4$; h_1: $Z:=k1(0*A_4)$; h_2: $Z:=L1(A_4*0)$;

h_3: $Z:=R1(A_4(n)*A_4)$; h_4: $Z:=L1(A_4*A_4(n))$; h_5: $Z:=R2(00*A_4)$.

Різним значенням керуючих сигналів (g_e, f_k, h_m) відповідають різні перетворення:

(f_1,g_2, h_0) – $Z:=A_1+1$; (f_1,g_1, h_0) – $Z:=15_{10}$;

(f_6,g_2, h_0) – $Z:=A_1+1.\rceil A_2(1:n)$; (f_4,g_2, h_1) – $Z:=R1(0(A_1+A_2(1:n)))$.

6.6. Операційний автомат типу S

У деяких операційних пристроях (ОП) МО виконуються над великою кількістю внутрішніх слів, від десятків до сотень (наприклад спеціалізовані процесори введення/виведення).

Для зменшення вартості таких пристроїв стандартну регістрову пам'ять доцільно замінити оперативним запам'ятовуючим пристроєм (ОЗП). ОА, пам'ять якого складається з ОЗП, називається S-автоматом.

Структура S-автомата показана на рис. 6.22.

Задана мікрооперація φ_m реалізується комбінаційною схемою F. Схема обчислює значення слова $Z:=\varphi_m(P_1,P_2)$.

Значення Z завантажується в ОЗП за адресою $A=A_3$. Такт S-автомата поділяється на такі дії:

1) читання першого операнда з пам'яті за адресою A_1: $P_1:=ЗП[A_1]$;

2) читання другого операнда з оперативного запамятовуючого пристрою за адресою A_2: $P_2:=ЗП[A_2]$;

3) виконання мікрооперації $Z:=\varphi_m(P_1,P_2)$;

4) запис результату до пам'яті оперативного запамятовуючого пристрою $[A_3]:=Z$.

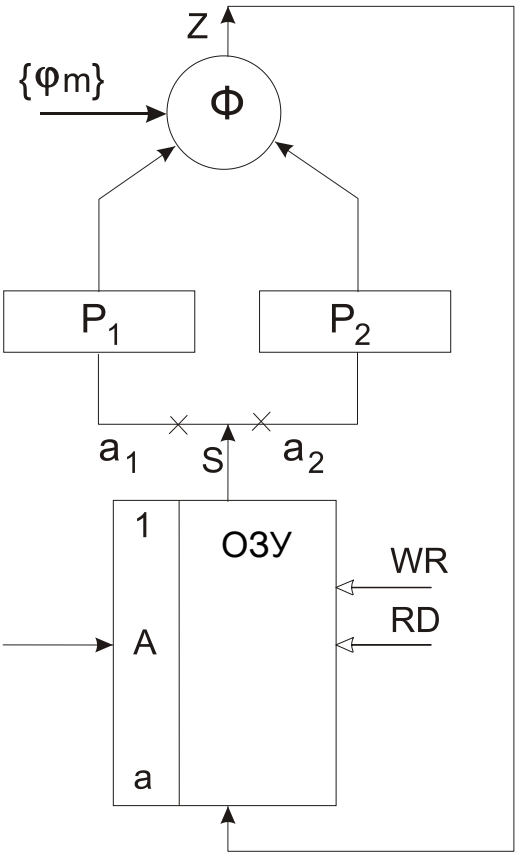


Рис. 6.22. Структура S-автомата

Зазвичай комбінаційна частина S-автомата будується за схемою M-автомата. У цьому випадку S-автомат виконує одну мікрооперацію за такт запам’ятовуючого пристрою $[A_3]:=Z$, а $Z:=\varphi_m(P_1,P_2)$. Швидкодія S-автомата зазвичай менше швидкодії ОА з регістрової пам'яттю. Причиною цього є можливість одночасної вибірки багатьох операндів. Швидкодія S-автомата визначається тривалістю такту. Залежно від способу побудови

комбінаційної частини S-автомат синтезується як I-, M-, IM-автомат.

Контрольні запитання

1. Поясніть поняття модульності, магістральності і мікропрограмованості МПС під час проєктування.
2. Перелічіть завдання, вирішувані розробниками під час проєктування МПС.
3. Перелічіть основні етапи проєктування МПС.
4. Назвіть концептуальні рівні опису МПС під час проєктування і розроблення.
5. Перелічіть основні методи контролю правильності проєктування МПС.
6. Які властивості має проєктована МПС для виконання етапу її налагодження?
7. Перелічіть види несправності під час проєктування МПС.
8. Назвіть причини фізичної і суб'єктивної несправностей МПС.
9. Поясніть поняття діагностики несправності, налагодження.

7. ОСНОВИ МОВ ОПИСУ АПАРАТУРИ

7.1. Мови опису апаратури

Суттєвим моментом при сучасному проєктуванні програмованою логікою є ієрархічне подання проєкту у взаємозв'язку з двома найбільш популярними мовами опису апаратури. Типовою класифікацією є поділ рівнів відповідно до технологій проєктування, наведений на рис. 7.1.

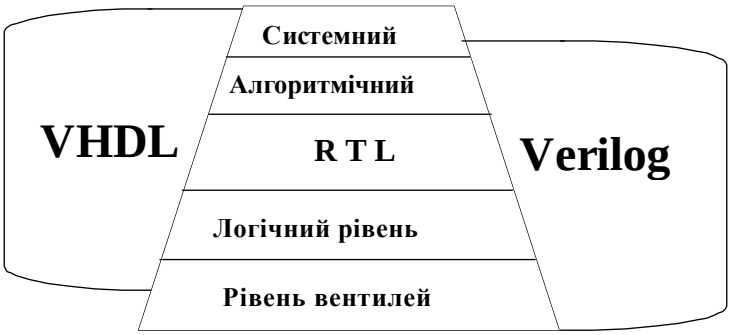


Рис. 7.1. Етапи проєктування і мови опису апаратури

1. Системне (алгоритмічне) проєктування. Характеризується високим рівнем абстракції опису моделей мовами VHDL, Verilog, System C. Швидкість виконання проєктів – більше 1 млн вентилів на місяць на одного проєктувальника, складність проєктів – 105-106 вентилів. Обсяг виконуваних на ринку проєктів – 60 %. Конструктор спільного апаратно-програмного проєктування Hardware-Software Cooperation Design і стратегія повторного застосування готових проєктів – IP-core reuse.

2. Схемотехнічне проєктування. Опис моделей на RTL (Register transfer level) рівні (рівні регістрових передач) за допомогою мов VHDL (більш універсальна і академічна, популярна в Європі, у середовищі університетських вчених), Verilog (більш проста мова, промислово-орієнтованай, популярна в Японії, США, у списку користувачів фірми SUN, Apple, Motorola), System C (у стадії розроблення компіляторів і тестування), EDIF (стандарт внутрішнього опису цифрових проєктів і подання схем після синтезу). Швидкість виконання проєктів – близько ста тисяч вентилів на місяць на одного

проєктувальника, складність проєктів – 104 вентилі. Обсяг виконуваних на ринку проєктів 40 %.

3. Логічне проєктування цифрових систем. Опис моделей на вентильному рівні у вигляді булевих рівнянь за допомогою мов VHDL, Verilog, System C, EDIF. Використовується самостійно для створення нескладних проєктів (близько 104 вентилів), що вимагають надвисокої роздільної швидкодії і мінімальних витрат апаратури. Зараз цей рівень розглядається як автоматична процедура синтезу цифрової системи.

У зв'язку з постійним удосконаленням технологій виробництва інтегральних схем з'являється можливість розміщувати більшу кількість елементів в одній мікросхемі. Цифрові системи все більше ускладнюються. Детальне проєктування таких пристроїв на вентильному (логічному) рівні стає практично неможливим. Через це зростає значущість мов опису апаратури. Вони дозволяють розробляти і верифікувати цифрові пристрої на верхніх рівнях до перетворення в логічний. Цей підхід подібний до написання програмних засобів мовами програмування (Сі) і використання компілятора для перекладу такої програми в машинний код. Дві найбільш популярні мови опису апаратури – VHDL і Verilog.

VHD – це абревіатура англійського виразу VHSIC Hardware Description Language. У свою чергу VHSIC походить від назви програми Very High Speed Integrated Circuit (високошвидкісні інтегральні схеми). Ця програма, фінансована Міністерством Оборони США, ставила собі за мету розвиток нового покоління високошвидкісних інтегральних схем. Перша версія мови була представлена в 1985 році. Згодом вона була передана IEEE для стандартизації. У 1987 році мова була затверджена як стандарт IEEE. Через п'ять років вона була розглянута повторно, у результаті чого нова версія 1076-93 містить ряд додаткових можливостей.

З того часу VHDL користується постійно зростаючою популярністю серед фахівців з автоматизованого проєктування електронних систем. Перші VHDL-додатки з'явилися на початку 1990-х років. Для синтезу був розроблений пакет IEEE 1164. На практиці кожен великий виробник систем автоматизованого проєктування підтримує VHDL, що дозволяє описувати системи

на різних рівнях. Він реалізує методологію спадного проєктування, у якій система спочатку описується на високому рівні і тестується за допомогою засобів моделювання, після чого поетапно приводиться до структурного опису, тісно пов'язаного з фактичною апаратною реалізацією. VHDL був розроблений незалежною для технології реалізації. Якщо якийсь проєкт, описаний у VHDL, реалізований для конкретної електронної технології, то цей же VHDL-опис може бути використано як вихідний для реалізації пристрою в новій технології.

Verilog, або Verilog HDL – мова опису апаратури, розроблена в 1985 році Філіпом Мурба (Philip Moorby), що потребувала простого, наочного і ефективного способу для опису цифрових схем, моделювання та аналізу їх функціонування. Мова стає власністю спочатку Gateway Design Automatiion, потім Cadence Design Systems. У 1990 році Cadence відкриває мову, що призводить її до стандартизації IEEE в 1995 році.

Що ж краще – VHDL або Verilog? Verilog легше в розумінні і використанні, застосовувалася і застосовується в промисловості, де потрібні моделювання та синтез. Але має недолік у конструкціях для опису рівнів системи. Verilog отримала визнання в проєктуванні ASIC-схем, особливо для проєктів низького рівня (регістрових передач і нижче). Вона найбільш популярна в Північній Америці і Азії, особливо Японії. VHDL - більш складна мова, її важче вивчати і використовувати. Тим не менш, вона має більшу гнучкість, що є її перевагою і недоліком через велику кількість допустимих стилів коду. Оскільки VHDL краще підходить для роботи з дуже складними проєктами, у наш час її популярність зростає. Особливо це стосується Європи, США і Канади. Мова не має успіху в Японії. Однак у світі все більше і більше обирають мову VHDL.

Інструментальне середовище ALDEC ACTIVE-HDL V6.1 (6.3) – повністю інтегроване середовище для розроблення і моделювання ПЛІС мовами VHDL, Verilog або змішано VHDL/Verilog і EDIF заснованих проєктів. Програмна оболонка Active-HDL є передовим засобом у сфері середовищ для моделювання і верифікації HDL-проєктів з великою кількістю вентилів. Оскільки в промисловості розмір програмованих мікросхем неухильно зростає, то потреба у високопродуктивних

інструментальних засобах перевірки проєктів з великою кількістю вентилів набуває все більшої і більшої актуальності. Active-HDL Standard Edition (Стандартна версія) була розроблена для інженерів, які розробляють FPGA/CPLD (ПЛІС/ПЛМ) проєкти і яким необхідно потужний моделюючий пристрій для опису цих проєктів на рівні регістрових передач, більш того вона (це засіб) має бути реалізована в операційному середовищі PC Windows. Active-HDL підтримує всі інструментальні засоби і бібліотеки незалежно від фірм-виробників інтегральних схем.

Active-HDL включає три різні програми для введення проєктів, VHDL'93 і Verilog компілятори, єдине ядро моделювання, програмні модулі налагодження, графічне і текстове виведення результатів моделювання, допоміжні інструменти для зручного управління файлами ресурсів, проєктами і бібліотеками. В Active-HDL включені також кілька потужних майстрів для полегшення створення нових вихідних файлів, проєктів і випробувальних стендів TestBench. Більшість операцій, які викликаються через графічний користувальницький інтерфейс, можна виконувати за допомогою команд макромови Active-HDL. Командні файли дозволяють автоматизувати процес проєктування.

Інструментальне середовище Xilinx Progect Navigator V5.1 (7.0) пакет Xilinx ISE – повністю інтегроване середовище для здійснення синтезу, імплементації на ПЛІС, часового моделювання. Підтримує VITAL-стандарт (Стандарт – IEEE P1076.4: Timing Methodology (VITAL)) представлений пакетом, що містить типи для докладного опису pin-to-pin і distributed затримок, часові константи, включаючи максимальне, мінімальне та номінальні значення. Пакет також включає процедури для виконання часового аналізу.

7.2. Ознайомлення з основами мови опису апаратури Verilog

7.2.1. Система та її подання Verilog. Опис модуля Verilog

Verilog – легше в розумінні і використанні. Але має недоліків у конструкціях для опису рівнів системи. Verilog отримала

визнання в проєктуванні ASIC схем, особливо для проєктів низького рівня. Найбільш популярна ця мова в Північній Америці і Азії, особливо в Японії. Непопулярна в Європі.

VHDL – більш складна мова, її важче вивчати і використовувати. Тим не менш вона має більшу гнучкість, що є його перевагою і недоліком через велику кількість допустимих стилів коду. VHDL краще підходить для роботи з дуже складними проєктами. Популярна в Європі, США і Канаді. Мова не має успіху в Японії. Однак у світі все більше і більше використовують мову VHDL. Оксфордський тлумачний словник містить таке визначення системи: «система – це група речей або частин, які працюють разом у постійному зв'язку (залежності)». Система в Verilog – сукупність модулів, пов'язаних один з одним певним чином. Структура Verilog-модуля наведена на рис. 7.2, а опис модуля – на рис. 7.3.

Модуль складається з інтерфейсу (описує входи і виходи модуля), виділеного блакитним кольором, і тіла (описує функціонування модуля), виділеного рожевим кольором.

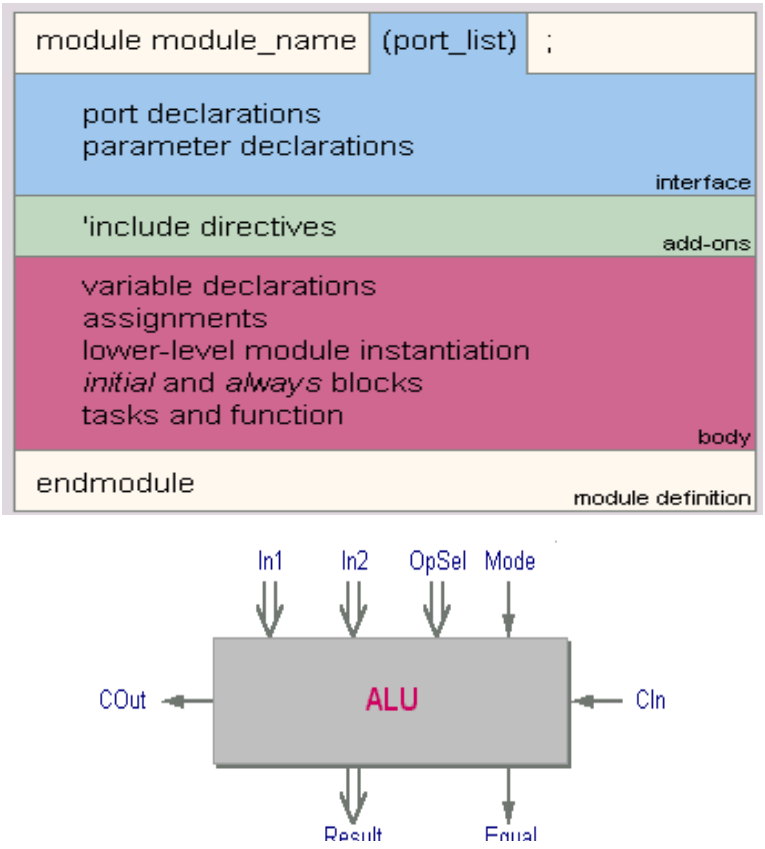


Рис. 7.2. Структура Verilog-модуля

```
module ALU (Result, COut, Equal, In1, In2,
            OpSel, CIn, Mode);

output [3:0] Result;   // operation result
output       COut;     // carry out
output       Equal;    // when 1, In1 = In2
input  [3:0] In1;      // first operand
input  [3:0] In2;      // second operand
input  [3:0] OpSel;    // operation select
input        CIn;      // carry in
input        Mode;     // mode arithm/logic;
                       // arithm when 0
 . . .

endmodule
```

Рис. 7.3 Опис інтерфейсу модуля

Зарезервоване слово module використовується для оголошення модуля.

ALU – це ім'я модуля, у дужках далі перераховані імена входів і виходів модуля. Зарезервоване слово output використовується для оголошення вихідного порта. Зарезервоване слово input використовується для оголошення вхідного порта. Зарезервоване слово endmodule означає кінець опису модуля. Типи портів явно не вказано. За замовчуванням порт має тип wire. Ідентифікатор імен модуля:

1) може складатися з букв, цифр, символу підкреслення (_);
2) має починатися з букви або символу підкреслення;
3) не може містити пробіли;
4) чутливий до регістру символів;
5) резервовані ключові слова не можуть бути використані.

Наприклад:	Коментарії
`reg enable;`	// on-line comment – однорядковий коментар
`wire _ready;`	
`integer group_a;`	/* block comment – блоковий коментар */
`reg and5;`	
`tri clk$1;`	

Існує три шляхи опису тіла модуля:

1) структурний стиль – з використанням примітивів і модулів низького рівня, наприклад вентилів;

2) data-flow стиль – з використанням опису вихідних сигналів як перетворення вхідних, наприклад з використанням булевих рівнянь або виразів алгебри;

3) поведінковий стиль – з використанням алгоритмічного вираження поведінки схеми.

Синтаксис імен модулів Verilog. Ідентифікатор імен модуля:

1) може складатися з букв, цифр, символу підкреслення (_);

2) має починатися з букви або символу підкреслення;

3) не може містити пробіли;

4) чутливий до регістру символів;

5) резервовані ключові слова не можуть бути використані, наприклад:

```
reg enable;
wire _ready;
integer group_a;
reg and5;
tri clk$1;
```

Структурний стиль опису модулів у Verilog полягає у використанні примітивів і модулів низького рівня, наприклад вентилів. Приклад структурного стилю опису модуля наведено на рис. 7.4. Стиль data-flow опису модулів у Verilog полягає в описі вихідних сигналів як перетворенні вхідних, наприклад з використанням булевих рівнянь. Приклад data-flow стилю опису модуля data-flow наведено на рис. 7.5. Поведінковий стиль опису на Verilog полягає в алгоритмічному вираженні поведінки схеми. Приклад поведінкового стилю опису модуля наведено на рис. 7.6.

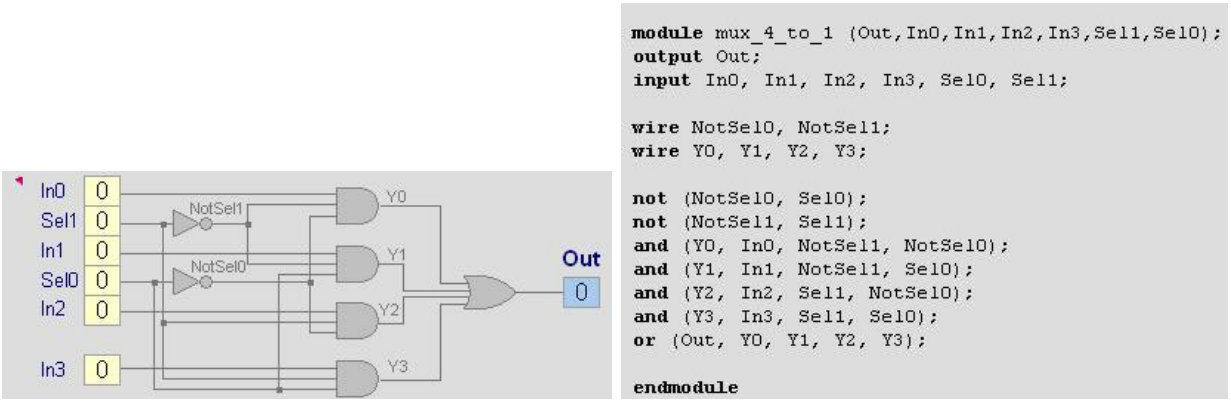
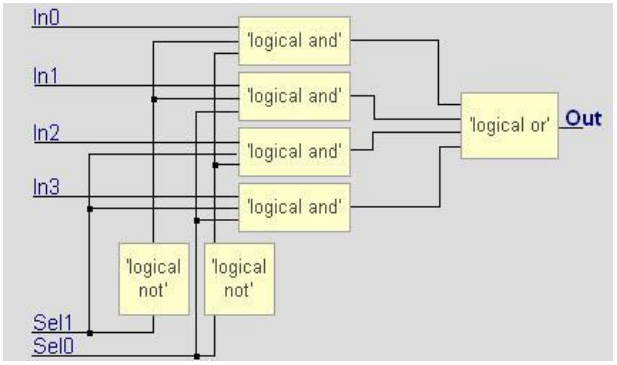


Рис. 7.4. Приклад структурного стилю опису модуля



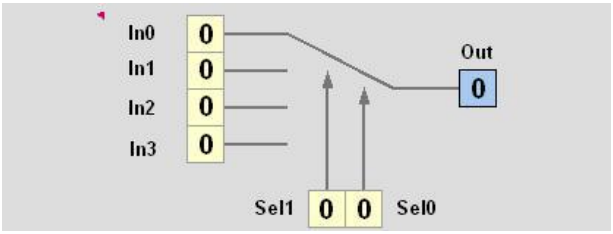

```
module mux_4_to_1 (Out, In0, In1, In2, In3, Sel1, Sel0);

output Out;
input In0, In1, In2, In3, Sel0, Sel1;

assign Out = (~Sel1 & ~Sel0 & In0) | (~Sel1 & Sel0 & In1)
           | (Sel1 & ~Sel0 & In2)| (Sel1 & Sel0 & In3);

endmodule
```

Рис. 7.5. Приклад data-flow стилю опису модуля




```
module mux_4_to_1 (Out,In0,In1,In2,In3,Sel1,Sel0);
output Out;
input In0, In1, In2, In3, Sel0, Sel1;
reg Out;

always @(Sel1 or Sel0 or In0 or In1 or In2 or In3)
  begin
    case ({Sel1, Sel0})
    2'b00 : Out = In0;
    2'b01 : Out = In1;
    2'b10 : Out = In2;
    2'b11 : Out = In3;
    default : Out = 1'bx;
    endcase
  end

endmodule
```

Рис. 7.6. Приклад поведінкового стилю опису модуля

7.2.2. Значення сигналів у Verilog, класи сигналів і вектори

Значення сигналів у Verilog на логічному рівні: 0 – логічний 0, умова false; 1 – логічна 1, умова true; x – невідоме значення; z – високий імпеданс.

Для значень сигналів допускається використання x і X, z і Z рівнозначно.

– – Приклад: `wire a; wire b, c; wire d = 1'b0;`

Клас **NET**

Wire/tri – лінія (лінія зв'язку – провідник, який проводить сигнал);

Wand/triand і wor/trior – лінії з І та АБО елементом;

supply0, supply1, tri0, tri1, trireg – управління описом елементів низького рівня.

Синтаксис ліній: Net declaration:

```
wire range delays list_of_identifiers [=
expression];
wand range delays list_of_identifiers [=
expression];
wor range delays list_of_identifiers [=
expression];
tri range delays list_of_identifiers [=
expression];
triand range delays list_of_identifiers [=
expression];
trior range delays list_of_identifiers [=
expression];
tri0 range delays list_of_identifiers [=
expression];
tri1 range delays list_of_identifiers [=
expression];
supply0 range delays list_of_identifiers [=
expression];
supply1 range delays list_of_identifiers [=
expression];
trireg strength range delays
list_of_identifiers [= expression];
```

Клас регістрових даних **REGISTER**: `reg DataOut;`

Відмінність цих класів ілюструє рис. 7.7:

а) не приєднаний до джерела сигналу провідник класу NET має значення z, а не приєднаний провідник класу REGISTER має значення x;

б) приєднані до джерела сигналу провідники класів NET і REGISTER мають значення джерела сигналу;

в) якщо після б) провідники від'єднати від джерела сигналу, не приєднаний провідник класу NET матиме значення z, а не приєднаний провідник класу REGISTER буде пам'ятати попереднє значення (як елемент пам'яті).

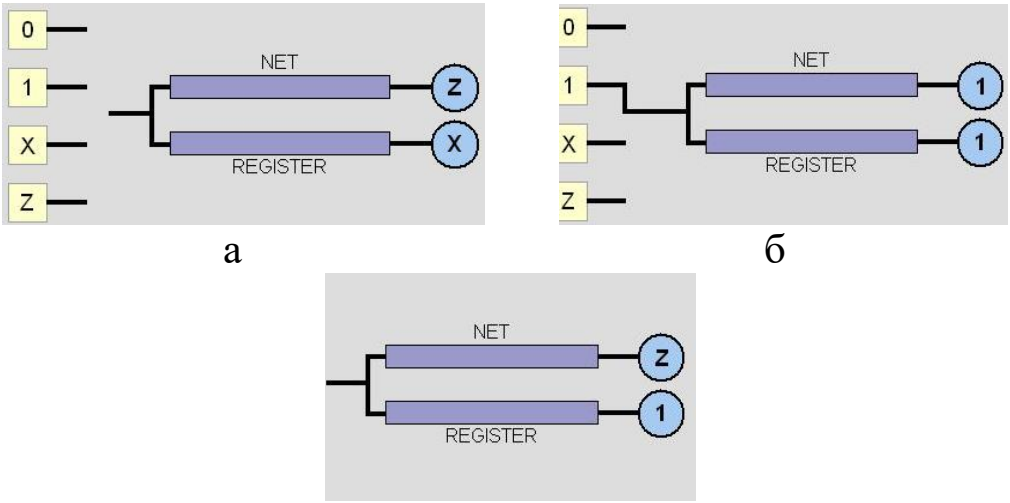


Рис. 7.7. Приклад поведінкового стилю опису модуля

Сигнали поділяються:

- на внутрішні (недоступні за модулем, оголошуються всередині модуля);
- зовнішні (доступні всередині модуля і поза модулем, оголошуються як порти).

Синтаксис ідентифікаторів (імен) сигналів:

– не може починатися з $ і символу підкреслення (_);

– не може містити пробіл;

– не може бути зарезервованим словом.

Сигнали бувають:

– скалярними (єдина лінія);

– векторними (множинна лінія – шина).

Опис векторів наведено на рис. 7.8, оголошення портів – на рис. 7.9, регістрові порти – на рис. 7.10.

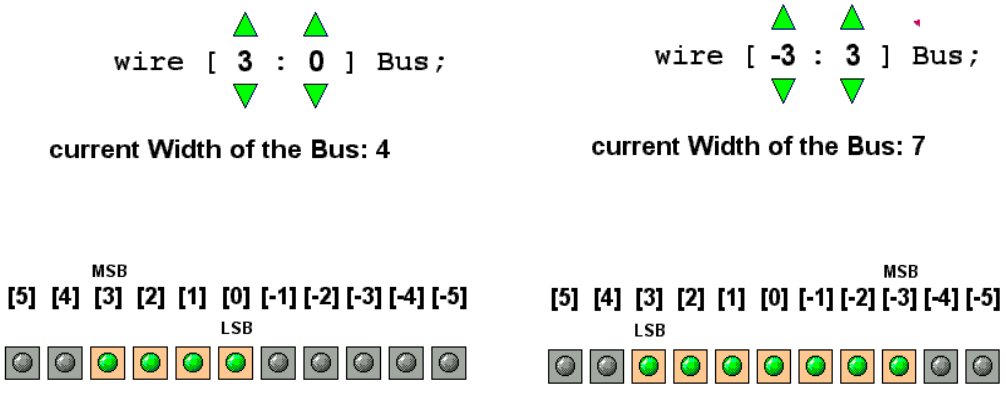
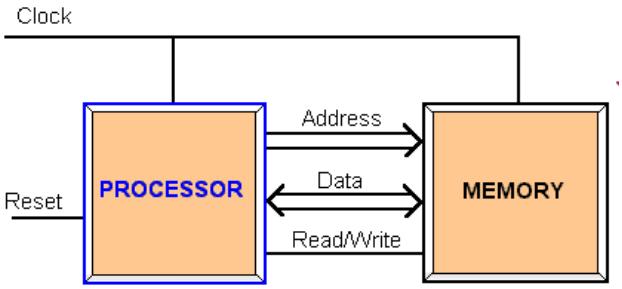
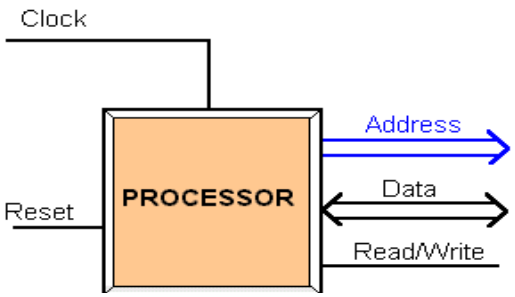


Рис. 7.8. Опис векторів




```
module Processor (Clock, Reset, Read_Write, Data, Address);
input Clock;
input Reset;
output Read_Write;
inout [15:0] Data;
output [19:0] Address;
. . .
endmodule
```

Рис. 7.9. Оголошення портів




```
module Processor (Clock, Reset, Read_Write, Data, Address);
input Clock;
input Reset;
output Read_Write;
inout [15:0] Data;
output [19:0] Address;
. . .
reg [19:0] Address;
endmodule
```

Рис. 7.10. Регістрові порти

7.2.3. Логічні елементи та примітиви у Verilog

Логічні елементи у Verilog описуються стандартними примітивами, наведеними на рис. 7.11. Приклад стандартного примітива наведено на рис. 7.12.

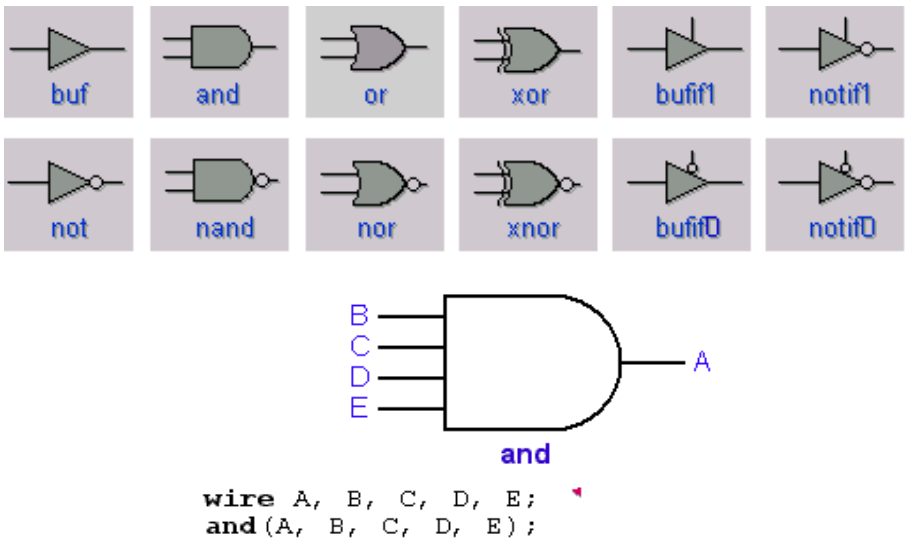


Рис. 7.11. Опис стандартних примітивів

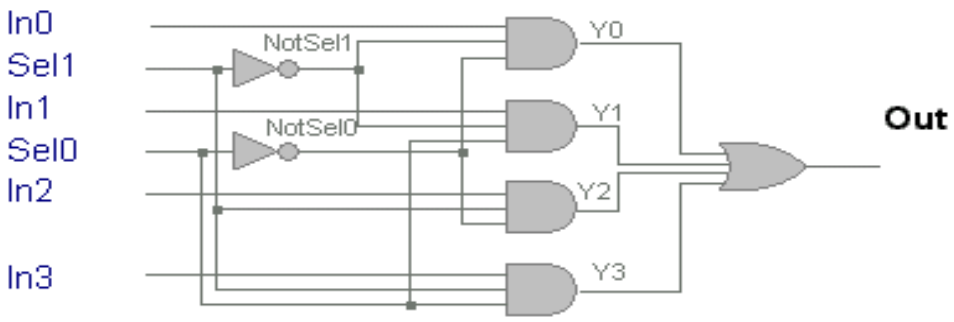


Рис. 7.12. Приклад стандартного примітива

```
module mux_4_to_1 (Out, In0, In1, In2, In3,
Sel1, Sel0);
output Out;
input In0, In1, In2, In3, Sel1, Sel0;
wire NotSel0, NotSel1;
wire Y0, Y1, Y2, Y3;
and (Y1, In1, NotSel1, NotSel0);
not (NotSel0, Sel0);
and (Y3, In3, Sel1, Sel0);
or (Out, Y0, Y1, Y2, Y3);
and (Y0, In0, NotSel1, NotSel0);
not (NotSel1, Sel1);
and (Y2, In2, Sel1, NotSel0);
```

Примітиви, визначені користувачем (UDP), можуть бути комбінаційними (наприклад мультиплексор) або послідовними (наприклад тригер) і загалом є більш складними, ніж стандартні примітиви (щось середнє між стандартним примітивом і модулем). UDP має спеціальну структуру і завжди задається таблицею. Таблиці істинності для комбінаційних і послідовних примітивів мають різні форми: у таблиці послідовних UDP замість одного значення виходу описуються значення виходу в момент часу t і t+1 (рис. 7.13). Приклади побудови примітива мультиплексора і тригерів наведено на рис. 7.14, а-в.

Приклад використання примітивів, визначених користувачем UDP, наведено на рис. 7.15: `udp_name (strengths) #(delays) instance_name [range] (list of ports);`

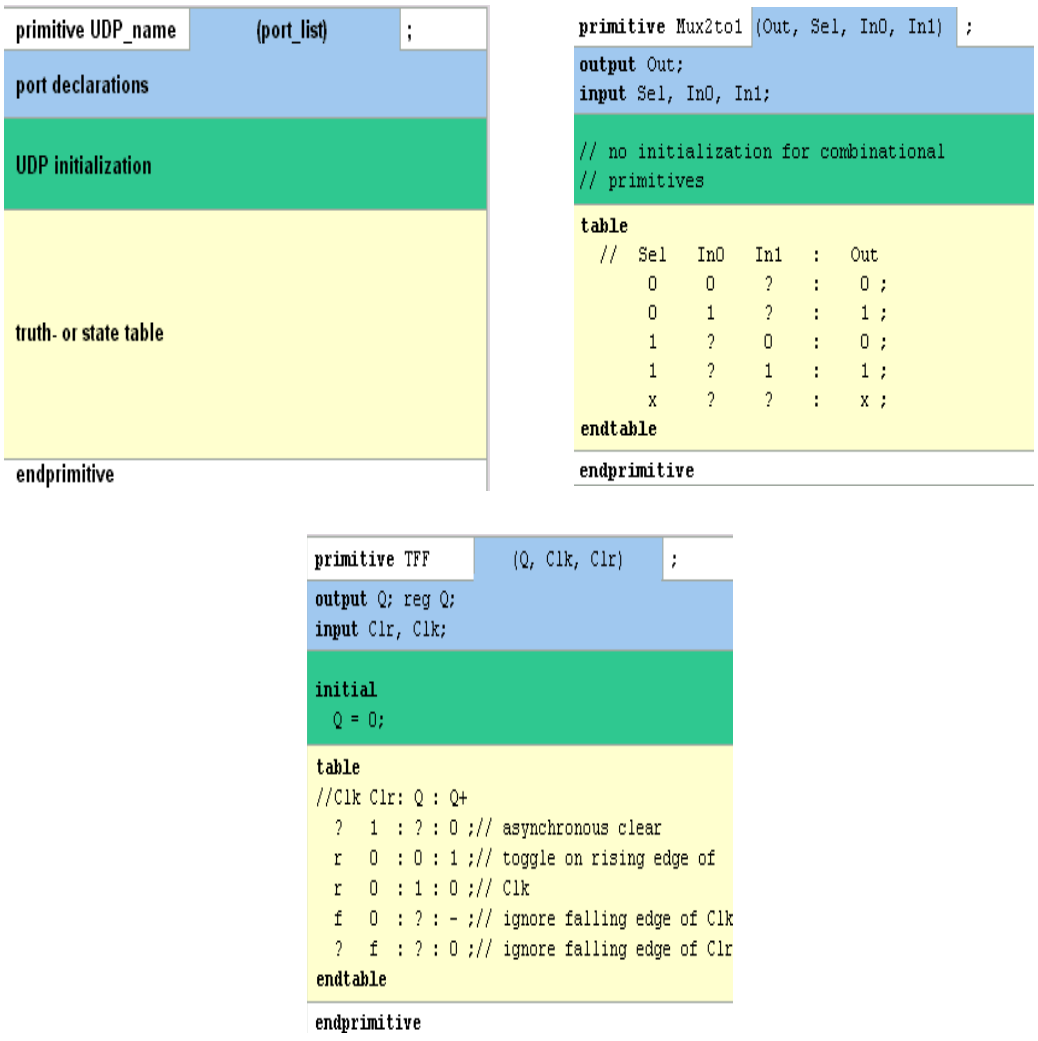
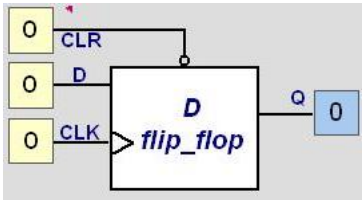


Рис. 7.13. Приклади примітивів, визначених користувачем (UDP)

```
primitive Mux2to1 (Out, Sel, In0, In1);
output Out;
input Sel, In0, In1;
table
  //  Sel   In0   In1   |   Out
  //  ------------------------------
      0     0     ?     :    0 ;
      0     1     ?     :    1 ;
      1     ?     0     :    0 ;
      1     ?     1     :    1 ;
endtable
endprimitive
```

а

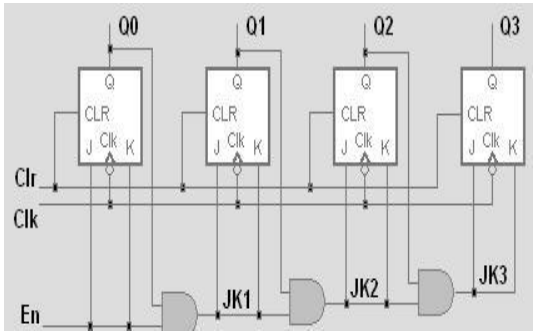


б

```
primitive D_FF (Q, D, Clk, Clr);
  output  Q; reg Q;
  input   D, Clk, Clr;
initial
  Q = 0;          // forces 0 at power on
  table
  //   D   Clk   Clr   :   Q   :   Q+
  //   ---------------------------------
       ?    ?     0    :   ?   :   0 ;
       0    r     1    :   ?   :   0 ;
       1    r     1    :   ?   :   1 ;
       ?    n     1    :   ?   :   - ;
       *    ?     1    :   ?   :   - ;
       ?    ?     p    :   ?   :   - ;
endtable
endprimitive
```

в

Рис. 7.14. Приклади побудови примітивів мультиплексора і тригерів



а

```
module Count4En (Q, Clr, Clk, En);
output [3:0] Q;
wire [3:0] Q;
input Clr, Clk, En;
wire JK1, JK2, JK3;
JKMS Bit0 (Q[0], Clr, Clk, En, En);
JKMS Bit1 (Q[1], Clr, Clk, JK1, JK1);
JKMS Bit2 (Q[2], Clr, Clk, JK2, JK2);
JKMS Bit3 (Q[3], Clr, Clk, JK3, JK3);
and (JK1, Q[0], En);
and (JK2, Q[1], JK1);
and (JK3, Q[2], JK2);
endmodule
```

PRIMITIVE

б

```
primitive JKMS (QOut, Clr, Clk, Jin, Kin);
output Qout; reg Qout;
input Clr, Clk, Jin, Kin;

table
  // . . . appropriate declarations
endtable
endprimitive
```

в

Рис. 7.15. Приклади побудови примітивів

Використання символів при побудові примітивних Verilog-моделей наведено в табл. 7.1.

Таблиця 7.1

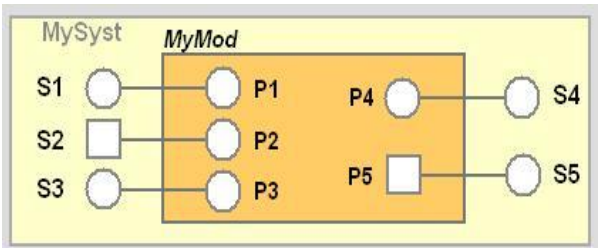
Використання символів при побудові примітивних Verilog-моделей

Символ	Інтерпретація	Коментар
0	Логічний 0	
1	Логічна 1	
x	Невизначене значення	
b	Зміна 0 та 1	Заборонено для виходів
?	Зміна 1 та 1	
-	Зміна 0, 1 та x	Дозволяється тільки для виходів послідовних UPD
(vw)	Перемикання з ***v*** в ***w***	Дозволяється використання тільки зі входами послідовних UPD, v, w, можуть бути 0, 1, x, b
*	Еквівалентно (??)	Будь-яка зміна значення на вході. Дозволяється тільки зі входами послідовних UPD
r	Еквівалентно (01)	Передній фронт на вході. Дозволяється тільки зі входами послідовних UPD
f	Еквівалентно (10)	Задній фронт на вході. Дозволяється тільки зі входами послідовних UPD
p	Еквівалентно (01), (0x), (x1)	Позитивний фронт на вході. Дозволяється тільки зі входами послідовних UPD
n	Еквівалентно (10), (1x), (x0)	Негативний фронт на вході. Дозволяється тільки зі входами послідовних UPD

Примітив може мати тільки один вихід (у списку портів він має стояти першим!).

Опис примітива може розташовуватися до модуля, після модуля, в окремому файлі з використанням директиви компілятора 'include. Модулі можуть реалізувати інші модулі – вони більш гнучкі, на відміну від примітивів.

Типи примитивів, які визначаються користуваем, будемо позначати так: тип **net** – кружком, а тип **reg** - квадратом, як наведено на рис. 7.16.




```
module MySyst (...);
  ...
wire S1,S3,S4,S5;
reg S2;
  ...
PortExmpl MyMod
  (S1,S2,S3,S4,S5);
endmodule
```

```
module PortExmpl
  (P1,P2,P3,P4,P5);
input P1,P2;
inout P3;
output P4,P5;
reg P5;
  ...
endmodule
```

Рис. 7.16. Приклади побудови примітивів

Якщо внутрішній модуль MyMod знаходиться в зовнішньому модулі MySyst, то на з'єднання накладаються обмеження:

1) вхідні порти (input) мають бути типу net, всередині приєднуються до типу reg або net ззовні;

2) двонаправлені порти (inout) мають бути типу net зсередини і ззовні;

3) вихідні порти (output) зсередини можуть мати тип net або reg, а ззовні приєднуватися тільки до типу net.

З'єднання за порядком

```
module dff (clk, d, q);
 input clk, d; output q; reg q;
  always @(posedge clk) q = d;
endmodule module top; reg data, clock;
 wire q_out, net_1;
dff inst_1 (clock, data, net_1);
dff inst_2 (clock, net_1, q_out);
endmodule
```

З'єднання за ім’ям

```
module top;
 reg data, clock;
 wire q_out, net_1;
dff inst_1 (.d(data), .q(net_1), .clk(clock));
dff inst_2 (.clk(clock), .d(net_1), .q(q_out));
endmodule
```

Масив копій модулів

```
module my_module (a,b,c);
 input a, b; output c; assign c = a & b;
endmodule
module top (a,b,c);
 input [3:0] a, b;
 output [3:0] c;
my_module inst [3:0] (a,b,c);
endmodule
```

7.2.4. Способи з'єднання сигналів, константи Integer і Real і тип даних Register у Verilog

Цілі числа (integer). Синтаксис: [sign] [size] 'base number

base: d і D – десяткова, h і H – шістнадцяткова, o і O – вісімкова, b і B – двійкова. D еквівалентно d – десяткова система, C еквівалентно c – вісімкова система.

```
15
'h f
'o 17
'd 15
'b 1111
'b 1_1_1_1
6'd3 // 8 бітове від’ємне число, буде подано в
додатковому коді
4'd 2 // неправильний опис
5'b1_1011
10'd 20
8'h z
```

```
6'o 71
8'b0 (8'b00000000)
8'b1 (8'b00000001)
8'bz (8'bzzzzzzzz)
8'bx (8'bxxxxxxxx)
4'b10?? (4'b10zz)
```

Дійсні числа (real). Синтаксис:

```
sign unsigned_number.unsigned_number
sign unsigned_number.unsigned_number e sign
unsigned_number
sign unsigned_number.unsigned_number E sign
unsigned_number
```

Приклади: `17.5 - 10e5, 0.5 - 0.5694_e 5, 1_000_000.0`

Тип даних register. Синтаксис:

```
reg range list_of_identifiers;
integer list_of_identifiers;
real list_of_identifiers;
time list_of_identifiers;
realtime list_of_identifiers;
```

Приклади:

```
integer counter;
initial counter = 1;
time save_sim_time;
initial save_sim_time = $time;
```

7.2.5. Оператори у Verilog

Оператор	Описание
+ - ! ~	Унарные
* / %	Арифметические
+ - (binary)	Бинарные
<< >>	Сдвига
< <= > =>	Отношения
== != === !==	Эквивалентности
& ~&	and nand
^ ~^ ^~	xor xnor
| ~|	or nor
&&	Логическое and
||	Логическое or
?:	Условный оператор

Арифметические операторы

Оператор	Описание
a + b	a плюс b
a - b	a минус b
a * b	a умножить на b
a / b	a разделить на b
a % b	a по модулю b

Операторы отношения

Оператор	Описание
a < b	a less than b
a > b	a greater than b
a <= b	a less than or equal to b
a => b	a greater than or equal to b

Операторы равенства

Оператор	Описание
a == b	
a != b	
a ===b	
a !== b	

Логические операторы

Оператор	Описание
a && b	Логическое И
a || b	Логическое ИЛИ
!a	Отрицание

Побитовые операторы

Оператор	Описание
~ b	not
a & b	and
a | b	or
a ^ b	xor
a ^~ b, a ~^ b	xnor

Оператори редукції:

```
//x=4'b1010
&x // 1&0&1&0=1'b0
|x  // 1|0|1|0=1'b1
^x  // 1^0^1^0=1'b0
```

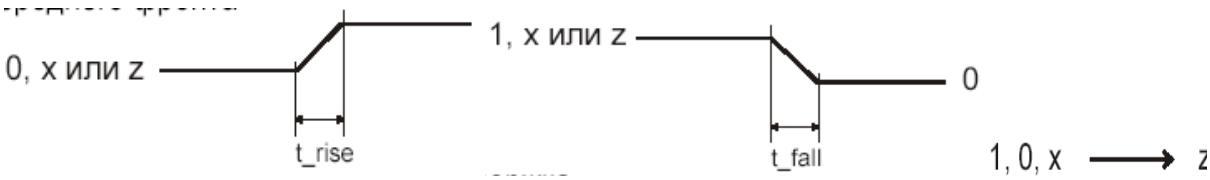
Оператори зсуву:

```
//x=4'b1100
y=x>>1 //y=4'b0110 y=<<x
// y=4'b1000 y=x<<2
//y=4'b0000
```

Оператори конкатенації. Синтаксис:

```
{expression_1, expression_2},
{multiplier{expression}}
//A=1'b1, B=2'b00, //C=2'b10, D3'b110
Y={B, C} //Y=4'b0010
Y={A, B, C, D, 3'b001}
   //Y=11'b10010110001
Y={2{B}, C} //Y=4'b000010
```

Затримки елементів наведено на рис. 7.17.



а б в

Рис. 7.17. Затримки елементів: а – затримка переднього фронту; б – затримка заднього фронту; в – Turn-off затримка

```
    //Задержка для всех переключений
    and #(задержка) a1(out, i1, i2);
    //Задержки переднего и заднего фронта
    and #(rise_val, fall_val) a1(out, i1, i2);
    //Описаны все три типа задержек
and #(rise_val, fall_val, turnoff_val) a1(out,
i1, i2);

    Прикдади:
    and #(5) a1(out, i1, i2);
    and #(4, 6) a1(out, i1, i2);
    and #(3, 4, 5) a1(out, i1, i2);
```

Оператор безперервного призначення у Verilog. Синтаксис:

```
    assign [drive_strength] [delay3] net_lvalue
= expression {,net_lvalue = expression}
```

Приклади:

```
    assign out=i1&i2;
    assign addr[15:0]=addr1[15:0]^addr2[15:0];
    assign {c_out,
sum[3:0]}=a[3:0]+b[3:0]+c_in;
    //Обычная форма
    wire out;
    assign out = i1 & i2;
    //Упрощенная форма
    wire out = i1 & i2;
```

1. З лівого боку допускається використання скалярної або векторної змінної класу net або конкатенація цих елементів.

2. Оператор завжди активний. Як тільки один з операндів з правого боку змінює своє значення, обчислюється нове значення виразу, яке присвоюється змінній.

3. З правого боку операндом може бути reg, net і виклик функції.

4. Властивості затримки подібні до властивостей затримок елементів.

Спрощена форма. У Verilog допускається в одному операторі записувати декларацію лінії і привласнювати їй нове значення.

Також затримка лінії може бути описана в момент її декларації.

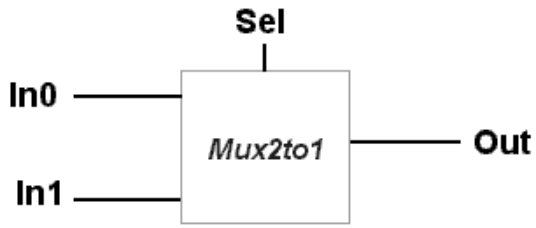
Затримки в операторі призначення. Директива компілятора **timescale.**

```
    `timescale time_unit / time_precision
    time_unit – одиниці вимірювання для
затримок
    time_precision – точність
розрахунку(округлення)
    `timescale 10 ns / 100 ps
    // звичайна форма
    wire out;
    assign #10 out = i1 & i2;

    // Спрощена форма
    wire #10 out = i1 & i2;
    //Затримка лыныъ
    wire #10 out;
    assign out = i1 & i2;
```

Умовне призначення. Синтаксис:

```
condition_expr ? true_expr: false_expr;
```




```
module mux2to1 (out, In0, In1, Sel);
output Out;
input In0, In1, Sel;
assign out = Sel ? In1 : In0;
endmodule
```

Condition_expr – вираз умови. Якщо воно має значення true (логічна 1), обчислюється вираз true_expr. Якщо умова T = false (логічний 0), обчислюється вираз false_expr. Якщо умова один x, то обчислюються обидва вирази true_expr і false_expr і їхні

значення порівнюються побітно. У кожній позиції, у якій біти різні, ставиться x, інакше – значення біта, якщо вони збігаються.

7.2.6. Управління моделюванням у Verilog

Verilog дозволяє зупиняти або переривати моделювання. Синтаксис:

```
$stop [(n)];
$finish [(n)];
```

Examples:

```
$stop;
#150 $finish(2);
```

$ stop – призупиняє моделювання. Необов'язковий параметр n визначає тип виведеного повідомлення. За замовчуванням він дорівнює 1.

$ finish – припиняє моделювання, закриває програму моделювання, передає управління операційній системі. Використовує ті самі параметри, що і $ stop (табл. 7.2).

Таблиця 7.2

Вираз	Повідомлення
0	Відсутне
1	Час моделювання
2	Час моделювання, пам'ять і час CPU, яке використовується для моделювання

Поведінкові моделі – це моделі високого рівня, які описують алгоритм функціонування пристрою без деталізації на логічні вентилі або шляхи передачі даних.

Блоки initial і always – це основні оператори поведінкових моделей. Всі інші поведінкові оператори розміщуються всередині цих структур.

Кожен оператор initial і always являють собою окремий потік, який починає моделювання в момент часу t = 0. Вони не можуть бути вкладеними.

Блоки initial і always – це основні оператори поведінкових моделей. Всі інші поведінкові оператори розміщуються всередині цих структур.

Кожен оператор initial і always являють собою окремий потік, який починає моделювання в момент часу t = 0. Вони не можуть бути вкладеними.

Оператор initial починається з ключового слова initial. Всі оператори всередині нього входять до блока ініціалізації. Починається виконання в t = 0 і виконується один раз. Якщо блоків initial кілька, кожен функціонує і закінчує виконання незалежно від іншого. Кілька операторів записуються між словами begin і and.

Оператори з блока Always починають своє виконання в момент t = 0 і виконуються безперервно подібно циклу. Якщо в модулі нема виразів $ stop і $ finish, моделювання виконується в нескінченному циклі.

Блок initial

```
module stimulus;
reg x, y, a, b;
initial m'b0;
initial
begin
#5 a = 1'b1;
#25 b = 1'b0;
end
initial
begin
#10 x = 1'b0;
#25 y = 1'b1;
end
initial
#50 $finish;
endmodule
```

Блок Always

```
module clock_gen;
reg clock;
// 0
initial
clock = 1'b0;
// 20
always
#10 clock = - clock;
initial
#1000 $finish;
endmodule
```

Процедурні оператори призначення у Verilog. Блокові і неблокові оператори призначення: <lvalue> = <expression>

Lvalue може бути:

- змінною регістрового типу reg, integer, real, realtime і time;
- одним бітом регістрової змінної (addr [0]);
- фрагментом регістрової змінної (addr [31:16]);
- словом пам'яті;
- конкатенацією будь-яких цих елементів.

Блокові оператори призначення. Оператори блокового призначення виконуються в порядку, у якому вони записані в блоці.

```
reg x, y, z;
reg [15:0] reg_a, reg_b;
integer count;
initial
begin
x=0; y=1; z=1; // time 0
count=0; // time 0
reg_a=16'b0; reg_b = reg_a; // time 0
#15 reg_a[2]=1'b1; // time 15
#10 reg_b[15:13]={x, y, z} // time 25
count=count+1; // time 25
end
```

Оператори блокового призначення

```
reg x, y, z;
reg [15:0] reg_a, reg_b;
integer count;
initial
begin
x=0; y=1; z=1;
count=0;
reg_a=16'b0; reg_b = reg_a;
#15 reg_a[2] <= 1'b1; // time 15
#10 reg_b[15:13] <= {x, y, z} // time 10
count <= count+1; // time 0
end
```

Дозволяють моделювати кілька шляхів передачі даних, які мають місце після однієї загальної події.

Як приклад розглянемо після переднього фронту clock.

Приклад 1.

1. Виконання читання розташованих з правого боку змінних (in1, in2, in3, reg1). Обчислюється значення змінної з лівого боку і зберігається програмою моделювання.

2. Виконується запис значень у змінні, розташовані з лівого боку, у порядку, описаному часовими параметрами кожного елемента:

```
reg1 // time 1
reg2 //задній фронт clock
reg3 // time 3
```

Порядок виконання призначень залежить від часових параметрів і не залежить від порядку запису операторів.

3. Порядок виконання записів нових значень не важливий. Програма моделювання зберігає старі значення reg1, які було перелічено при надходженні події.

Приклад 2

```
always @(posedge clock)
begin
reg1 <= #1 in1;
reg2 <= @(negedge clock) in2 ^ in3;
reg3 <= #1 reg1;
end
```

Приклад 3

```
always @(posedge clock)
a = b;
always @(posedge clock)
b = a;
```

Приклад 4

```
always @(posedge clock)
a <= b;
always @(posedge clock)
b <= a;
```

7.2.7. Часовий (Timing Control) і подієвий контроль (Event Control)

- **delay** control (symbol #) - контроль затримкою;
- **event** control, (symbol @) - подієвий контроль;
- **level-sensitive timing** control – чутливий до часу контроль.

Контроль затримкою:

```
<delay>
::= #<NUMBER>
||= #<identifier>
||= #(<min:typ:max> <,<min:typ:max>>*)
```

Контроль затримкою буває трьох типів: regular delay control, intra-assignement delay control, zero-delay control.

Контроль затримкою (**delay** control) вказує час між обчисленням оператора і його виконанням.

Regular delay control

intra-assignement delay control – затримка розміщується в правій частині оператора призначення. Обчислює значення з правого боку в даний час, а привласнення нового значення виконується після зазначеної затримки.

zero-delay control – застосовується для усунення гонок операторів, що виконуються в один час, гарантуючи їх послідовне виконання. Зазвичай не рекомендується використовувати, являє собою зручний механізм для управління порядком виконання операторів при моделюванні.

regular delay control	intra-assignement delay control	zero-delay control
`reg x, y;`	`reg x, y, z;`	`initial`
`initial`	`initial`	`begin`
`begin`	`begin`	`x=0; y=0;`
`x=0;`	`x=0; z=0;`	`end`
`#10 y = 1;`	`y = #5 x + z;`	`initial`
`#y x=x+1;`	`end`	`begin`
`end`		`#0 x=0;`
		`#0 y=0;`
		`end`

Подієвий контроль (Event Control).

regular event control – вираз виконується при зміні сигналу або при виконанні фронту named event control – вимагає декларації події event, потім trigger і recognize.

```
 event OR control
level-sensitive timing control
regular event control
@(clock) q=d;
@(posedge clock) q=d;
@(negedge clock) q=d;
q = @(posedge clock) d;
named event control
event received_data;
```

```
always @(posedge clock)
begin
if (last_data_packet) ->received_data;
end
always @(received_data);
data_buf =
{data_pkt[0],data_pkt[1],data_pkt[2],data_p
kt[3]}
```

event OR control

```
always @(reset or clock or d)
begin
if(reset) q=1'b0;
else q=d;
end
level-sensitive timing control
alwais
wait (count_enable) #20 count = count + 1;
```

7.2.8. Оператор IF, Case, Casez і Casex цикли у Verilog

Оператор IF. Синтаксис:

```
//Type1
if (<expression>) true_statement;
//Type2
if (<expression>) true_statement;
else faulse_statement;
//Type3
if (<expression1>) true_statement1;
else if (<expression2>) true_statement2;
else if (<expression3>) true_statement3;
else default_statement;
faulse_statement;
```

Оператор CASE. Синтаксис:

```
case (expression)
expression: statement1;
expression {, expression}: statement2;. . .
default: default_statement;
endcase
```

Приклад:

```
reg [1:0] address;
case (address)
2’b00 : statement1;
2’b01, 2’b10 : statement2;
default : statement3;
endcase
```

Оператори CASEZ і CASEX

1) `casez (expression)`

```
expression: statement1;
expression {, expression}: statement2;. . .
default: default_statement;
endcase
2) casex (expression)
expression: statement1;
expression {, expression}: statement2; . .
.
default: default_statement;
endcase
```

Приклад:

```
reg [3:0] encoding;
integer state;
casex (encoding)
4'b1xxx: next_state=3;
4'bx1xx: next_state=2;
4'bxx1x: next_state=1;
4'bxxx1: next_state=0;
default: next_state=0;
endcase
```

Цикли. Синтаксис:

```
while (expression) statement;
forever statement;
repeat (expression) statement;
for (assignment; expression; assignment)
statement;
```

Приклад:

1) `integer counter;`

```
initial
for(count=0; count<128; count=count+1)
```

```
2) initial
begin
clock=1'b0;
forever #10 clock = ~clock;
end
```

Блоки дозволяють групувати кілька операторів, виконання яких розглядається як один оператор.

У попередніх прикладах для групування операторів використовувалися ключові слова begin - end.

Блоки бувають послідовними і паралельними.

Послідовні оператори виконуються в тому порядку, у якому вони описані, наступний після виконання попереднього.

Оператори в паралельних блоках виконуються паралельно; порядок виконання операторів контролюється затримками і подіями; затримки і події відносяться до початку виконання блока.

Приклад послідовного блока

```
reg x,y; reg [1:0]
    z, w;
initial
begin
x = 1'b0;
y = 1'b1;
z = {x, y};
w = {y,z};
end
```

Паралельні блоки. Синтаксис:

```
par_block ::=
fork [:block_identifier {
block_item_declaration}]
{ statement }
join
```

Приклад:

```
fork
#50 r = 1'b0;
#100 r = 1'b1;
#150 r = 1'b0;
#200 r = 1'b1;
#250 -> end_wave;
join
```

7.2.9. Приклад опису автомату у Verilog

У записі X/Y X – умова переходу (вхідні сигнали), Y – виконувана дія (вихідні сигнали). Вершини відповідають станам автомата, дуги позначають переходи між станами. Для автоматів типу Мілі X/Y записані біля дуги графа, що позначає перехід між станами. Для автоматів типу Мура X записано біля дуги графа, а

Y – у вершині графа під ім'ям стану. Якщо в однієї дуги записані дві умови переходів, це означає, що в дійсності там дві дуги. Якщо умова переходу не вказана, то означає, що перехід безумовний і буде виконуватися під час вступу синхросигналу.

Наприклад, для пристрою Мілі з входами M, K і виходами Ad, Sh запис M/Ad означає, що перехід буде виконуватися, коли M = 1 (значення K не відіграє ролі), при цьому Ad = 1, а Sh = 0. M' означає M = 0, тобто $\overline{м}$.

Наприклад, для пристрою Мура з входами C, St і виходами Su, Sh запис S3/Su означає, що коли автомат переходить у стан S3, виробляється сигнал Su = 1, при цьому Sh = 0, C' означає C = 0, а C означає C = 1. Якщо дузі (переходу) відповідає C, то C = 1, (значення St не відіграє ролі).

Усі пристрої повинні мати сигнал скидання в початковий стан RESET.

Двоблокова модель автомата, наведена на рис. 7.17, включає один блок always, що обчислює наступний стан і комбінаційні сигнали автомата Мілі, а другий блок - послідовних. Така модель є синтезованою.

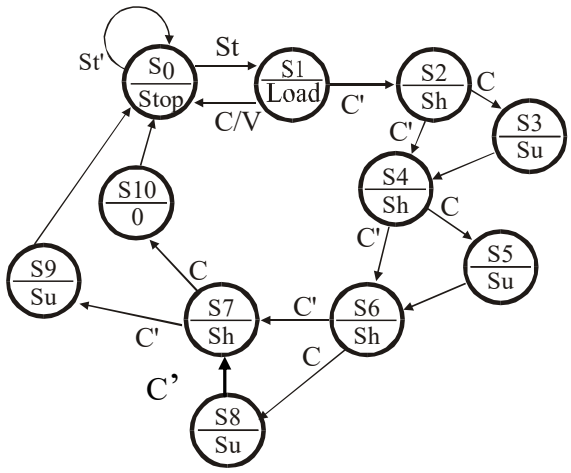


Рис. 7.17. Автомат Мура

```
module FSM (Clk, Reset, St, C, Stop, Load, Sh,
Su, V);
    input Clk, Reset, St, C;
    output Stop, Load, Sh, Su, V;
    reg Stop, Load, Sh, Su, V;
    // Метки состояний
```

```
    parameter S0 = 4'b0000;
    parameter S1 = 4'b0001;
    parameter S2 = 4'b0010;
    parameter S3 = 4'b0011;
    parameter S4 = 4'b0100;
    parameter S5 = 4'b0101;
    parameter S6 = 4'b0110;
    parameter S7 = 4'b0111;
    parameter S8 = 4'b1000;
    parameter S9 = 4'b1001;
    parameter S10 = 4'b1010;
    reg [3:0] state, next_state;
    // блок для послідовної логіки
    always @(posedge Clk or posedge Reset)
           if(Reset) state <=S0;
        else    state <=next_state;
    // блок для комбінаційної логіки
    always @( state or St or C )
    begin
    Stop<=1'b0; Load<=1'b0; Sh<=1'b0; Su<=1'b0;
V<=1'b0;
        case (state)
            S0 : if (St) next_state<=S1;
               else next_state<=S0 ;
             S1 : if (C) begin next_state<=S0;
                    V<=1'b1; end
               else next_state<=S2 ;
             S2 : if (C) next_state<=S3;
               else next_state<=S4 ;
             S3 : next_state<=S4;
             S4 : if (C) next_state<=S5;
               else next_state<=S6;
             S5 : next_state<=S6;
             S6 : if (C) next_state<=S8;
               else next_state<=S7;
             S7 : if (C) next_state<=S9;
               else next_state<=S10;
             S8 : next_state<=S7;
             S9 : next_state<=S0;
```

```
            S10 : next_state<=S0;
        default : next_state<=S0;
        endcase
    Stop <= (state==S0)? 1'b1: 1'b0 ;
    Load <= (state==S1)? 1'b1: 1'b0 ;
    Sh <=
(state==S2)||(state==S4)||(state==S6)||(state==
S7)? 1'b1: 1'b0;
    Su <=
(state==S3)||(state==S5)||(state==S8)||(state==
S9)? 1'b1: 1'b0 ; end endmodule
```

Автомат Мілі з входом enable, що визначають умови переходів станів, наведено на рис. 7.18.

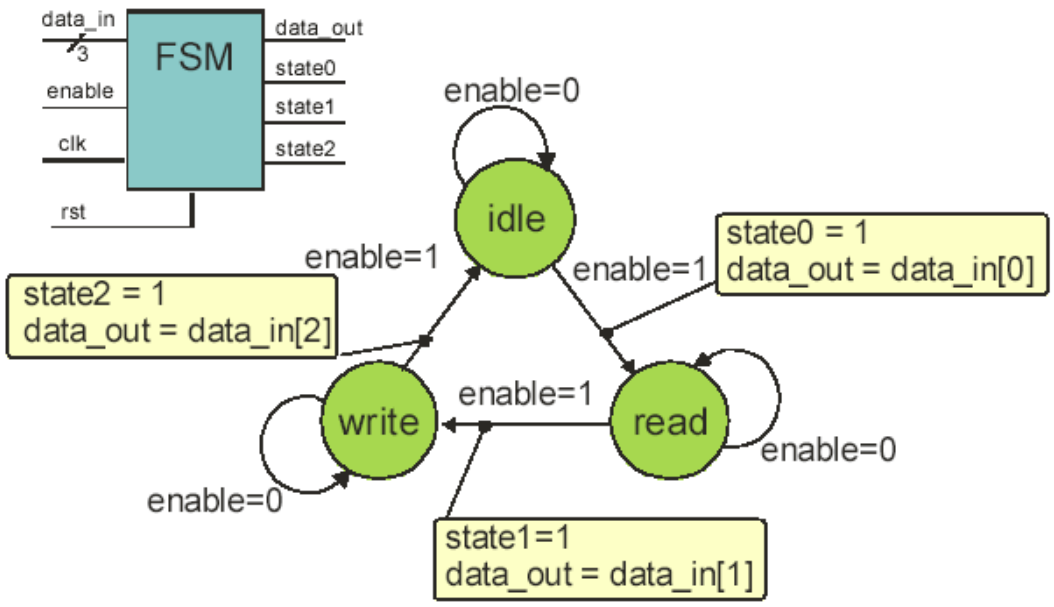


Рис. 7.18. Автомат Мілі з входом enable, що визначають умови переходів станів

```
module FSM1_synplify(clk, rst, enable, data_in,
data_out, state0, state1,state2);
   input clk, rst, enable; input [2:0] data_in;
   output data_out, state0, state1, state2;
   /* Визначені мітки станів*/
   parameter deflt=3'bxxx;
   parameter idle=3'b001;
```

```
   parameter read=3‘b010;
   parameter write=3‘b100;
   reg data_out, state0, state1, state2;
   reg [2:0] state, next_state;
/* Блок Always для послідовної логіки*/
always @(posedge clk or negedge rst)
if (!rst) state <= idle;
else state <= next_state;
/* Блок Always для комбінаційної логіки*/
always @(state or enable or data_in) begin
/* Значення за замовчуванням для виходів
автомата FSM*/
state0 <= 1‘b0; state1 <= 1‘b0; state2 <= 1‘b0;
data_out <= 1‘b0;
case (state)
idle : if (enable)
    begin state0 <= 1‘b1;
data_out <= data_in[0];
          next_state <= read; end
   else next_state <= idle;
read : if (enable)
begin state1 <= 1‘b1;
data_out <= data_in[1];
          next_state <= write; end
   else next_state <= read;

write : if (enable)
   begin state2 <= 1‘b1;
       data_out <= data_in[2];
         next_state <= idle;end
else next_state <= write;
/* Default привласнення для моделювання*/
default : next_state <= deflt;
 endcase end endmodule
```

7.2.10. Завдання і функції у Verilog

Verilog пропонує два типи підпрограм: **Task** (завдання) і **Function** (функції). Це дозволяє розбивати великий код на окремі фрагменти.

TASK використовується коли:

- підпрограма має включати затримки, часовий і подієвий контроль;
- не має або має кілька вихідних аргументів;
- не має вхідних аргументів.

Для декларації аргументів використовуються такі самі ключові слова, що і для портів. Синтаксис:

```
task identifier;
parameter_declaration;
input_declaration;
output_declaration;
inout_declaration;
register_declaration;
event_declaration;
statement;
endtask
```

Приклад **Task**:

```
task first_task;
parameter size=4;
input a;
integer a;
inout [size-1:0] b;
output c;
reg [size-1:0] d;
event e;
begin
d = b;
c = |d;
b = ~b;
if (!a) -> e;
end
endtask
integer x;
reg a, b, y;
reg [3:0] z;
reg [7:0] w;
first_task(x, z, y);
first_task(x, w[7:4], w[1]);
first_task(1,{a,b,w[3], x[0]}, y);
```

Function використовується коли:

- нема затримок, часового і подієвого контролю;
- повертається одне значення;
- є як мінімум один вхідний аргумент.

Регістр з ім'ям identifier (<name_of_function>) оголошується всередині функції. Функція повертає значення з цього регістра.

Параметр <range_of_type> описує ширину всередині регістра. За замовчуванням ширина дорівнює 1. Синтаксис:

```
function type_or_range identifier;
parameter_declaration;
input_declaration;
register_declaration;
event_declaration;
statement;
endfunction
```

Приклад:

```
function real multiply;
input a, b;
real a, b;
multiply = ((1.2 * a) * (b * 0.17)) * 5.1;
endfunction
//...
real a;
initial begin
a = multiply(1.5, a); end
```

Порівняльна характеристика Task (завдання) і Function (функції) наведена в табл. 7.2.

Таблиця 7.2

Порівняльна характеристика Task (завдання) і Function (функції)

Funktion – функція	Task – завдання
1	2
1. Може викликати іншу функцію, завдання нема	1. Може викликати як завдання, так і функції
2. Функція завжди виконується миттєво	2. На виконання завдання може знадобитися ненульовий проміжок часу

Продовження табл. 7.2

1	2
3. Не може містити оператори управління затримками, подіями або часові	3. Може містити оператори управління затримками, подіями або часові
4. Може мати тільки вхідні аргументи input	4. Може мати аргументи типу input, output, inout
5. Завжди повертає єдине значення	5. Не повертає значення, але може передавати їх через аргументи типу input, output, inout

Контрольні запитання

1. За якими ознаками класифікуються запам'ятовуючі пристрої?
2. Які типи пам'яті належать до адресних?
3. Які типи пам'яті належать до послідовних?
4. Які типи пам'яті належать до асоціативних?
5. Чим відрізняється статична пам'ять від динамічної?
6. Назвіть відомі вам твердотільні запам'ятовуючі пристрої.
7. Принципи організації пам'яті за структурою 2D.
8. Принципи організації пам'яті за структурою 3D.
9. Принципи організації пам'яті за структурою 2DM.
10. Структура блокового запам'ятовуючого пристрою.
11. Принцип організації масочних запам'ятовуючих пристроїв.
12. Назвіть типи перепрограмувальних запам'ятовуючих пристроїв.
13. У чому полягає принципова відмінність flash-пам'яті від пам'яті типу *EPROM?*
14. Основні переваги та недоліки багаторівневих комірок flash-пам'яті.

8. ВИВЧЕННЯ ІСНУЮЧИХ НАПРЯМІВ І ТЕНДЕНЦІЙ У ПРОЄКТУВАННІ І ВИРОБНИЦТВІ СУЧАСНИХ ІНТЕГРАЛЬНИХ СХЕМ (ІС)

8.1. Класифікація програмованих логічних інтегральних схем (ПЛІС)

За важливістю справ з точки зору виробництва сучасні ІС можна поділити на стандартні і спеціалізовані (ASIC) (рис. 8.1).

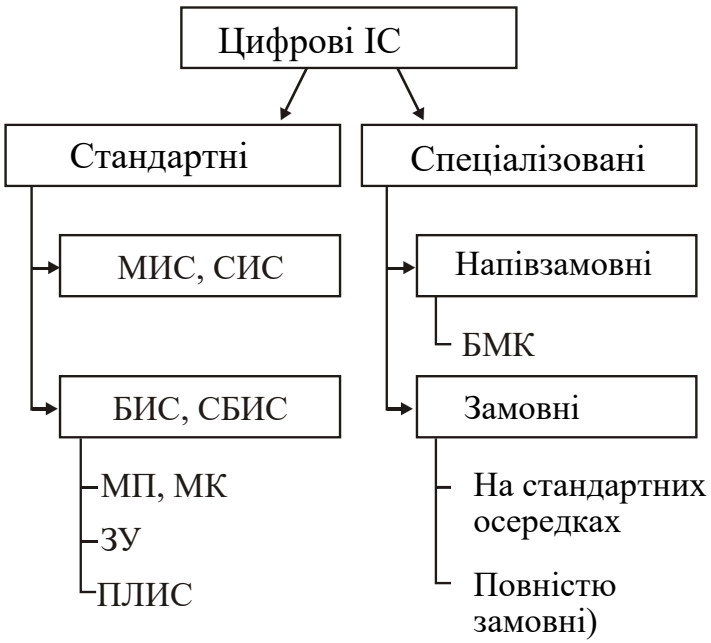


Рис. 8.1. Класифікація цифрових інтегральних схем

Стандартні інтегральні схеми (ІС) мають жорстку внутрішню архітектуру. На їх розроблення витрачаються значні кошти, повернення яких вимагає масових накладів продукції, що випускається. Спеціалізовані ж мікросхеми розробляються за конкретним замовленням. До стандартних відносять такі види мікросхем:

– мікросхеми малого і середнього рівня інтеграції (MIC і CIC) (SSI – Small Scale Integration, MSI – Medium Scale Integration), представлені добре відомими серіями стандартних елементів. У наш час ще іноді використовуються, але побудова складних пристроїв з їх застосуванням неможливо;

– БІС і НВІС мікропроцесорів і мікроконтролерів. Застосовуються для вирішення завдань програмної реалізації алгоритмів;

– БІС і НВІС запам'ятовуючих пристроїв належать до найбільш масових ІС з широкою номенклатурою;

– програмовані логічні ІС (ПЛІС) мають двоякий характер. Для виробників вони є мікросхемами масового виробництва, а для споживачів спеціалізованими пристроями. Крім зазначених, до стандартних мікросхем можна віднести аналого-цифрові і цифро-аналогові перетворювачі.

За способом і витратами на проєктування спеціалізовані мікросхеми (ASIC) можна поділити на напівзамовні і замовні. Різновидами замовних є повністю замовні і спроєктовані за допомогою «стандартних осередків» (Satndart Cell Logic). Останній підхід зараз найбільш популярний, оскільки є більш дешевим. У цьому випадку проєкт будується з використанням існуючих спеціальних бібліотек схемних рішень, які вже добре відпрацьовані. Це спрощує процес побудови фотошаблонів, що знижує вартість створення нових мікросхем. За критеріями площі, швидкості і потужності споживання такі мікросхеми будуть поступатися повністю рекомендованим. Однак розроблення ASIC-проєкту методом стандартних осередків займає у два рази менше часу.

Розроблення замовної мікросхеми дуже дорогий процес. Повний комплект фотошаблонів становить 15-20 штук. Вартість проєктування і виготовлення одного шаблона дорівнює кільком десяткам-сотням тисяч доларів. Вартість витрат на розроблення одного повного комплекту фотошаблонів: для технології 0,18 мкм – 300-350 тис. дол., для технології 0,13 мкм – близько 500 тис. дол., для технології 0,1 мкм – 750 тис. дол.

До напівзамовних мікросхем відносять базові матричні кристали (БМК (MPGA або LPGA)). Це стандартні матриці, на індивідуальних міжз’єднаннях яких за допомогою фотошаблонів (MPGA) або за допомогою лазерного променя виконується руйнування міжз’єднань (LPGA). Вартість і тривалість проєктування порівняно з повністю замовними мікросхемами знижуються в 3-4 рази, але результат ще далі від оптимального. Порівняльна характеристика FPGA і ASIC наведена в табл. 8.1. Ці

характеристики роблять PLD більш привабливими для одних класів задач, а другий – для інших. Багато фірм використовують FPGA на стадії проєктування і підготовки до виробництва, переходячи на ASIC при масовому випуску. Популярність FPGA зростає у зв'язку зі зростаючою вартістю одноразових витрат на проєктування (NRE) для ASIC. Згідно з Gartner Dataquest кількість проєктів FPGA/PLD стартували у 2003 році близько 80 000 порівняно з 3 800 ASIC-проєктів.

Таблиця 8.1

Порівняльні характеристики

Характеристика	FPGA	ASIC
Время создания продукта до появления на рынке	Небольшое	Значительное
Стоимость модулей большого объема	Низкая	Высокая
Единовременные затраты на проектирование	Нет	Высокие
Возможность реконфигурации после создания	Высокая	Нет
Производительность	Средняя	Очень высокая
Плотность	Средняя	Очень высокая
Потребляемая мощность	Высокая	Низкая
Минимальный размер заказа	Нет	Высокий
Сложность процесса проектирования	Средняя	Очень высокая
Сложность тестирования	Низкая	Высокая
Общее время производственного цикла	Часы	Месяцы

Час створення продукту до появи на ринку, вартість модулів більшого об’єму, одночасні затрати на проєктування, можливість ре конфігурації після створення, продуктивність, щільність, споживана потужність, мінімальний розмір замовлення, складність процесу проєктування, складність тестування, загальний час виробничого циклу

За оцінками фахівців фірми Xilinx, промислового лідера у виробництві FPGA доларовий 57-мільярдний ринок інтегральних схем поділяється на 14.0 млрд дол. на ASIC, 3.3 млрд дол. на PLD (ПЛІС), 31.5 млрд дол. на ASSP і 8.5 млрд дол. на інші стандартні логічні елементи. Ринок PLD складається з 2.8 млрд дол. на FPGA і 0.5 млрд дол. на CPLD.

Класифікація ПЛІС відбувається за архітектурою (рис. 8.2), рівнем інтеграції (рис. 8.3), кратністю програмування (рис. 8.4):

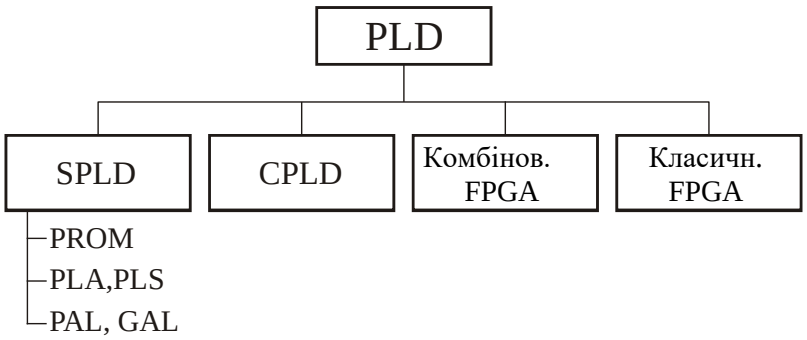


Рис. 8.2. Класифікація ПЛІС за архітектурою

PLD (Programmable Logic Device) – програмований логічний пристрій;

FLEX (Flexible Logic Element Matrix) – матриця елементів гнучкої логіки;

FPGA (Field Programmable Gate Array) – програмована користувачем вентильна матриця;

SPLD (Standart PLD или Simple PLD) – стандартний або простий програмований логічний пристрій;

CPLD (Complex PLD) – складний програмований логічний пристрій;

PLA (Programmable Logic Array) – програмована логічна матриця;

PROM (Programmable Read Only Memory) – перепрограмований постійний запам'ятовуючий пристрій;

PAL (Programmable Array Logic) – програмована матрична логіка;

PLS (Programmable Logic Sequencer) – програмований логічний секвенсер;

GAL (Generic Array Logic) – типова або узагальнена матрична логіка.

Перший клас – це стандартні або прості програмовані логічні пристрої. Його можна поділити на групи: програмовані постійні запам'ятовуючі пристрої (ППЗУ), програмовані логічні матриці (ПЛМ) і програмована матрична логіка (ПМЛ). У першому випадку це таблична реалізація функцій, коли в пам'ять

заносяться значення функції для вхідних наборів відповідних адресі осередків. Наприклад, в осередок з адресою 0000 записується значення функції чотирьох змінних (x_1, x_2, x_3, x_4) для $x_1=0$, $x_2=0$, $x_3=0$, $x_4=0$.

ПЛМ і ПМЛ реалізують диз'юнктивні нормальні форми (ДНФ) перемикальних функцій за допомогою двох матриць: І і АБО. Перша матриця має m входів і формує q термів, друга матриця з q можливих термів формує n функцій. Таким чином, у ПЛМ можна реалізувати n функцій m змінних, що складаються з не більше q термів. У ПМЛ матриця АБО є фіксованою, що не набагато знижує функціональні можливості цих пристроїв, оскільки для основної маси поширених у практиці завдань великий перетин функцій за термами не є типовим.

У складних програмованих логічних пристроях (CPLD) кілька блоків ПМЛ або ПЛМ об'єднуються за допомогою програмованої комутаційної матриці. CPLD може складатися з сотень блоків і десятків тисяч еквівалентних матриць. Фірми-розробники CPLD: Altera, Atmel, Lattice Semiconductor, Cypress Cemiconductor, Xilinx та ін.

Вентильні матриці (FPGA) складаються з великої кількості конфігурованих логічних блоків, розташованих по рядках і стовпцях. При цьому використовується табличний спосіб реалізації функцій. Фірми-розробники FPGA: Xilinx, Actel, Agere System (раніше Lucent Technologies), Quick Logic.

Перші роки після створення архітектури CPLD і FPGA були представлені в чистому вигляді. Кожна з них має свої переваги і недоліки. Прагнення поєднувати їхні переваги в одному пристрої і зростання інтеграції призвело до появи ПЛІС з комбінованою архітектурою. Такі мікросхеми займають проміжне положення між FPGA і CPLD і поєднують властивості обох класів. Існує велика різноманітність таких архітектур. Але, як правило, вони мають структуру, подібну CPLD, і при цьому використовується таблична реалізація функцій. Як приклад перших ПЛІС з комбінованою архітектурою можна назвати мікросхеми серії Flex, що випускаються фірмою Altera.

Системи на кристалі – SoC (System on Chip) або SoPC (System on Programmable Chip). До них відносять мікросхеми великого рівня інтеграції: від сотень тисяч до мільйонів і більше

еквівалентних вентилів. Крім зазначених двох позначень, зустрічаються і інші: PSoC, CSoC, FIPSoC. Часто кожна фірма користується своєю власною абревіатурою та реєструє її як товарний знак.

Класифікація ПЛІС за рівнем інтеграції, наведена на рис. 8.3, відображує останні роки розвитку програмованої логіки. У ній мікросхеми розміром до сотень тисяч еквівалентних вентилів віднесені до досистемного рівня.

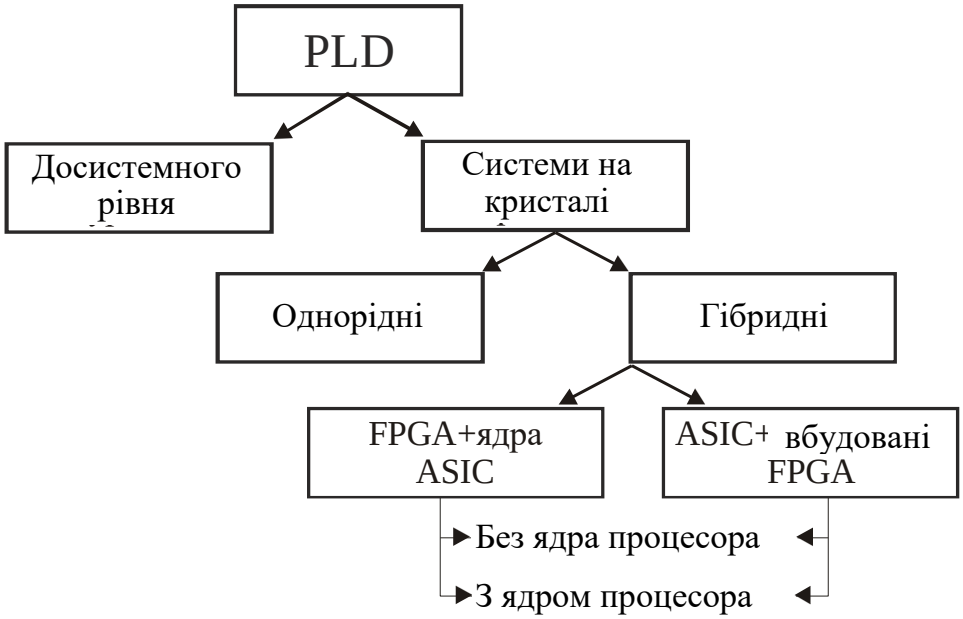


Рис. 8.3. Класифікація ПЛІС за рівнем інтеграції

Клас SoC поділяється на однорідні і блокові (гібридні). Останні мають у своєму складі апаратні ядра, спроєктовані і виконані за технологією ASIC (hard-ядра). Вони реалізують спеціальні функції, наприклад пам'ять, блоки множення, мікропроцесори, різні інтерфейси. Такі пристрої ASIC займають меншу площу кристала і мають більшу швидкість функціонування порівняно з тим, якби вони були реалізовані на FPGA. Зворотним боком є зменшення універсальності мікросхеми і, як наслідок, скорочення кола її споживачів. Це призводить до зменшення обсягів її випуску і подорожчання.

При проєктуванні систем на кристалі використовують «одиниці інтелектуальної власності» - IP (Intellectual Properties). Їх також називають soft-ядрами. Це заздалегідь реалізовані

параметризовані мегафункції, які є повністю синтезованими і можуть розташовуватися в будь-якому місці мікросхеми. Класифікація ПЛІС за кратністю програмування (рис. 8.4) визначається типом тіньової пам'яті, використовуваної для збереження конфігурації мікросхеми.

Рис. 8.4. Класифікація ПЛІС за кратністю програмування

В одноразово програмованих ПЛІС використовуються спеціальні перемички або однозатворні ЛІЗМОП-транзистори.

У ПЛІС декілька поколінь використовувалися плавкі перемички – fuse. У наш час найбільш популярними стали пробивні перемички – antifuse. Вони дуже компактні. Їх площа дорівнює площі перетину двох з'єднуючих ліній. Великих успіхів у розробленні цієї техніки домоглися фірми Alctel (перемички ONO), QuickLogic (перемички ViaLink), Crosspoint Solution (кремнієво-аморфні перемички) і Xilinx (перемички MicroVia).

ЛІЗМОП-транзистори використовують і в одноразово програмованих мікросхемах, і в мікросхемах, програмованих у спеціальних режимах. В англійській літературі пам'ять конфігурації з елементами такого типу називають EPROM-OTP (Electrically Programmable Read-Only Memory – One Time Programmable). У другій групі мікросхеми мають додаткові засобии для репрограмування. Стирати інформацію можна через спеціально зроблене в корпусі мікросхеми вікно за допомогою ультрафіолетового випромінювання. Такі мікросхеми мають

назву РПЗУ-УФ (репрограмовані запам'ятовуючі пристрої з ультрафіолетовим стиранням) або просто EPROM. У мікросхемах з пам'яттю EEPROM (Electrically Erasable Programmable Read-Only Memory) стирання даних здійснюється за допомогою електричних сигналів. У російській термінології EPPROM відповідає РПЗУ-ЕС (репрограмовані запам'ятовуючі пристрої з електричним стиранням).

До пам'яті типу EEPROM близька пам'ять типу Flash. Запам'ятовуючі елементи в обох ідентичні. Різниця полягає в організації процесів запису і стирання даних. Розроблення Flash-пам'яті вважають кульмінаційним пунктом десятирічного розвитку пам'яті типу EEPROM. У сучасних ПЛІС використовуються обидва види електрично стираної пам'яті реконфігурації. У пристроях CPLD застосовується майже виключно цей вид пам'яті.

Останній клас – оперативно репрограмована пам'ять. У них використовується звичайна статична (триггерна) пам'ять – SRAM (Static Random Access Memory). Для конфігурації не потрібні ні програматори, ні спеціальні режими роботи. Осередок пам'яті (рис. 8.5) містить звичайний МОП-транзистор, керований тригером пам'яті конфігурації.

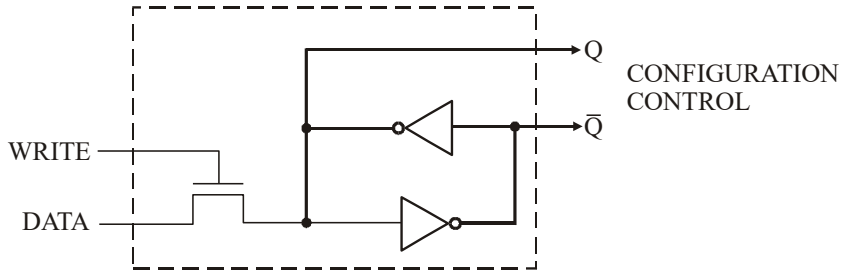


Рис. 8.5. Конфігуруючий осередок пам'яті

Коли сигнал WRITE надходить на передавальний транзистор, значення DATA зберігається в осередку. Кожна точка з'єднання в матриці має пов'язаний осередок пам'яті. Дані, збережені в ній, визначають наявність з'єднання. Програмування з'єднання зводиться до установлення тригера в стан 0 або 1. Зміна конфігурації може проводитися необмежену кількість разів. Така пам'ять є енергозалежною, тому після вимикання живлення конфігурація втрачається. Для її зберігання використовуються

спеціальні мікросхеми енергонезалежної пам'яті, інформація з яких переписується в ПЛІС щоразу після ввімкнення пристрою.

Властивості оперативної реконфігурації SRAM-пам'яті використовуються в розробках архітектур, які активно ведуться, з динамічним репрограмуванням, коли конфігурація системи швидко змінюється в процесі функціонування. Наприклад, у мікросхему в тіньову пам'ять завантажується декілька конфігурацій. Одна з них є активною, перехід до іншої виконується при потребі за один такт синхронізації. Це дозволяє знизити апаратні витрати на вирішення завдання і знизити енергоспоживання пристрою.

Дослідження також ведуться шляхом заміни вичерпаної саму себе, з високою споживаною потужністю SRAM на більш ефективні, енергонезалежні та швидкі типи пам'яті. Більше десятка компаній працюють зараз у цьому напрямі. Найбільш перспективними є магніторезистивна пам'ять, фероелектрична пам'ять і пам'ять на аморфних напівпровідниках. Сфери застосування ПЛІС наведено в табл. 8.2.

Серед провідних фірм у виробництві ПЛІС можна назвати Xilinx, Altera, Lattice і Actel. Основними виробниками FPGA є компанія Xilinx, заснована в 1984 році, і фірма Altera, заснована в 1983 році, виробляє FPGA і структуровані ASIC. Обидві компанії надають повний комплект програмного забезпечення для проєктування за ціною від нуля до декількох тисяч доларів.

Таблиця 8.2

Сфери застосування ПЛІС

Сегмент ринку	Підсегмент	Додатки
1	2	3
Комунікації	Бездротові	Станції сотового зв’язку
		Бездротові LAN
	Дротові	Мережі метрополітену
		Оптичні мережі (Metro area network)
		Модеми DSL
	Мережеві	Комутатори
		Маршрутизатори
		ЗП надвеликої швидкості

Продовження табл. 8.2

1	2	3
Сервери (Storage)	Storage	Мережі Storage area
		Network attached Stf
	Сервери	Високошвидкісні сервери
		Комп’ютери периферії
		ЗП надвеликої швидкості
	Автоматизація конторських робіт	Ксерокси, принтери
Споживачі, виробництво	Споживачі	Газорозрядна індукційна панель
		Цифрове відео
		MP3-плеєри
		Цифрові камери
	Виробництво	Автоматизація виробництва
		Медицинська графіка
		Прилади тестування
	Автоматизація	Мультимедійні системи
		Глобальні навігаційні системи GPS
		Системи розпізнавання речі
	Воєнне виробництво	Супутникове спостереження
		Системи радарної і гідролокації
		Секретний зв'язок

Пакет ISE фірми Xilinx існує в чотирьох видах комплектації: ISE Foundation – 2 495 дол., Alliance – 1 495 дол., Base X – 695 дол. і WebPack – безкоштовний. ISE Foundation включає додаткові функції управління часовими характеристиками проєкту (advanced timing driven implementation tools) разом з можливостями ведення проєкту, синтезу і верифікації. Дешевші пакети мають менше можливостей і підтримують тільки менші за розміром пристрої. Програма ChipScope Pro дозволяє додавати ядра, аналізує логіку в проєктах, що дає можливість користувачеві спостерігати всі внутрішні сигнали і лінії, у FPGA пакет EDK підтримує проєктування систем, містить процесори

IBM PowerPC (hard-ядра) і Xilinx MicroBlaze (soft-ядра). Програма EDK (Embedded Development Kit) коштує 495 дол., а ChipScope Pro – 695 дол.

Xilinx ISE Alliance пакет призначений для користувачів, які хочуть інтегрувати програмне забезпечення фірми Xilinx в існуючі в них засоби EDA.

Altera має програмний пакет Quartus II для проєктування систем на кристалі (SOPC). Quartus II пропонує єдиний процес проєктування для проєктів, що реалізуються на FPGA, CPLD і структурованих ASIC. Підтримує створення вбудованих програм (embedded software programming), синтез, реалізацію, верифікацію і конфігурацію пристрою. Набір програм від Altera включає Quartus II, ModelSim, бібліотеку IP блоків MegaCore, evaluation версію вбудованого процесора Nios. Пакет коштує 2,000 дол. для node-locked ліцензії. Вільна версія Quartus II Web Edition включає більшість властивостей основного пакета, необхідних для проєктування останніх версій CPLD, дешеві FPGA і невеликі за розміром мікросхеми FPGA з сімейств високої щільності. Інші доступні інструменти - SOPC Builder для побудови систем на рівні блоків або компонентів, DSP Builder який має інтерфейс з MATLAB/Simulink.

Altera з партнерами бере участь у програмі Altera Commitment to Cooperative Engineering Solutions (ACCESS) для створення єдиного процесу проєктування систем на кристалі.

8.2. Основні типи FPGA

PLD дозволяють реалізувати послідовні схеми, але не придатні для створення складних цифрових систем. У цьому випадку використовуються більш гнучкі і багатофункціональні програмовані вентильні матриці (PGA) і складні програмовані логічні пристрої, що дозволяють реалізувати великі цифрові системи на одній мікросхемі.

Типова PGA – це мікросхема, що складається з масиву ідентичних логічних осередків з програмованими сполуками. Користувач може запрограмувати функції, реалізовані кожним логічним осередком, і з'єднання між ними. Такі PGA часто

називають FPGA, оскільки вони є програмованими в умовах експлуатації (field-programmable).

Нижче описується внутрішня структура деяких типових FPGA, що випускаються фірмою Xilinx. Розглядаються методи їх програмування для реалізації цифрових логічних схем. У загальних рисах описуються основні етапи проєктування з використанням FPGA. Розглядаються особливості застосування Altera CPDL. Існують два основні класи FPGA: матричні (symmetrical array) і строкові (row-based). Матричні FPGA випускаються фірмою Xilinx. Ці мікросхеми є матрицями логічних елементів, між рядками і стовпцями яких розташовані канали трасування. До строкових FPGA (рис. 8.7) належать мікросхеми фірми Actel. У цих пристроях логічні елементи розташовані у вигляді рядків, між якими знаходяться канали трасування.

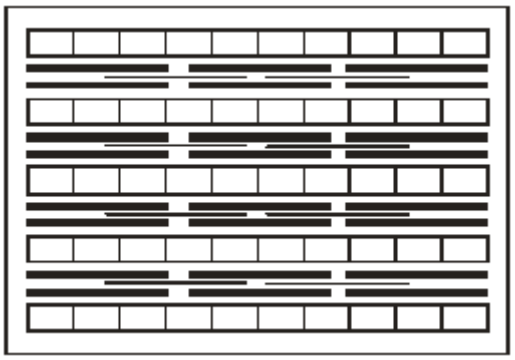


Рис. 8.7. Архітектура строкової FPGA

Крім структурної організації, матричні і рядкові FPGA розрізняються технологією виготовлення. У матричних, як програмованих, елементах використовуються елементи статичного ОЗУ (SRAM), у строкових одноразово перепалювані перемички (antifuse). Матричні FPGA, засновані на SRAM, допускають реконфігурацію, але вимагають додаткових пристроїв для зберігання їхньої конфігурації. Строкові програмуються один раз і не вимагають додаткової апаратури. Перевага строкових пристроїв у тому, що їхні програмовані елементи значно менше, ніж у матричних FPGA, тому вони мають кращі характеристики при трасуванні міжз’єднань, ніж останні.

Розміри логічних елементів строкових FPGA зазвичай менше, ніж матричних. Тому пристрій, реалізований на строковій FPGA, вимагає більше конфігурованих логічних блоків і міжз'єднань, ніж це було б необхідно у випадку матричної FPGA. Але оскільки логічні елементи і міжз'єднання строкових FPGA працюють значно швидше, це не позначається на потужності проєктованого пристрою.

Сучасні мікросхеми фірми Actel мають до 2 млн еквівалентних вентилів і системну частоту 350 МГц. Розмір мікросхем фірми Xilinx досягає 8 млн еквівалентних вентилів. Як приклад FPGA ми розглянемо Xilinx XC3020 Logic Cell Array (LCA). На рис. 8.8 показана її основна структура, що складається з внутрішнього масиву 64 конфігурованих логічних блоків (configurable logic blocks – CLBs), оточених по периметру 64 вхідними/вихідними інтерфейсними блоками. З'єднання між цими блоками можуть програмуватися шляхом збереження даних у внутрішніх конфігурованих осередках пам'яті. Кожен конфігурований логічний блок складається з комбінаційної схеми та двох D тригерів і може бути запрограмований на виконання різних логічних функцій. Конфігуровані осередки пам'яті програмуються після подачі живлення на LCA, і запрограмовані логічні функції і міжз'єднання зберігаються до вимикання напруги. Під час конфігурації осередка пам'яті (рис. 8.9) вибираються по черзі. Коли сигнал WRITE надходить на передавальний транзистор, значення DATA зберігається з осередку. Кожна точка з'єднання в LCA має пов'язаний осередок пам'яті, і дані, збережені в цьому осередку, визначають наявність з'єднання.

На рис. 8.10 подано конфігурований логічний блок. Блок має п'ять логічних входів (A, B, C, D, E), вхід даних (DI), синхровхід (K), вхід дозволу синхронізації (ЄС), безпосереднє скидання (RD) і два виходи (X і Y). Трапецієподібні блоки на схемі позначають мультиплексори, які можуть бути запрограмовані на вибір одного з входів. Наприклад, значення на вихід X можуть надходити або з верхнього тригера (QX), або з виходу F блока «Комбінаційна функція (Combinatorial Function)». Так само значення на вихід Y можуть надходити або з нижнього тригера (QY), або з виходу G і позначає конфігурований осередок пам'яті, і дані в осередку визначають, який вхід мультиплексора буде використаний.

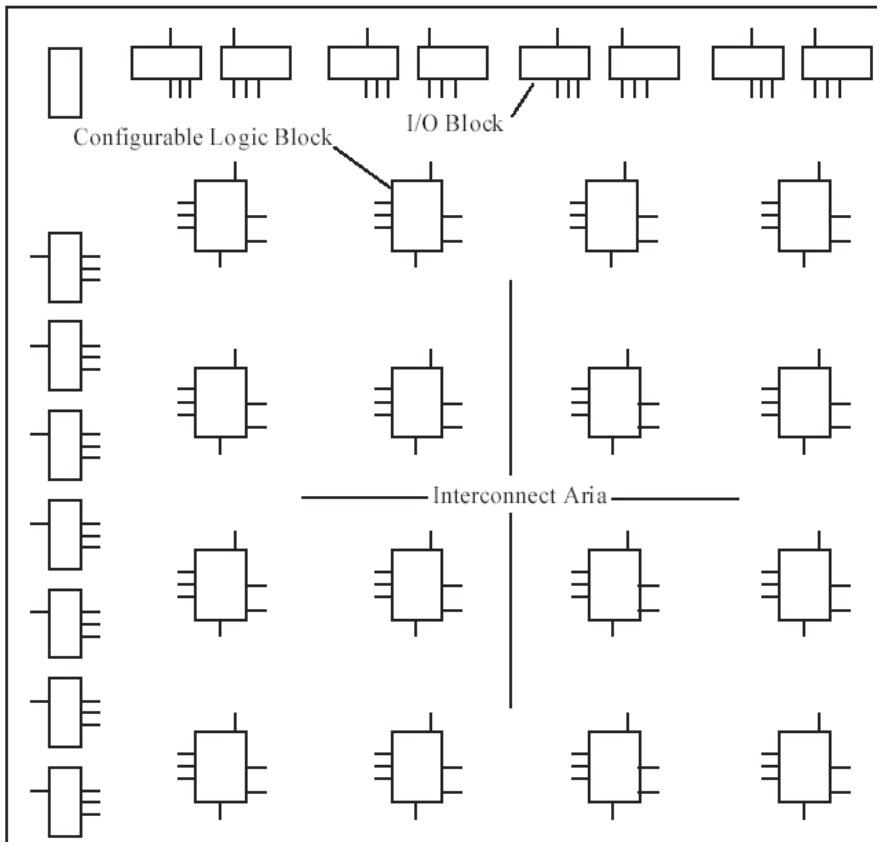


Рис. 8.8. Структурна схема масиву програмованих логічних осередків

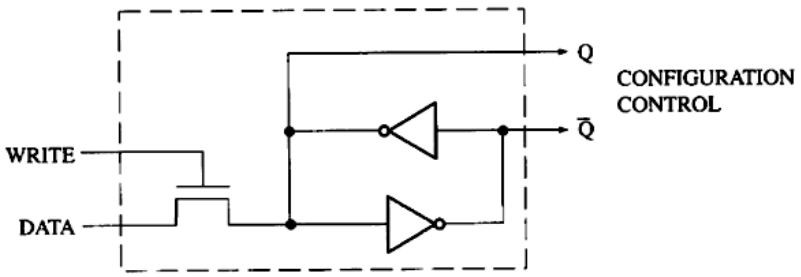


Рис. 8.9. Конфігурований осередок пам'яті

Блок комбінаційних функцій складається з осередків RAM і може бути запрограмований на реалізацію будь-якої функції п'яти змінних або двох функцій чотирьох змінних. Функції зберігаються у вигляді таблиці істинності, таким чином, кількість вентилів, необхідна для їх реалізації, не має значення. На рис. 8.11 подано три можливих операційних режими для функціонального блока.

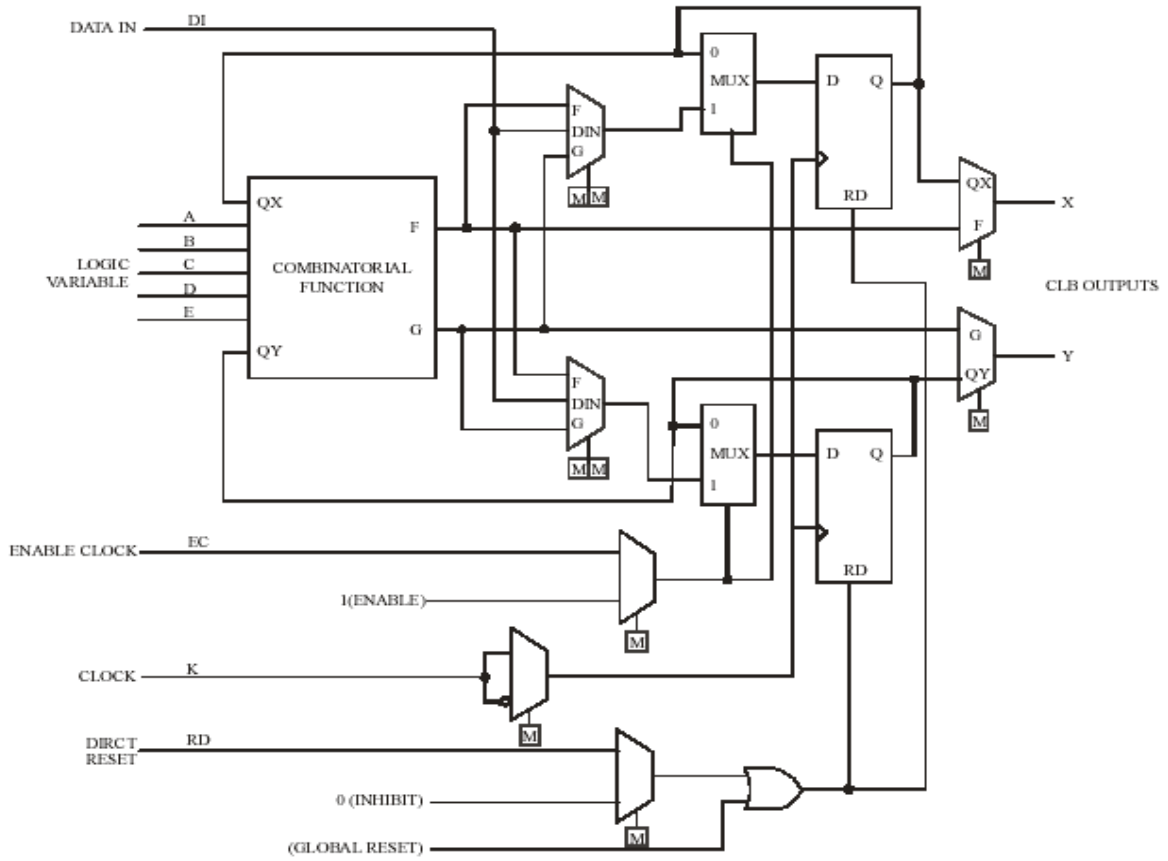


Рис. 8.10. Логічний осередок Xilinx серії 3000

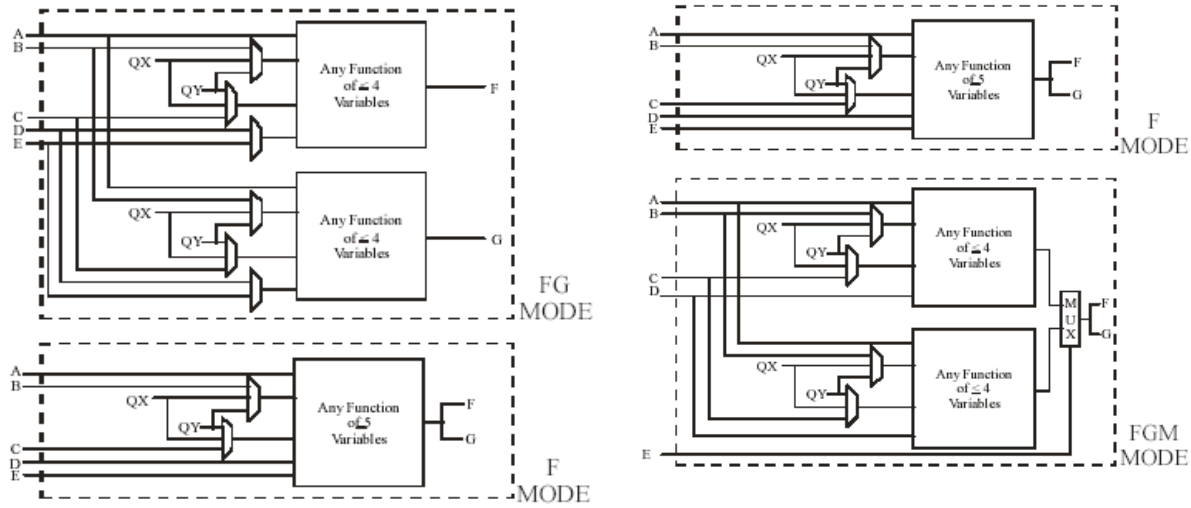


Рис. 8.11. Варіанти режимів комбінаційних схем

Як і вище, кожен трапецієподібний блок позначає мультиплексор, який може бути запрограмований на вибір одного з входів. FG-режим генерує дві функції чотирьох змінних. Одна змінна (A) має бути загальною для обох функцій. Наступні дві

змінні можуть бути обрані з B, C, QX і QY. Останньою змінною може бути або D, або E. Наприклад, можна згенерувати функції F = AB '+ QX E і G = A'C + QYD. Якщо QX і QY не використовуються, то дві функції чотирьох змінних повинні мати три загальних змінних A, B, і C, а четверта змінна може бути D або E.

F-режим може генерувати одну функцію п'яти змінних (A, D, E і дві змінні вибираються з B, C, QX і QY). Можна реалізувати функцію за складністю від простого AND вентиля, F = G = ABCDE до функції парності, яка має 16 термів, якщо розкласти її на диз'юнкцію і кон'юнкцію. Режим FGM використовує мультиплексор з контрольним входом E для вибору однієї з двох функцій чотирьох змінних. Кожна функція використовує входи A, D і два з входів B, C, QX і QY. У режимі FGM можна реалізувати деякі функції шести і семи змінних. Наприклад, у цьому режимі можна реалізувати функцію семи змінних $F=G=E(AB'+QXD) + E'$ (A'C+ QYD).

D-вхід кожного тригера може бути запрограмований на отримання даних з F, G або DI входу. Обидва тригери мають спільний вхід синхронізації. Мультиплексор MUX пов'язаний з входом CLOCK (K) може бути запрограмований на вибір K або K', таким чином, D тригер може перемикатися по передньому або задньому фронту синхросигналу. Синхроімпульс може бути або завжди дозволений, або може управлятися входом Enable Clock (EC). Мультиплексор MUX зв'язує D вхід кожного тригера, використовується для вимикання синхронізації. Якщо EC = 0, сигнал з Q виходу надходить назад на D вхід, таким чином, $Q^+ = Q$, і тригер ніколи не змінює стан навіть за наявності синхроімпульсу. Якщо EC = 1, D вхід з'єднується з F, G і DIN, і стан тригера змінюється за відповідним фронту синхросигналом. З'єднання D тригера і мультиплексора еквівалентно D тригеру з входом дозволу синхронізації (E0 вхід, як показано на рис. 8.12). Оскільки Q може змінювати значення тільки коли EC = 1, наступне характеристичне рівняння описує поведінку тригера: $Q^+ = EC\ D+EC'Q$.

Використання тригерів такого типу усуває необхідність у стробуванні синхровходу за допомогою керуючого сигналу. Оскільки синхроімпульс може безпосередньо надходити на вхід кожного тригера, дуже легко можна отримати правильні операції

синхронізації. Тригери мають вхід асинхронного скидання (RD). Тригер скидається в 0 (якщо скид не заборонений) при надходженні 1 на лінію. Очистити все – очищує тригери в усіх осередках мікросхеми.

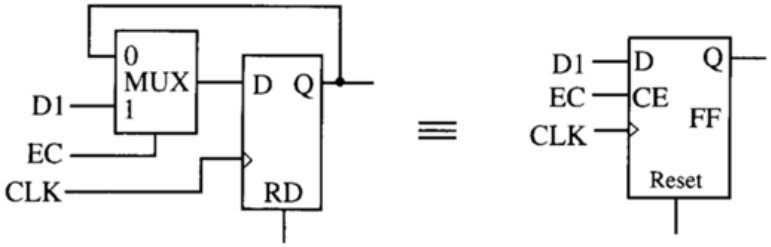


Рис. 8.12. Тригер з дозволом синхронізації

Для прикладу ми реалізуємо пристрій паралельного додавання/віднімання з накопиченням, використовуючи XC3020. Якщо Ad = 1, значення з B входу буде додано до акумулятора, якщо S_u = 1 – вирахувано. Віднімання буде виконуватися додаванням додаткового коду з B з акумулятором. Якщо $A_d = S_u = 0$, вміст акумулятора не зміниться. Ми будемо формувати додатковий код, взявши зворотний код з B і встановивши 1 на вхід перенесення першого повного суматора.

Оскільки кожен логічний осередок має два тригери, можна реалізувати два біти акумулятора в одній комірці. Однак якщо два біти будуть реалізовані в одній комірці, необхідно два тригери акумулятора плюс вихід перенесення в наступний осередок. Оскільки кожна клітинка має тільки два виходи, така схема не буде працювати. Тому можна реалізувати тільки один біт на осередок.

На рис. 8.13 показано типовий осередок пристрою паралельного додавання/віднімання. Логічні входи b_i, c_i (перенесення з попередньої комірки) і Su. Значення з виходу тригера акумулятора (a_i) подається назад у клітинку.

Блок комбінаційної функції реалізує такі рівняння.

Якщо Su = 0, ці рівняння скорочуються до стандартних рівнянь повного суматора з $x_i = a_i$ і $y_i = b_i$. Якщо S_u = 1, b_i доповнюється за допомогою виключного АБО. Якщо перенесення в наймолодший значущий біт також пов'язане з S_u, коли S_u = 1 додатковий код з B складається з A, таким чином, виконується віднімання. Оскільки F і G – функції одних і тих

самих чотирьох змінних, ми знаємо, що вони можуть бути реалізовані блоком комбінаційних функцій у режимі FG з рис. 8.11. На рис. 8.13 c_i і b_i з'єднані з A і B входами блока, таким чином, внутрішній зворотний зв'язок з a_i тригера (QX) має йти на третій вхід блока. Потім залишився вхід S_u, він може бути пов'язаний з D або E входом блока. Оскільки акумулятор має змінюватися, коли $A_d = 1$ або $S_u = 1$, ми з'єднаємо вхід дозволу синхронізації (EC) з сигналом $A_d + S_u$. Елемент АБО з іншого логічного осередка генерує цей сигнал, що використовується всіма осередками додавання/віднімання.

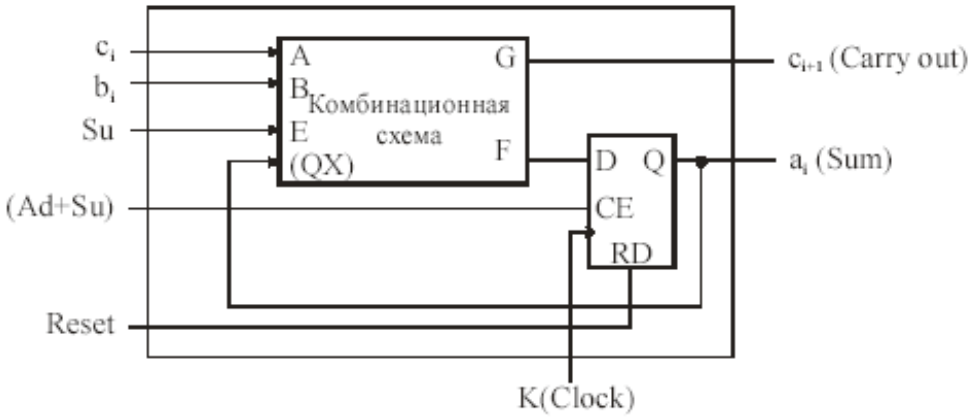


Рис. 8.13. Логічний осередок пристрою паралельного додавання/віднімання

Пунктирною лінією на рис. 8.14 виділено шляхи сигналів у логічному осередку після програмування. F-функція зв'язується з D входом тригера акумулятора (ai), і G-функція зв'язується з перенесенням у наступний розряд (c_{i+1}).

Рис. 8.15 ілюструє реконфігурований вхідний/вихідний блок (input-output block – IOB). Контакт I/O зв'язується з однією з ніжок корпусу мікросхеми, таким чином, зовнішній сигнал може надходити або виходити з матриці логічних осередків. Щоб використовувати осередок як вхід 3-state, керуючий сигнал має бути встановлений у забезпечувальний буфер. Щоб використовувати осередок як вихід, тристабільний буфер має бути вимкнений. Тригери забезпечують, щоб вхідні та вихідні значення могли бути збережені в IOB. Тригери блокуються, коли необхідно пряме з'єднання входів або виходів. Дві лінії синхронізації (CK_1 і CK_2) можуть бути запрограмовані на з'єднання з будь-яким з двох тригерів.

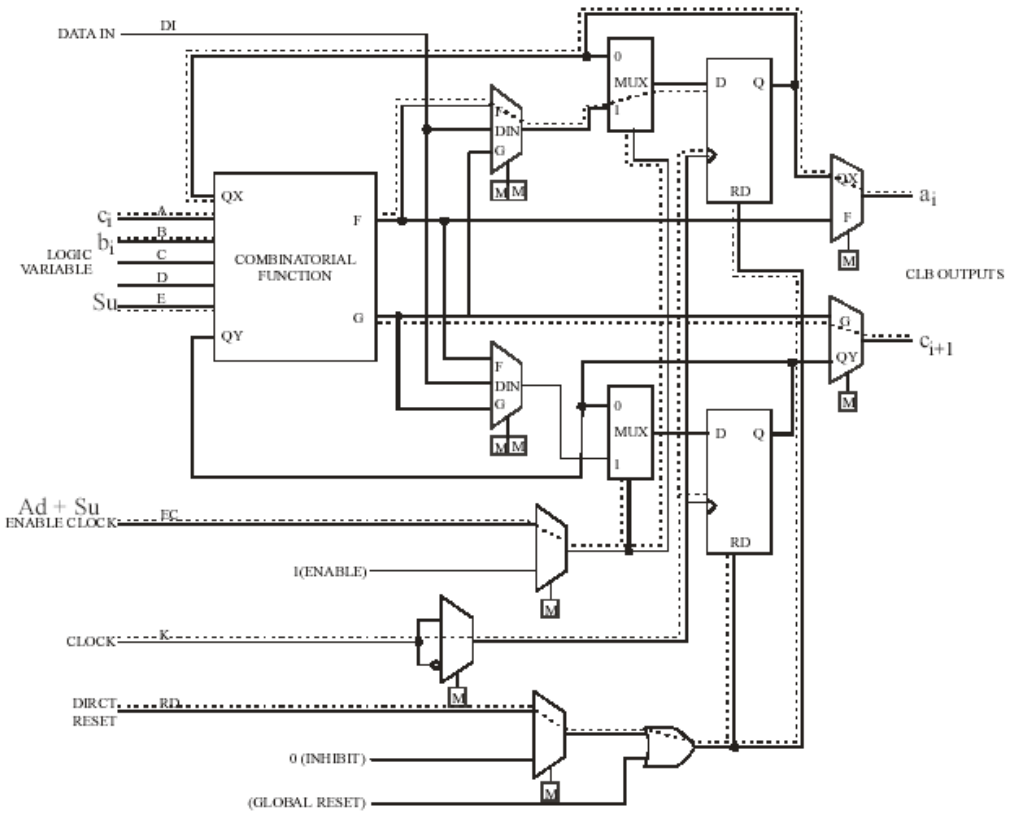


Рис. 8.14. Шляхи сигналів у логічному осередку пристрою додавання/віднімання

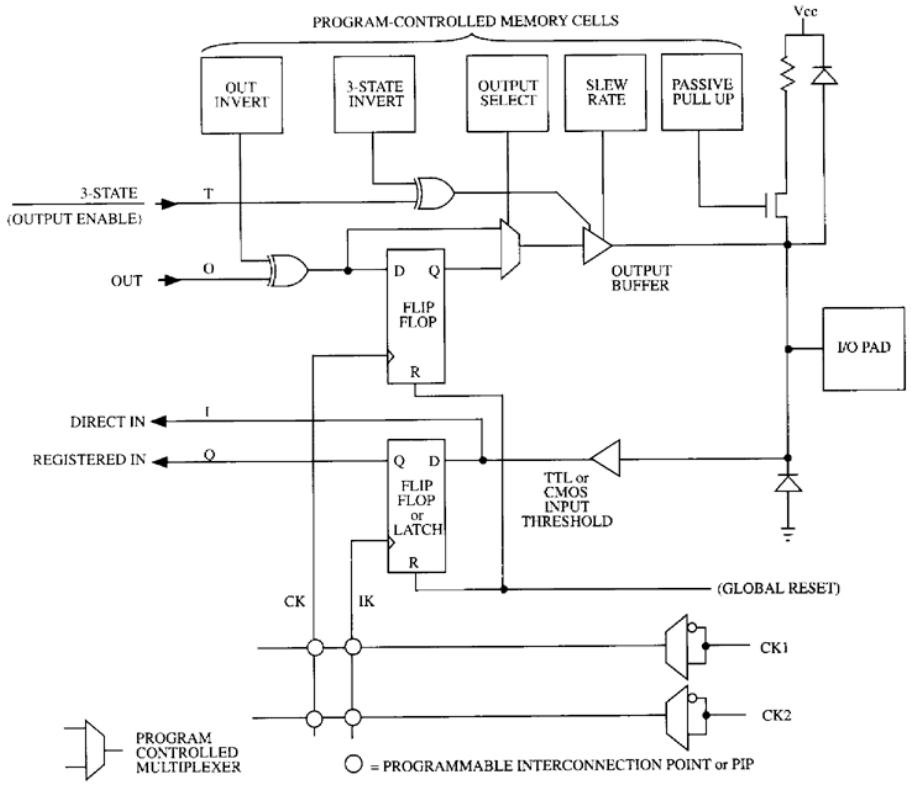


Рис. 8.15. Вхідний/вихідний блок Xilinx серії 3000

Вхідний тригер може бути запрограмований на роботу як тригер, що перемикається за фронтом D або явний тригер-засувка. Навіть якщо I/O контакт не використовується, I/O тригери тим не менш можуть використовуватися для збереження даних.

Вихідний OUT сигнал йде з логічного масиву, спочатку проходить через вентиль виключного АБО, де він доповнюється чи ні залежно від того, як запрограмований OUT-INVERT біт. Вихідний OUT сигнал може бути збережений у тригері за бажанням. Програмування OUTPUT-SELECT біта визначає вихідний OUT сигнал або вихід тригера з'єднується з вихідним буфером. Якщо 3-STATE сигнал дорівнює 1 і біт 3-STATE INVERT дорівнює 0 (або якщо 3-STATE сигнал дорівнює 0 і 3-STATE INVERT біт - 1), вихідний буфер має на виході значення високого імпедансу. В іншому випадку буфер передає вихідний сигнал на I/O контакт. Коли I/O контакт використовується як вхід, вихідний буфер має бути в стані високого імпедансу. Зовнішній сигнал йде з контакту I/O через буфер і потім надходить на D тригер. Буфер виходу подає сигнал DIRECT IN в логічний масив. Інакше вхідний сигнал може бути збережений у D-тригері, з якого подається REGISTERED IN сигнал у логічний масив.

Кожен IOB має вхідні/вихідні опції, які можуть бути обрані конфігурацією осередків пам'яті. Вхідна порогова величина може бути запрограмована на рівень сигналів TTL або CMOS. Біт SLEW RATE управляє частотою, з якою вихідний сигнал може змінюватися. Коли вихід управляє зовнішніми пристроями, зменшення швидкості наростання вихідної напруги бажано для зменшення наведений шумів, які можуть з'являтися при швидкій зміні вихідного сигналу. Коли встановлено PASSIVE PULL-UP біт, навантажувальний резистор з'єднується з I/O pad. Внутрішній навантажувальний регістр може бути використаний для усунення рухомих входів.

Програмовані з'єднання між конфігурованими логічними блоками і вхідними/вихідними блоками можуть бути зроблені декількома способами – універсальні міжз’єднання (general-purpose interconnect), пряме міжз’єднання(direct interconnect) і довгі лінії (long line). Рис. 8.16 ілюструє систему універсальних міжз’єднань. Сигнали між CLB або CLB і lOB можуть бути

спрямовані через матриці перемикачів, як вони проходять через горизонтальні і вертикальні з'єднувальні лінії. Прямі з'єднання можливі між сусідніми блоками CLB, як показано на рис. 8.17. Довгі лінії забезпечують з'єднання CLB розташованих далеко один від одного. Всі міжз’єднання програмуються збереженням біта у внутрішніх конфігурованих осередках пам'яті в LCA.

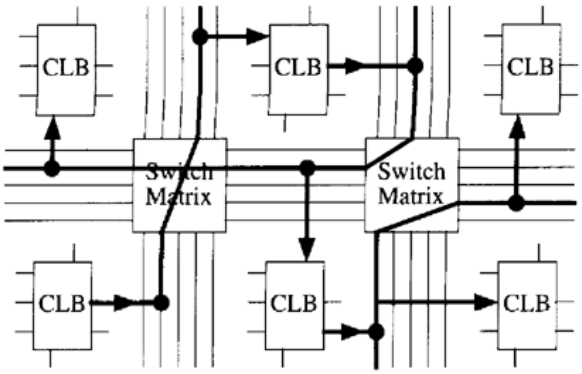


Рис. 8.16. Універсальні міжз’єднання (general-purpose interconnect)

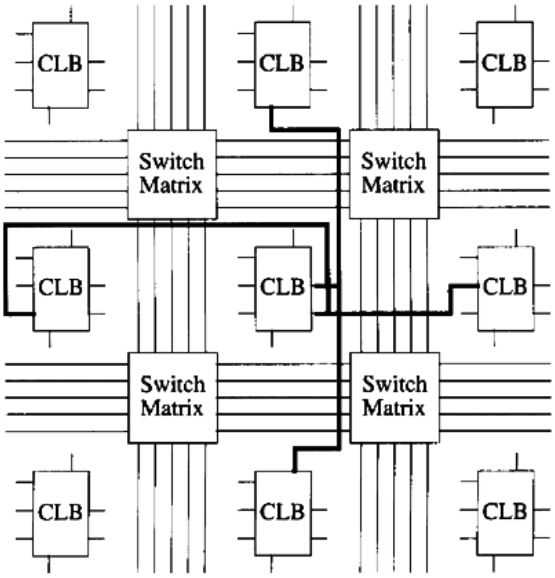


Рис. 8.17. Прямі міжз’єднання між сусідніми блоками CLB

Довгі лінії забезпечують високий коефіцієнт розгалуження за виходом, низький розподіл розфазування імпульсів (low-skew distribution), які мають проходити відносно великі відстані. На

рис. 3.18 показано чотири вертикальних довгих лінії між двома сусідніми рядами CLB, дві з них можуть бути використані для синхронізації. Також подано дві горизонтальні лінії, які працюють між двома сусідніми рядами CLB. Довгі лінії охоплюють повністю довжину або ширину з'єднувальної області.

Кожен логічний осередок має два сусідніх тристабільних буфери, які з'єднуються з горизонтальними довгими лініями. Проєктувальники можуть використовувати ці довгі лінії і буфери для реалізації тристабільних шин. На рис. 8.19, а як тристабільні буфери можуть бути використані для мультиплексування сигналів у горизонтальній довгій лінії. Ці буфери мають вихідний дозволяючий сигнал низького рівня, так, коли A = 0, сигнал D ^ надходить на лінію. Weak keeper circuit в кінці лінії пам'ятає останнє значення, яке надійшло на лінію, таким чином, воно ніколи не загубиться. Увага має бути спрямована на усунення конфліктів шини, які можуть статися, якщо значення 0 і 1 надійдуть на шину в один і той самий час. Тристабільні буфери можуть бути також використані для реалізації функції монтажного I, як показано на рис. 8.19, б. Коли один або більше входів D мають значення 0, лінія отримує значення 0. Коли всі D входи мають значення 1, всі виходи буферів знаходяться в стані високого імпедансу, і навантажувальний резистор виштовхує лінію в стан 1.

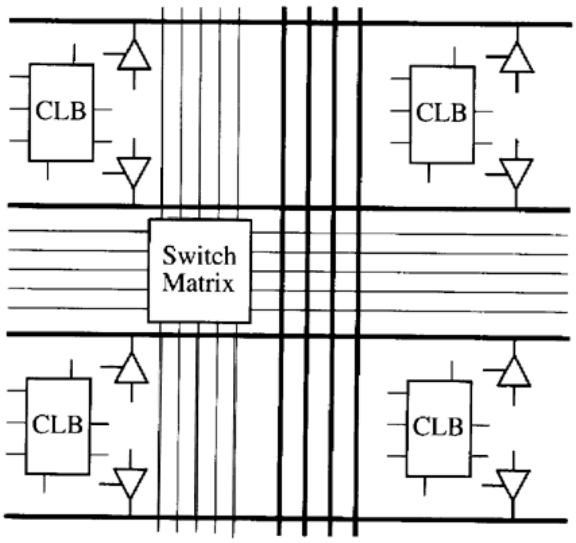
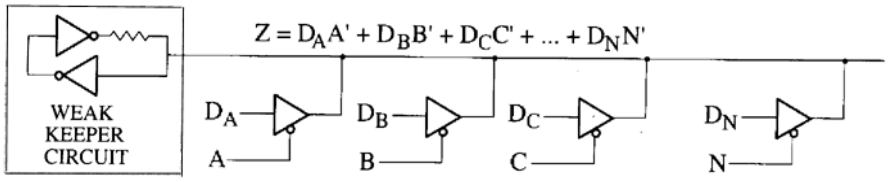
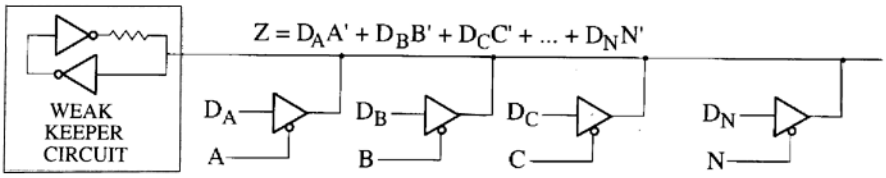


Рис. 8.18. Вертикальні і горизонтальні довгі лінії



а



б

Рис. 8.19. Використання тристабільних буферів:
а – реалізація мультиплексора; б – реалізація монтажного І

Кварцовий генератор може бути реалізований, використовуючи внутрішній швидкодіючий інвертований буфер разом із зовнішнім кристалом (Yl), резистором і конденсаторами, як показано на рис. 8.20. Зовнішні компоненти з'єднуються з двома контактами IOB, і вихідний генератор з'єднується з альтернативним буфером синхронізації (alternate clock buffer). Альтернативний буфер синхронізації управляє горизонтальними довгими лініями, які у свою чергу можуть бути використані для управління вертикальними довгими лініями і входами синхронізації K (clock) логічних блоків. Якщо використовується зовнішня синхронізація, вихід синхронізації з'єднується з загальним буфером синхронізації (global clock buffer). Цей буфер управляє всією схемою, що забезпечується високим коефіцієнтом розгалуження за виходом, синхронізуючим синхровхід усіх IOB і логічних блоків. Якщо потрібна симетрична синхронізація, до виходу генератора може бути підключений тригер, який ділить частоту на 2.

Мікросхема XC3020 FPGA, яка тут розглядається, має 64 CLB (8 х 8), 64 призначених для користувача входів / виходів, 256 тригерів (128 в CLB і 128 в lOB), 16 горизонтальних довгих ліній, і 14779 конфігурують біт даних. Інші члени сімейства XC3000 мають до 484 CLB (22 х 22), 176 призначених для користувача I / O 1320 тригерів, 44 горизонтальних довгих ліній, і 94984 конфігурують біт даних.

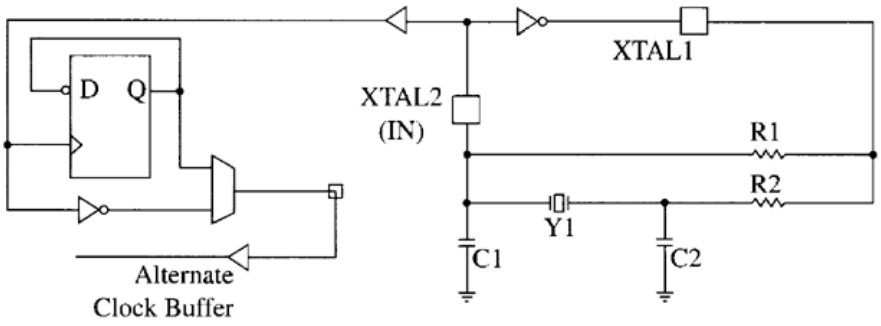


Рис. 8.20. Кварцовий генератор

8.3. Проєктування з FPGA

Доступні складні CAD інструменти сприяють проєктуванню систем, які використовують програмовані матриці вентилів. Один з методів проєктування цифрових систем з FPGA включає такі кроки:

1. Створити блок-схему цифрової системи. Визначити умови та керуючі сигнали і створити діаграму станів об'єкта (SM chart) або граф станів, що описують необхідну послідовність операторів.

2. Створити VHDL опис системи. Промоделювати і налаштувати VHDL код, зробити всі необхідні корекції проєкту, що був розроблений на кроці номер 1.

3. Визначити логічні складові проєкту системи, використовуючи вентилі, тригери, регістри, лічильники, суматори і т. д.

4. Ввести логічну схему системи в комп'ютер, використовуючи програми введення схеми. Промоделювати і налаштувати логічну схему і зробити необхідні виправлення в проєкті, створеному на кроці 3.

5. Виконати програму декомпозиції. Ця програма розбиває логічну схему на частини, які можна реалізувати на конфігурованих логічних блоках.

6. Виконати програму автоматичного розміщення та розведення (place and routeprogram). Вона розміщує логічні блоки на відповідні місця в FPGA і створює з'єднання між логічними блоками.

7. Виконати програму, яка генерує конфігурацію бітів (bit pattern), необхідну для програмування FPGA.

8. Завантажити конфігурацію бітів у внутрішні конфігуровані осередки пам'яті в FPGA і протестувати роботу FPGA.

Наявні в розпорядженні автоматичні синтезуючі інструменти використовують як вхідні дані VHDL опису і генерують міжз'єднання вентилів і тригерів для реалізації системи. Використання таких інструментів ефективно автоматизує кроки 3 і 4. Однак необхідно використовувати деякі обмеження в VHDL описі для того, щоб зробити його синтезуючим.

Коли кінцева система побудована, конфігурація бітів для програмування FPGA зазвичай зберігається в EPROM і автоматично завантажується в FPGA коли ввімкнена напруга. EPROM з'єднується з FPGA, як показано на рис. 8.21. FPGA скидає себе після застосування напруги. Потім зчитує дані конфігурації з EPROM, подаючи послідовність адрес на входи EPROM і зберігаючи вихідні дані з EPROM у внутрішніх конфігурованих осередках пам'яті FPGA.

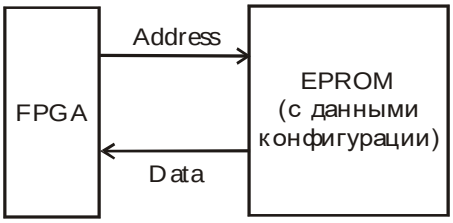


Рис. 8.21. Підключення EPROM для LCA ініціалізації

Реалізація функцій шести і більше змінних. Хоча деякі логічні функції шести змінних можуть бути реалізовані на одному або двох логічних осередках, у загальному випадку функція шести змінних вимагає три осередки. Загальний метод реалізації будь-якої такої функції спочатку використовує її розкладання, як показано нижче:

$$Z(a,b,c,d,e,f)=a'Z(0,b,c,d,e,f)+aZ(1,b,c,d,e,f\}= \\ = a'Z0+aZ1. \qquad (8.1)$$

Це приклад теореми розкладання Шеннона. Можна перевірити, що рівняння (8.1) правильне, встановивши a в 0, а

потім в 1. Якщо воно правильно для a = 0 і для a = 1, то воно завжди правильне.

На підставі рівняння (8.1) можна розробити схему, у якій задіяно два осередки для реалізації Z_0 і Z_1 (рис. 8.22, а). Половина третього осередку використовується для реалізації функції трьох змінних: $Z = a'Z_0 + aZ_1$.

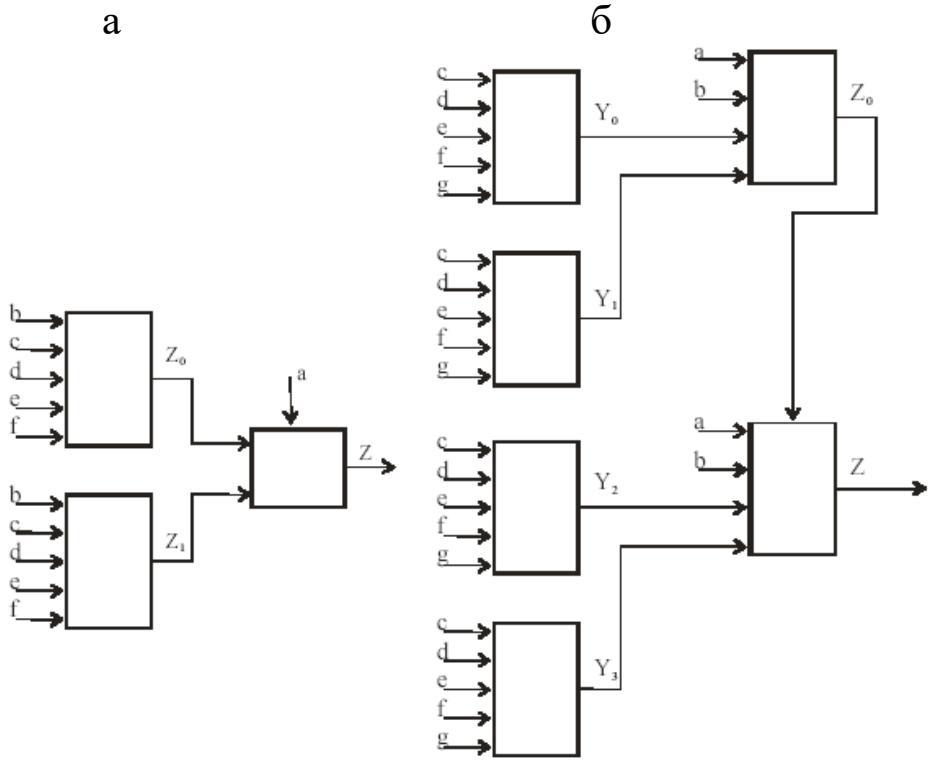


Рис. 8.22. Реалізація функцій шести і семи змінних:
а – загальна функція шести змінних;
б – загальна функція семи змінних

Наприклад, розглянемо таку функцію:

```
Z = abcd'ef'+ a'b'c'def' + b'cde'f .
```

Установка a = 0 дає Z_0 = 0. bcd'ef' + 1. b'c'def' + b'cde'f = = b'c'def + b'cde'f, а установке a = 1 відповідає Z_1 = 1.bcd'ef' + + 0.b'c'def' + b'cde'f = bcd'ef' + b'cde'f.

Оскільки Z_0 і Z_1 є функціями п’яти змінних, кожна з них може бути реалізована на одному осередку.

Будь-яка функція семи змінних може бути реалізована за допомогою менш логічних осередків. Розкладання для загальної функції семи змінних має вигляд

$$Z(a,b,c,d,e,f,g) = a'b'Z(0,0,c,d,e,f,g) + a'bZ(0,1,c,d,e,f,g) + ab'Z(l,0,c,d,e,f,g) + abZ(l,l,c,d,e,f,g) = a'b'Y0+a'bY1+ab'Y2+abY3. \quad (8.2)$$

Рівняння (8.2) може бути отримано застосуванням теореми розкладання двічі: перше розкладання за змінною a, потім розкладання за змінною b. Наприклад, розглянемо функцію семи змінних:

```
Z=c'de'fg+bcd'e'fg'+a'c'def'g+a'b'd'ef'g'+
                + ab'defg'.
```

Підставляючи a = b = 0, отримуємо Y0 = c'de'fg + c'def'g + + d'ef'g'; підставляючи a = 0, b = 1, отримуємо Y1 = c'de'fg + + cd'e'fg' + c'def'g; підставляючи a = 1, b = 0, отримуємо Y2 = = c'de'fg + defg'; підставляючи a = b = 1, отримуємо Y3 = c'de'fg + + cd'e'fg'.

Схема на рис. 3.22, б відповідає рівнянню (8.2). Чотири осередки реалізують функції змінних Y0, Y1, Y2 і Y3. П'ятий осередок реалізує функцію чотирьох змінних, Z0 = a'b'Y0 + a'b'Y1, м, Z = Z0 + ab'Y2 + abY3. При збільшенні кількості змінних n швидко зростає максимальна кількість логічних осередків, необхідних для реалізації функції n змінних. З цієї причини при великій кількості вхідних змінних n використання CPLD може бути кращим рішенням, ніж FPGA.

Xilinx FPGA, серії 4000 схожі з пристроями серії 3000, але мають більшу кількість входів і виходів і багато інших додаткових властивостей. Рис. 3.23 ілюструє спрощену блок-схему конфігурованого логічного блока XC4000.

Він має дев'ять логічних входів (*Fl, F2, F3, F4, Gl, G2, G3, G4 і H1*). Таким чином, можна згенерувати дві незалежні функції чотирьох змінних:

G(G1,G2,G3,G4) і *F(F1,F2,F3,F4).*

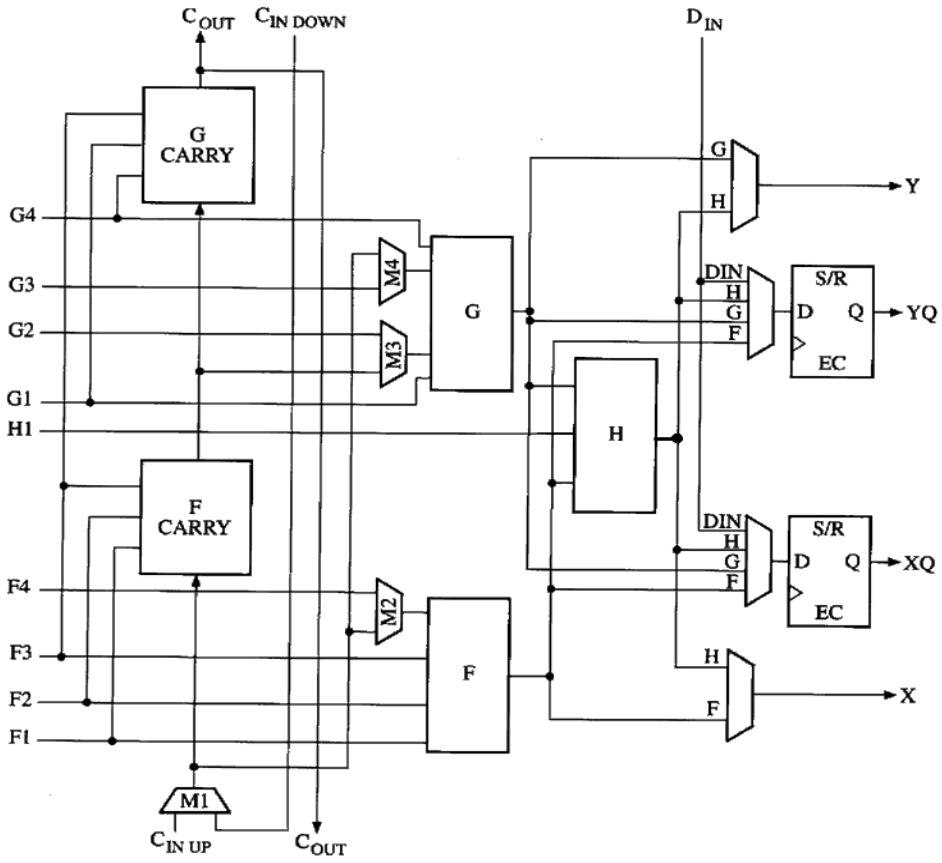


Рис. 8.23. Додаткова логічна схема перенесення XC4000

На відміну від серії 3000, де входи генеруються, функції чотирьох змінних частково перекриваються. CLB серії 4000 може також генерувати функцію H, яка визначається F, G і H1. Встановивши Fl = Gl, F2 = G2, F3 = G3 і F4 = G4, можна згенерувати будь-яку функцію п’яти змінних за формулою *H = F(F1,F2,F3,F4) H1’ + G(F1,F2,F3,F4) H1*. Можна також реалізовувати деякі функції 6, 7, 8, і 9 змінних.

CLB має два D тригери з входом дозволу синхронізації (EC). CLB має також чотири виходи, два від тригерів і два від реалізованих функцій комбінаційної логіки. На відміну від CLB серії 3000, CLB 4000 не має внутрішнього зворотного зв'язку, таким чином, коли необхідний зворотний зв'язок, виходи з тригерів мають бути спрямовані назад до логічних входів зовнішнім чином. CLB має S/R (set/reset - установлення/скидання), вхід яких може бути незалежно налаштований для з'єднання SD або RD входом кожного тригера. Таким чином, один тригер може бути встановлений в 1, а інший в 0 одним і тим самим S/R сигналом. Синхронізуючий вхід кожного тригера може бути налаштований

на спрацювання за переднім або заднім фронтами K (clock) входу. EC вхід кожного тригера може або завжди дозволяти перемикання, або за сигналом дозволу зі входу EC логічного блока. D вхід кожного тригера може бути з'єднаний *DIN, F, G* або *H.*

Логічний осередок серії 4000 містить спеціальну логічну схему перенесення (dedicated carry logic), як показано на рис. 8.24. Кожна клітинка містить логічну схему перенесення двох бітів, F і G функціональні перетворювачі можуть бути використані для генерування бітів суми. Таким чином, кожна клітинка може бути запрограмована для одночасного обчислення суми і перенесення двох бітів повного суматора. Лінії перенесення між осередками є фіксованими (непрограмованими), щоб забезпечити швидку передачу перенесення. Перенесення можуть передаватися вгору або вниз між осередками і на програмований мультиплексор (мітка M1 на рис. 8.24), який вибирає напрямок передачі сигналу.

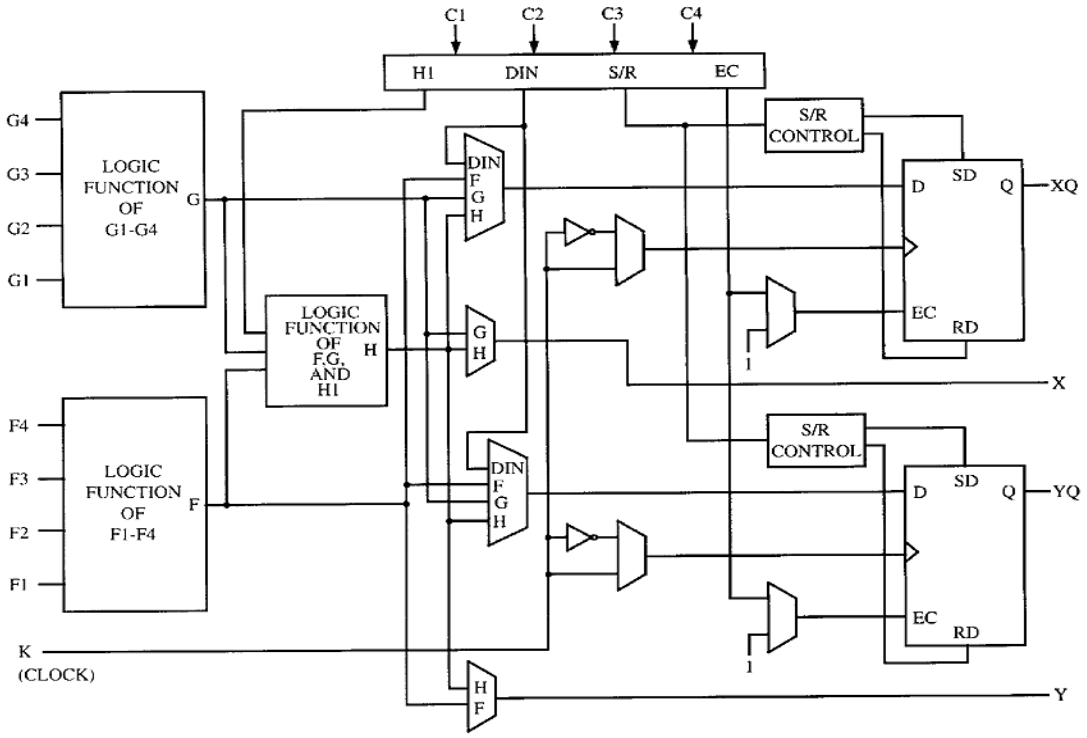


Рис. 8.24. Спрощена блок-схема для CLB серії 4000

Мультиплексори M2, M3 і M4 дозволяють деяким входам F і G генераторів функцій отримувати значення з перенесень замість нормальних F і G входів осередків. До того ж до суматора схема

перенесення може бути запрограмована на реалізацію пристроїв віднімання, додавання/віднімання, збільшення/зменшення (incrementer/decrementer), довічного доповнення (2's complementer) і лічильників, коли осередок програмується на реалізацію двох бітів повного суматора. Логічна схема еквівалентна схемі, поданій на рис. 8.25. У такій конфігурації Ai і B йдуть з входів осередка Fl і F2. Ai + 1 і Bi + 1 йдуть з входів осередка Gl і G4. Вхід Ci і вихід Ci + 2 з'єднується з лініями передачі фіксованого перенесення. Рис. 3.26 показує зв'язок для чотирибітного суматора.

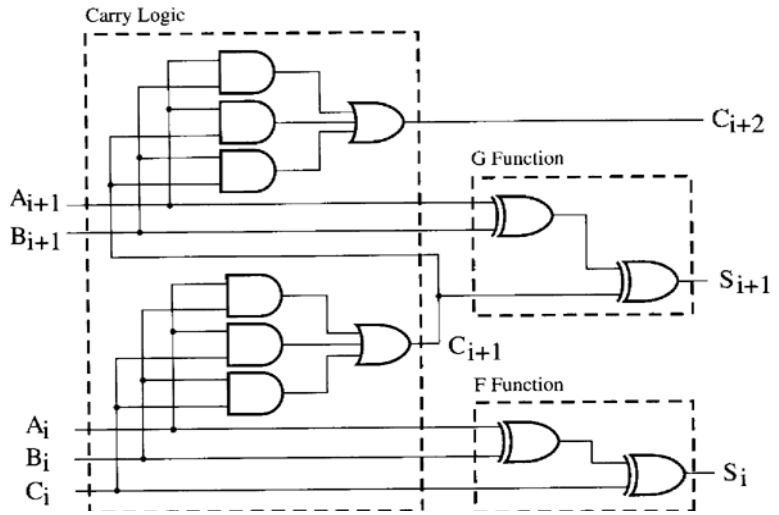


Рис. 8.25. Схема концептуального подання типового суматора (два біти на клітинку)

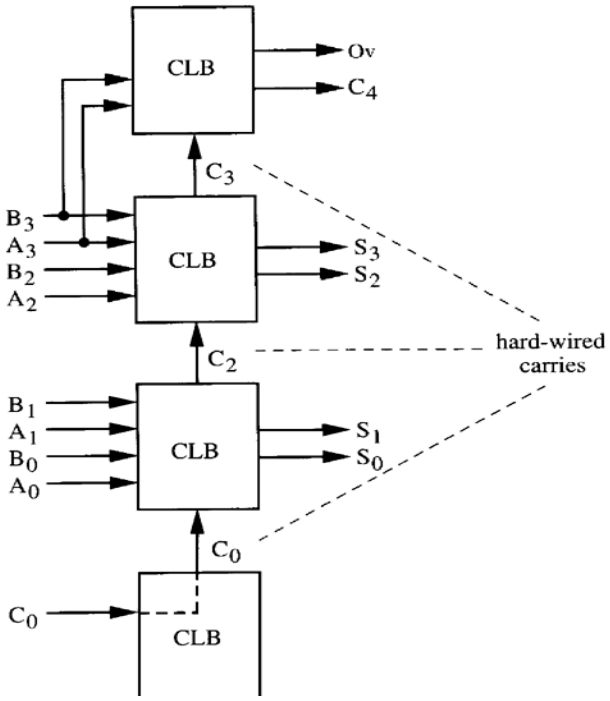


Рис. 8.26. З'єднання для чотирибітного суматора

Два середніх CLB виконують додавання чотирьох бітів. Якщо перенесення в останній значущий розряд необхідний, C0 має бути спрямований через додатковий осередок, тому що не дозволено пряме з'єднання фіксованої лінії перенесення. Якщо необхідна ознака переповнення і зовнішнє перенесення з наймолодшого розряду, потрібен четвертий осередок. У цьому прикладі C3 (замість C4) направляється в четвертий осередок, і входи в осередку A3 і B3 дублюються. C4 обчислюється за допомогою F функціонального генератора, що забезпечить зовнішнє перенесення з осередка. C4 також перераховується з використанням логічної схеми перенесення в клітинку, і потім переповнення обчислюється G функціональним генератором. Цей чотирибітний суматор може бути легко розширений до 8- або 16-бітного, додаванням двох або шести додаткових осередків. Такі модулі суматора є в бібліотеці Xilinx.

Коли осередок програмується для реалізації двох бітів пристроїв додавання/віднімання, сигнал має бути з'єднаний з F3 і G3. Коли = 0, входи Bi і Bi + 1 інвертуються всередині логічної схеми перенесення, і функціональні генератори, таким чином, виконують віднімання додаванням довічного доповнення. *F i G* функціональні генератори можуть бути запрограмовані для роботи як RAM пам'яті (рис. 3.27). Кожна клітинка може бути налаштована як 16 немов двобітна RAM. Входи F і G функціональних генераторів *(F1 = G1, F2 = G2, F3 = G3* і *F4 = G4)* забезпечують чотирибітну адресацію, C1 використовується як лінія дозволу запису *(WE), C2* та *C3* служать для введення даних *(Dl* і *DO).* Вміст адресованих комірок пам'яті доступно по виходах F і G функціональних генераторів. Крім того, осередок може бути налаштований, як 32x1 біт RAM, з C2 використовуваним як п'ятий адресний біт і C3 – як вхід даних. Конфігуровані біти *WRITE G i WRITE F* мають бути встановлені на дозвіл запису в *F i G* функціональні генератори відповідно. На рис. 3.28 показано вхідний/вихідний (I/O) блок серії 4000, який схожий за можливостями з входом/виходом блока серії 3000.

Мітки M на діаграмі являють собою конфігуровані біти пам'яті. Як і IOB 3000, IOB 4000 може бути запрограмований інвертувати входи, виходи, тристабільний контрольний буфер і вхід C – синхронізація.

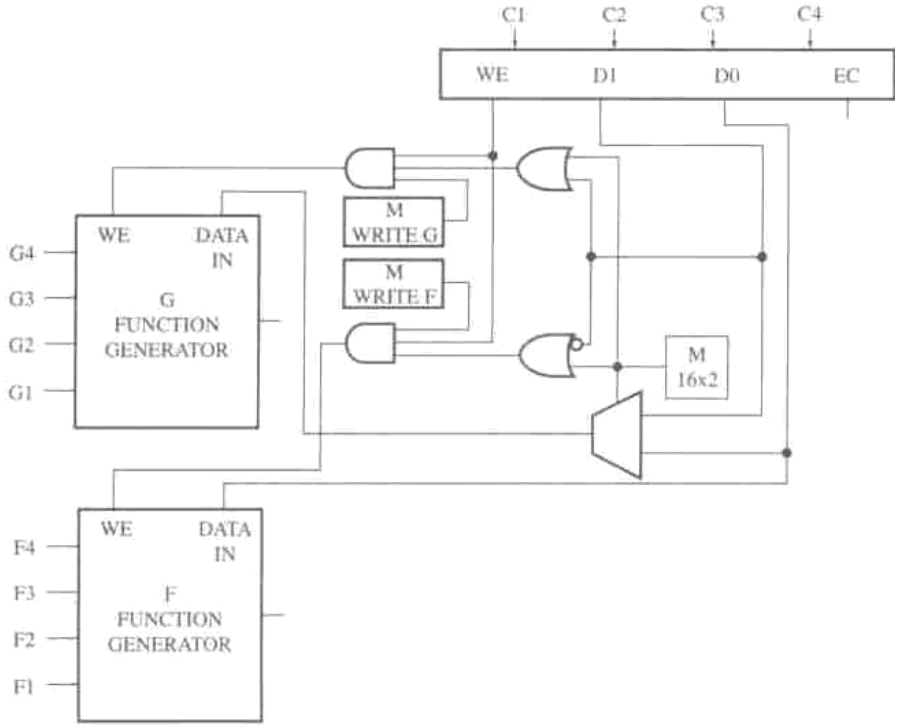


Рис. 8.27. CLB як читаючий/записуючий осередок пам'яті

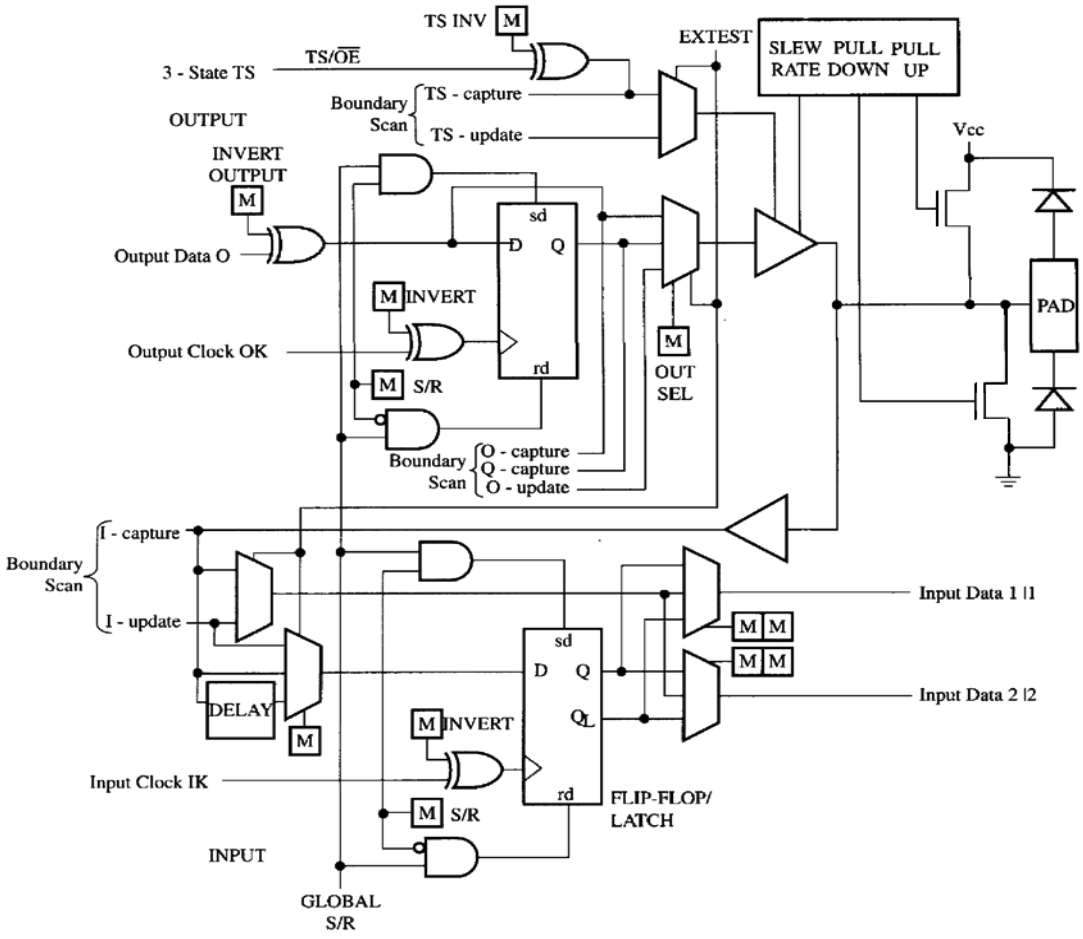


Рис. 8.28. Вхідний/вихідний блок серії 4000

На додачу, загальна S/R (GLOBAL S/R) лінія може бути встановлена на установлення або скидання кожного тригера. Вхідний/вихідний блок має обидва необов'язкових pull-up і pull-down (навантажувальний резистор; резистор установлення робочої точки, узгоджувальний резистор; резистор витоку) резистори, пов'язаних з вхідним/вихідним контактом (I / O pad), також містить додаткову логіку периферійного скануючого тестування, яке буде обговорюватися в розд. 10.

XC4003 FPGA, який ми щойно описали, має 100 CLB (10 x 10), близько 3000 вентилів, 80 призначених для користувача I/O, 360 тригерів (200 в CLB і 160 в lOB), 45636 конфігуруючих бітів даних. Інші представники сімейства XC4000 мають до 2304 CLB (48 x 48), 384 призначених для користувача I/O, 5376 тригерів і більше одного мільйона конфігурованих бітів даних.

Подальший розвиток FPGA XC4000X призвів до створення нової серії мікросхем Virtex, а також спрощеної версії Virtex - Spartan. Серія Virtex FPGA дає високоінтегровані і багатофункціональні мікросхеми. Потужність і гнучкість цих мікросхем робить їх альтернативою масковим матрицям логіки. Серія Virtex налічує 9 типів мікросхем, деякі з них наведені в табл. 8.3. Їхні основні характеристики: високий ступінь інтеграції; щільність від 50 кбайт до 1 Мбайт системних вентилів; частотні можливості до 200 МГц; сумісність з 66 МГц PCI. Мікросхеми серії Virtex-II містять до 8 Мбайт системних вентилів.

Таблиця 8.3

Мікросхеми серії Virtex

Device	System Gates	CLB Array	Logic Cells	Maximum Avaible I/O	BlockRAM Bits	Maximum SelectRAM+™Bits
XCV50	57 906	16x24	1 728	180	32 768	24 576
XCV100	108 904	20 x 30	2 700	180	40 960	38 400
XCV800	888 439	56 x 84	21 168	512	114 688	301 056
XCV1000	1 124 022	64 x 96	27 648	512	131 072	393 216

Загальна схема чіпа Virtex FPGA зображена на рис. 8.29, складається з двох основних елементів: конфігурованих логічних блоків (configurable logic block, CLB) і вхідних/вихідних блоків (input/output block, LOB). CLB забезпечують функціональні

елементи для конструювання пристроїв. IOB надають інтерфейс між контактами мікросхеми та CLB.

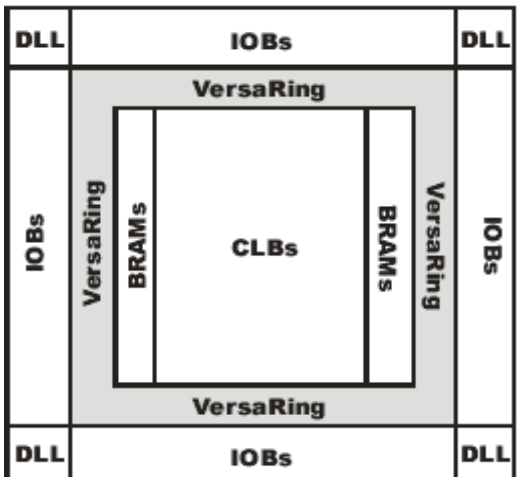


Рис. 8.29. Загальна схема Virtex-архітектури

Зв'язок між CLB виконується за допомогою загальних матриць маршрутизації (general routing matrix, GRM). GRM являють собою матриці перемикачів, розташованих на перетинах вертикальних і горизонтальних каналів трасування. Кожен CLB у свою чергу вмонтований у VersaBlock, що дозволяє виконувати місцеву маршрутизацію для зв'язку CLB з GRM. VersaRing вхідний/вихідний інтерфейс надає додаткові ресурси для маршрутизації по периметру мікросхеми, які покращують можливості маршрутизації входів/виходів і полегшують зв'язок із зовнішніми контактами.

Virtex-архітектура включає також такі елементи, доступ до яких можна здійснювати через матриці маршрутизації (GRM): додаткові блоки пам'яті по 4096 бітів в кожному; синхронізацію DLL для компенсації затримки поширення синхросигналу і управління синхронізацією або тривалістю синхроімпульсу; тристабільні буфери, пов'язані з кожним CLB, які управляють виділеними сегментованими горизонтальними ресурсами трасування.

Дані, що зберігаються в статичних елементах пам'яті, управляють конфігурованими логічними елементами і ресурсами трасування. Вони заносяться в осередок пам'яті при подачі напруги і можуть бути вивантажені в разі потреби зміни функції пристрою.

Конфігурований логічний блок (Configurable Logic Block, CLB) є основним елементом у Virtex, LC є логічним осередком (logic cell, LC). Він складається з генератора функції чотирьох змінних, схем перенесення (carry logic) і логічного елемента. Вихід генератора функцій може бути пов'язаний з будь-яким з виходів CLB або входом D-тригера. Кожен Virtex CLB містить чотири LC, організованих у вигляді двох однакових секторів, як це показано на рис. 8.30. На рис. 8.31 подана більш детальна схема одного сектора.

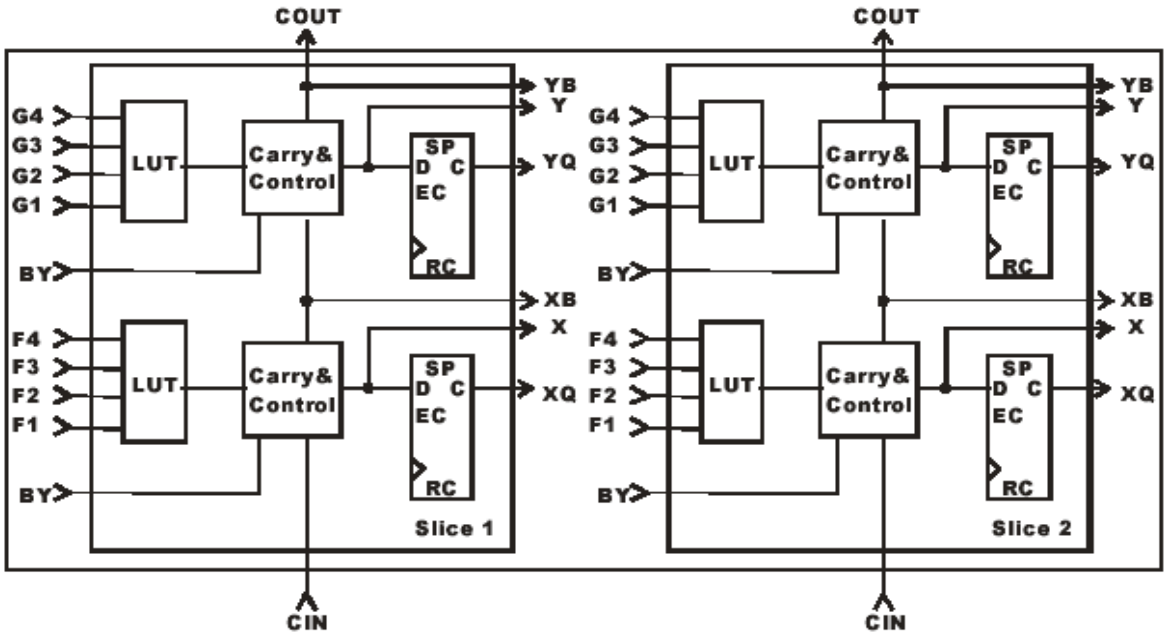


Рис. 8.30. Конфігурований логічний блок мікросхеми Virtex

На додачу до чотирьох основних LC Virtex CLB містить логіку, яка об'єднує функціональні генератори для реалізації функцій п'яти або шести змінних. Тому коли оцінюють кількість вентилів, що надаються даним пристроєм, кожен CLB оцінюється як 4,5 LC.

Look-Up Table – LUT. Virtex – функціональний генератор, організований як чотиривходовий (look-up table) LUT. На додачу до генерації функції кожен LUT може функціонувати як 16x1 біт синхронне ОЗП (RAM). Більш того, два LUT з одного сектора можуть бути об'єднані для створення 16x2 або 32x1 біт синхронних ОЗП, або 16x1 біт двоканальних синхронних ОЗП. Virtex LUT може також працювати як 16-бітовий зсувний регістр, ідеальний для високошвидкісних або пакетних даних.

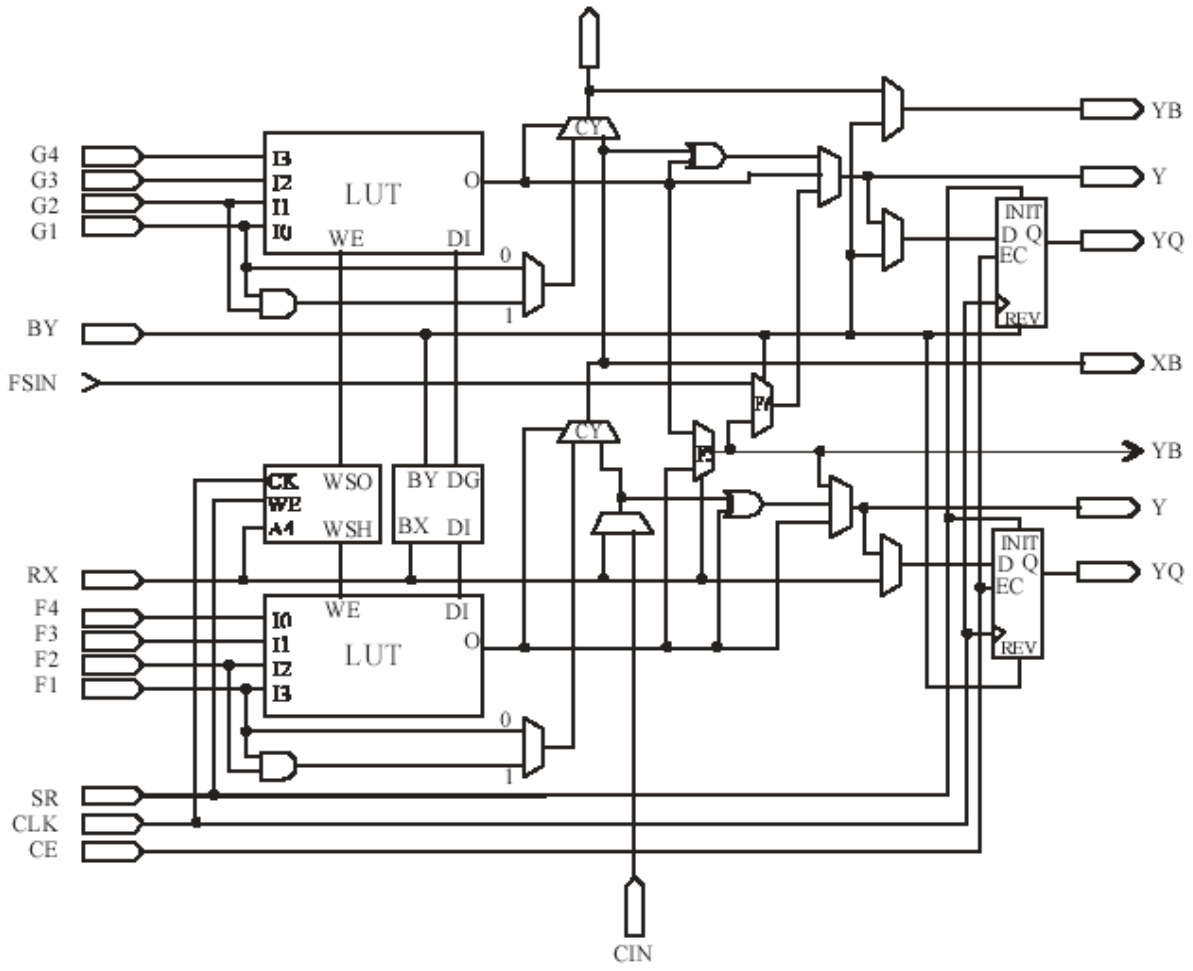


Рис. 8.31. Детальна схема сектора Virtex FPGA

Тригери в секторі CLB можуть бути налаштовані як D-тригери з синхронізацією за фронтом або рівнем. Значення на вхід тригера подаються з виходу функціонального генератора або безпосередньо зі входів сектора в обхід функціональних генераторів.

Кожен тригер на додачу до сигналів синхронізації і дозволу синхронізації має синхронне скидання SR і установлення BY. SR встановлює елемент пам'яті в початковий стан, як воно описано для нього в конфігурації. BY встановлює елемент пам'яті в протилежний стан. Дозволяється налаштування цих сигналів для роботи в асинхронному режимі.

Всі керуючі сигнали можуть бути незалежно проінвертовані і є загальними для обох тригерів сектора.

Додаткова логіка (Additional Logic). Мультиплексор F5 в кожному секторі об'єднує виходи функціональних генераторів. Таким чином, реалізується одна функція п’яти змінних,

мультиплексор 4×1 або деякі окремі випадки функцій до дев’яти змінних.

Аналогічно мультиплексор F6 об'єднує виходи всіх чотирьох функціональних генераторів у CLB, вибираючи один з виходів мультиплексорів F5. Це дозволяє реалізувати будь-яку функцію шести змінних, мультиплексор 8×1 або деякі функції до 19 змінних.

Кожен CLB має чотири безпосередніх наскрізних шляхи, по одному на LC. Вони надають додаткові вхідні лінії для даних або маршрутизації додаткової логіки, дозволяючи не витрачати ресурси схеми.

Арифметична логіка (Arithmetic Logic). Спеціальна схема перенесення виконує швидке обчислення перенесення, даючи можливість реалізовувати високошвидкісні схеми перенесення. Virtex CLB підтримує два різні ланцюги перенесення, по одному на кожен сектор. Висота ланцюга два біта на CLB.

Арифметична логіка містить один елемент XOR, дозволяючи реалізовувати один біт повного суматора на схемі. До того ж виділений I елемент дозволяє здійснити ефективну реалізацію множення. Схема перенесення може використовуватися і для каскадування функціональних генераторів при реалізації функцій великої кількості змінних.

Кожен Virtex CLB містить два тристабільних драйвери (BUFTs), які можуть управляти шинами в мікросхемі. Кожен Virtex BUFT має незалежний тристабільний керуючий контакт і незалежний вхідний контакт.

Блоки ОЗП (Block RAM). Virtex FPGA містять кілька великих виділених блоків ОЗП BlockSelectRAM+, на додачу до SelectRAM+ LUTRAM, роззосереджених по CLB, і формують тіньову ОЗП-структуру, реалізовану в конфігурованих логічних блоках.

Блоки пам'яті BlockSelectRAM+ організовані в стовпці. Всі пристрої Virtex містять два таких стовпчики, що поширюються на всю висоту чіпа, по одному уздовж кожної з вертикальних сторін. Кожен блок пам'яті має висоту, рівну 4 CLB. Відповідно Virtex-пристрій висотою в 64 CLB містить 16 блоків пам'яті в стовпці і 32 блоки пам'яті в мікросхемі. У табл. 8.4 подано кількість блоків пам'яті Block SelectRAM+ в деяких мікросхемах Virtex.

Таблиця 8.4

Кількість Block SelectRAM+ в мікросхемах серії Virtex

Device	# of Blocks	BlockRAM Bits
XCV50	8	32 768
XCV100	10	40 960
XCV800	28	114 688
XCV1000	32	131 072

Кожний осередок Block SelectRAM+, як показано на рис. 8.32, є повністю синхронізованим двопортовим 4096-бітовим блоком ОЗП з незалежним управлінням сигналами для кожного порту. Ширина даних може бути незалежно налаштована для кожного порту, що забезпечується вбудованим перетворювачем ширини.

У табл. 8.5 описано взаємозв'язок між шириною шини та іншими параметрами Block SelectRAM+. Блоки ОЗП в Virtex включають також додаткове трасування, що забезпечує ефективний інтерфейс між блоками CLB та іншими модулями ОЗП.

Вхідний-вихідний блок (IOB) Virtex, зображений на рис. 8.33, дає можливість конфігурації входів і виходів, підтримує велику кількість вхідних і вихідних стандартів (табл. 8.6). Ці високошвидкісні входи і виходи підтримують PCI інтерфейс до 66 МГц.

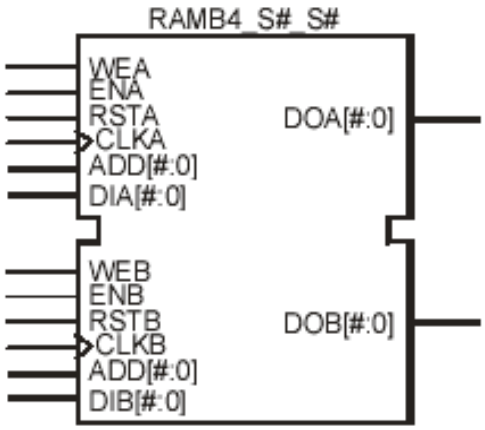


Рис. 8.32. Двопортовий Block SelectRam+

Таблиця 8.5

Залежність параметрів ОЗУ Block SelectRAM+
від ширини шини

Width	Depth	ADDR Bus	Data Bus
1	4096	ADDR<11:0>	DATA<0>
2	2048	ADDR<10:0>	DATA<1:0>
4	1024	ADDR<9:0>	DATA<3:0>
8	512	ADDR<8:0>	DATA<7:0>
16	256	ADDR<7:0>	DATA<15:0>

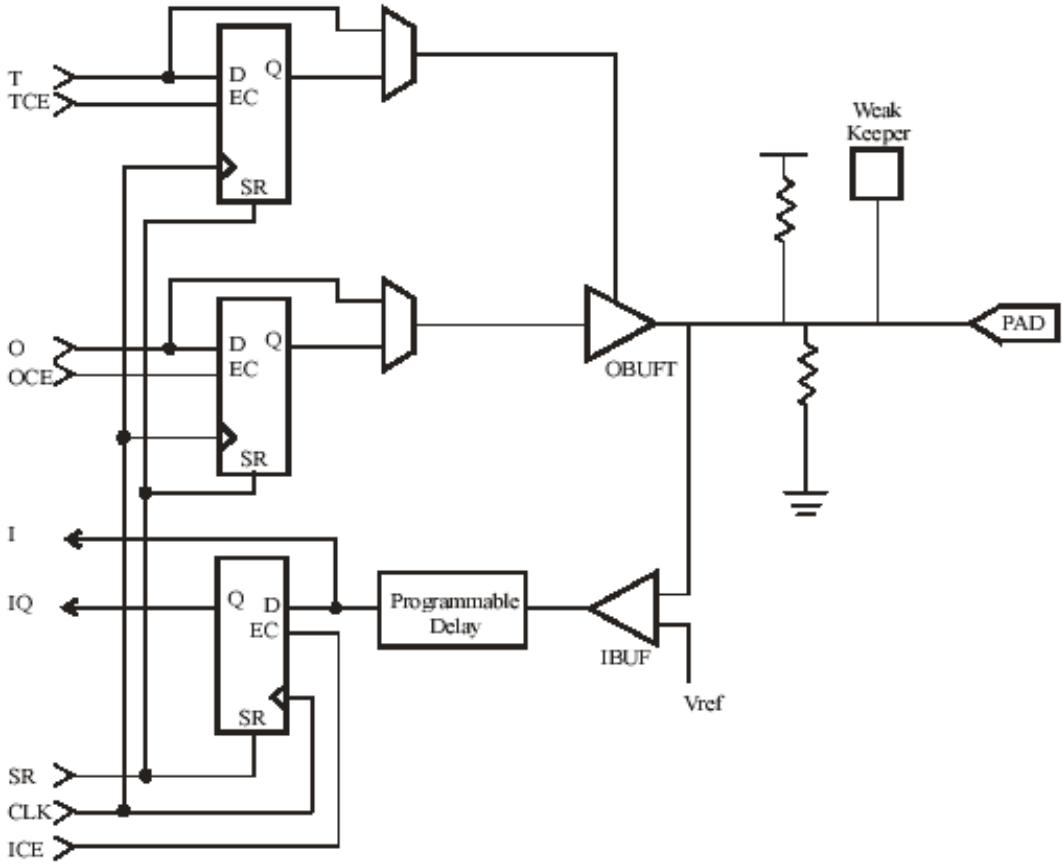


Рис. 8.33. Вхідний/вихідний блок Virtex FPGA

Три елементи пам'яті в IOB можуть функціонувати як тригери з синхронізацією за фронтом або рівнем. Кожен блок має сигнал синхронізації, загальний для всіх тригерів, і незалежні сигнали дозволу синхронізації для кожного тригера. До того ж до керуючих сигналів CLK і CE тригери мають загальний сигнал скидання/установлення SR. Для кожного тригера цей сигнал може бути незалежно налаштований як синхронне або асин-

хронне скидання чи установлення. Вхідні і вихідні буфери і всі IOB керуючі сигнали мають незалежний контроль за напрямком.

Таблиця 8.6

Підтримувані вхідні/вихідні стандарти

I/O Standart	Input Reference Voltage (Vref)	Output Source Voltage (Vcco)	Board Termination Voltage (Vtt)
LVTTL 2-24 mA	N/A	3.3	N/A
LVCMOS2	N/A	2.5	N/A
PCI	N/A	3.3	N/A
GTL	0.8	N/A	1.2
GTL+	1.0	N/A	1.5
HSTL Class I	0.75	1.5	1.5
HSTL Class III	0.75	1.5	1.5
HSTL Class IV	0.75	1.5	1.5
SSTL3 Class I and II	1.5	3.3	1.5
SSTL2 Class I and II	1.25	2.5	1.25
CTT	1.5	3.3	1.5
AGP	1.32	3.3	N/A

Усі контакти мають захист мікросхеми від пошкоджень через стрибки напруги. Пропонуються дві форми для захисту від високої напруги. Одна надає погодження з 5 В (compliance), інша ні. У першому випадку подібно до тунельного пробою виконується з'єднання з землею, коли напруга стає вище, приблизно 6,5 В. Коли нема необхідності в узгодженні з 5 В, до виходу підключається звичайний обмежувальний діод, що підтримує напругу VCCO. Тип захисту може бути обраний незалежно для кожного контакту.

Кожен контакт має підвищувальний (pull-up) і знижувальний (pull-down) резистори і weak-keeper схему. До програмування мікросхеми всі її виходи вважаються конфігурованими і примусово встановлюються в стан високого імпедансу. Знижувальні резистори і weak-keeper схеми не активні, але на входи може бути поданий високий рівень напруги.

До конфігурації активізація підвищувальний резистор контакту управляється глобальною логікою. Якщо вони не активні, напруга на контактах плаває. Тому слід використовувати

зовнішні підвищувальні або знижувальні резистори для контактів, що вимагають наявності на них до конфігурації певного логічного рівня напруги. Усі Virtex IOB підтримують IEEE 1149.1 стандарт граничного сканування (boundary scan).

Якщо блок Virtex IOB використовується як вхідний, сигнал з зовнішнього контакту через буфер IBUF передається прямо або через тригер на внутрішню логіку. Необов'язковий програмований елемент затримки на вході D-тригера дозволяє компенсувати затримку на лінії між контактами (pad-to-pad hold time). Величина затримки узгоджується з затримкою внутрішньої синхронізації FPGA, і в цьому випадку вважається, що згаданий інтервал (pad-to-pad hold time) дорівнює 0.

Вихідний шлях містить тристабільний буфер, який управляє надходженням сигналу на вихід. Сигнал може подаватися безпосередньо на зовнішній контакт або через IOB вихідний тригер. Тристабільний вихідний буфер може управлятися безпосередньо з внутрішньої логіки мікросхеми або через тригер, який дозволяє синхронізувати сигнал заборони і дозволу.

Кожен контакт може бути запрограмований на роботу в режимі будь-якого з підтримуваних електричних стандартів. Використанням деяких з них визначається користувачем порогова напруга VREF. Допускається конфігурація входів необов'язковими підвищувальними і знижувальними резисторами 50-150 кОм. Кожен вихід може постачати до 24 мА і приймати до 48 мА. Управління потужністю і швидкістю наростання вихідної напруги дозволяє зменшити перехідні процеси на шині. У багатьох стандартах вихідна напруга високого рівня залежить від подаваної ззовні напруги Vcco.

Необов'язкова схема weak-keeper під'єднується до кожного виходу, стежить і м'яко управляє максимальною (High) або мінімальною (Low) напругою вихідного сигналу, порівнюючи її з вхідною. Якщо контакт приєднаний до сигналу з безліччю джерел, схема буде зберігати останнє значення сигналу, коли виходи всіх його джерел знаходяться в стані високого імпедансу. Це дозволяє уникнути виникнення брязкоту на шині. Схема weak-keeper використовується у вхідних буферах (IOB) для спостереження за вхідним рівнем і забезпечення відповідної напруги V_{REF}, якщо цього вимагає стандарт сигналу V_{REF}.

Банки входів/виходів (I/O Bank). Деякі з вхідних/вихідних стандартів вимагають використання напруг V_{CCO} і V_{REF}, які подаються ззовні і приєднуються до контактів, обслуговуючи групи IOB блоків. Такі групи називаються банками. Існують обмеження на комбінацію вхідних/вихідних стандартів у кожному банку.

Вісім вхідних/вихідних банків є результатом ділення кожного боку FPGA на два банки, як це показано на рис. 8.34. Кожен з них має кілька Vcco контактів. Всі вони мають бути підключені до одного й того самого джерела напруги, яке визначається використовуваним вихідним стандартом.

Всередині одного банку різні стандарти можуть бути використані, якщо вони мають однакову V_{cco} напругу. У табл. 8.7 описана можливість суміщення різних стандартів. GTL і GTL+ можуть застосовуватися з будь-якою з трьох напруг, оскільки їхні виходи з відкритим стоком (open-drain outputs) не залежать від V_{cco}.

Деякі вхідні стандарти вимагають певної користувачем порогової напруги V_{ref}. У цьому випадку окремі, призначені для користувача, вхідні/вихідні контакти автоматично конфігуруються як входи для Vref напруги. Для цього може бути використаний приблизно кожен шостий контакт у банку.

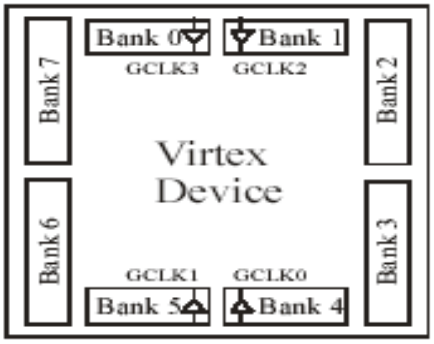


Рис. 8.34. Virtex I/O банки

Таблиця 8.7

Сумісність вихідних стандартів

Vcco	Совместимые стандарты
3.3 V	PCI, LVTTL, SSTL3 I, SSTL3 II, CTT, AGP, GTL, GTL+
2.5 V	SSTL2 I, SSTL2 II, LVCMOS2, GTL, GTL+
1.5 V	HSTL I, HSTL III, HSTL IV, GTL, GTL +

Усі V_{ref} контакти мають внутрішнє з'єднання в межах одного банку, отже, в одному банку може бути використана тільки одна Vref напруга.

Програмовані матриці трасування (Programmable Routing Matrix). Затримка найдовшого шляху визначає швидкість будь-якого проєкту. Оптимізація, яка мінімізує затримки довгих шляхів, дозволить створити пристрій з кращими системними можливостями. Вона також скоротить час компіляції проєкту, оскільки архітектура стає сприятливою до програмного забезпечення, що відповідно зменшить і загальну тривалість циклів проєктування.

Місцеве трасування (Local Routing). VersaBlock забезпечує локальні ресурси трасування і, як показано на рис. 8.35, дає такі типи зв'язку:

– з’єднання між LUT, тригерами і GRM;

– зворотний зв'язок для CLB, що забезпечує високошвидкісне з'єднання LUT усередині одного CLB і об'єднує їх у ланцюг з мінімальною затримкою на лініях трасування;

– пряме з'єднання, яке забезпечує високошвидкісне з'єднання між сусідніми горизонтальними CLB, виключаючи затримку на GRM.

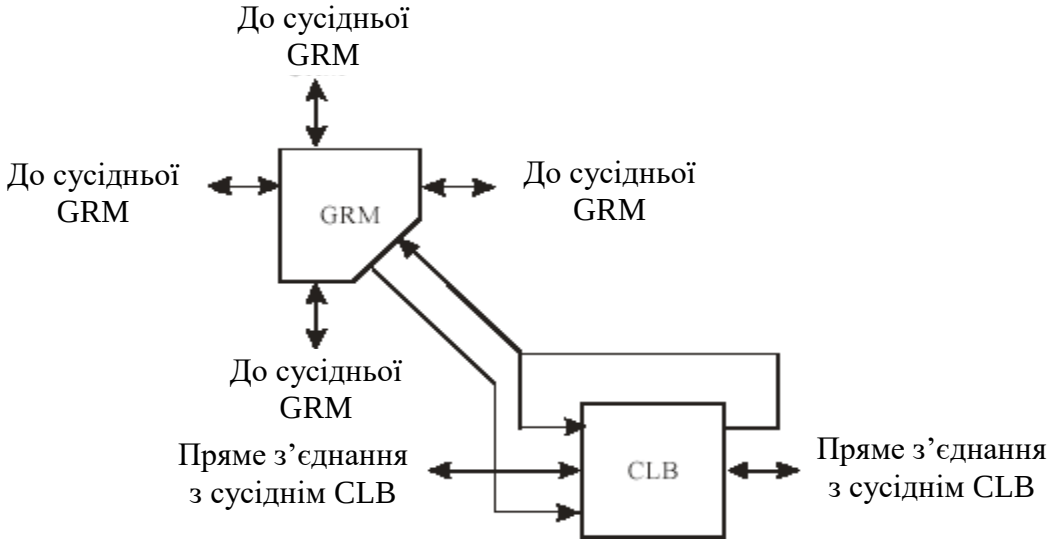


Рис. 8.35. Virtex локальна маршрутизація

Трасування загального призначення (General Purpose Routing). Більшість Virtex-сигналів прямують через трасування

загального призначення, відповідн, більшість з'єднань асоціюються з цим рівнем ієрархії трасування. Ресурси трасування загального призначення розташовуються в горизонтальних і вертикальних каналах, пов'язаних з рядками і стовпцями CLB. Ці ресурси перераховані нижче:

– сусідньою з кожним CLB є матриця загального трасування (General Routing Matrix, GRM). GRM – це матриця перемикачів, що з'єднує горизонтальні і вертикальні ресурси і дає для CLB доступ до ресурсів трасування загального призначення;

– 24 лінії трасування одиничної довжини передають сигнали до сусідніх GRM в кожному з чотирьох напрямків;

– 72 буферізовані Hex lines передають сигнали в кожному з чотирьох напрямків. Організовані в шаховому порядку Hex lines, можуть управлятися тільки в їхніх кінцевих точках. Сигнали з них доступні або в кінцевих, або в середніх точках (три блоки від джерела). Кожна третя лінія - двонаправлена, інші – однонаправлені;

– 12 довгих ліній (Longlines) є буферизованими, двонаправленими, передають сигнал через пристрій швидко і ефективно. Вони мають довжину всього чіпа, у висоту або ширину.

Вхідне/вихідне трасування (I/O Routing). Пристрої Virtex мають додаткові ресурси трасування на його межах, що формують інтерфейс між матрицею CLB і IOB-блоками. Таке додаткове трасування, VersaRing, полегшує pin-свопинг і pin-з'єднання, так що змінений проєкт може бути адаптований до існуючої PCB (Printed Circuit Board – друкована плата) планування. Це дозволяє скоротити час виходу на ринок, оскільки PCB та інші компоненти системи можуть вже випускатися, у той час як проєкт знаходиться ще на стадії розроблення.

Виділене трасування (Dedicated Routing). Деякі види сигналів вимагають виділених ресурсів трасування. У Virtex-архітектурі такі ресурси введені для двох класів сигналів:

– горизонтальні ресурси трасування є внутрішніми тристабільними шинами. Кожному ряду CLB надаються чотири шинні лінії, дозволяючи створювати складні шини, як це показано на рис. 3.36;

– дві додаткові лінії на CLB передають сигнали перенесення вертикально між сусідніми CLB.

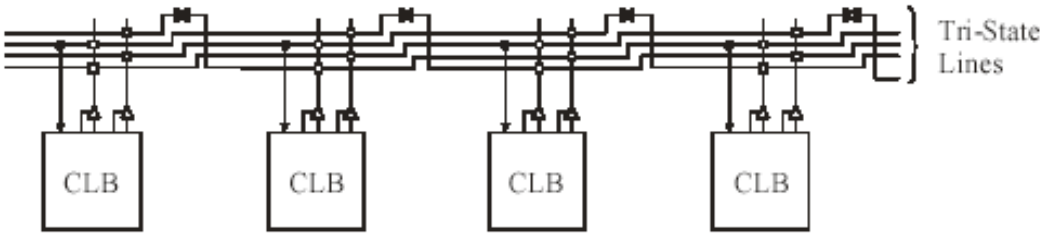


Рис. 8.36. Підключення BUFT до виділених горизонтальних ліній

Глобальне трасування (Global Routing). Глобальні ресурси трасування розподіляють синхросигнали та інші сигнали, що мають високе розгалуження за влаштуванням. Пристрої Virtex включають два типи глобальних ресурсів трасування: первинні і вторинні.

До первинних глобальних ресурсів маршрутизації відносять чотири виділені глобальні лінії з виділеними вхідними контактами, які спроєктовані для поширення з мінімальним розфазуванням високорозподілених синхросигналів. Кожна лінія глобальної синхронізації може управляти всіма синхровходами блоків CLB, IOB і RAM. Первинні глобальні лінії управляються тільки глобальними буферами. Існує чотири глобальних буфери, по одному на кожну лінію. Вторинні глобальні ресурси трасування складаються з 24 магістральних ліній (backbone lines), 12 уздовж верху мікросхеми і 12 внизу. Від цих ліній можуть бути поширені до 12 окремих сигналів у стовпці через 12 довгих ліній. Вторинні ресурси є більш гнучкими, ніж первинні, і їх використання не обмежується тільки сигналами синхронізації.

Поширення синхросигналів (Clock Distribution). Virtex забезпечує високошвидкісне, з низькою затримкою, поширення синхросигналів через первинні глобальні ресурси маршрутизації, описані вище. Типове поширення сигналів зображено на рис. 8.37.

Є чотири глобальних буфери: два вгорі і два внизу, по центру пристрою. Вони управляють чотирма первинними глобальними лініями, які у свою чергу управляють будь-якими синхровходами. Існують чотири виділених синхроконтакти, по одному на кожен глобальний буфер. Вхід глобального буфера вибирається тільки через ці контакти або сигнали трасування загального призначення.

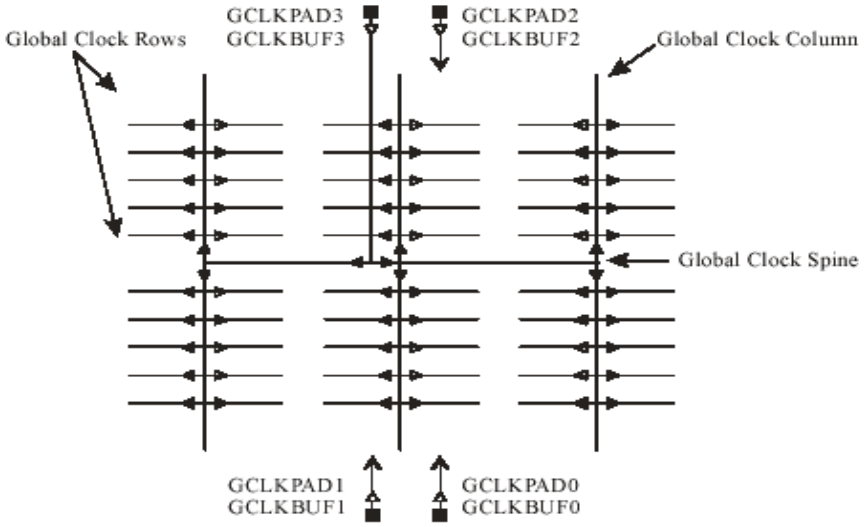


Рис.8.37. Схема розподілу глобальної синхронізації

Система автоматичного підстроювання за затримкою. Пов'язаний з кожним глобальним синхросигналом вхідний буфер є повністю цифровою системою автоматичного підстроювання за затримкою (Delay-Locked Loop DLL), яка може виключати зрушення сигналу між вхідним контактом синхросигналу і внутрішнім синхровходом. Кожна DLL може управляти двома глобальними синхролініями. DLL спостерігає вхідний і розповсюджуваний синхросигнали і автоматично регулює елемент, що затримує синхросигнал. Додаткова затримка вводиться таким чином, щоб фронт на внутрішні тригери поступав з затримкою на один період синхроімпульсу. Ця система має зворотний зв'язок і ефективно виключає затримку поширення сигналу, гарантуючи, що він буде поданий на внутрішні тригери синхронно з надходженням фронту на вхід.

На додачу до виключення затримки поширення сигналу DLL забезпечує додаткове управління тривалістю синхроімпульсу. DLL пропонує чотири квадратурні фази вихідного синхроімпульсу і може подвоювати його частоту або ділити її на 1.5, 2, 2.5, 3, 4, 5, 8, 16. Він має шість виходів.

DLL також функціонує як дзеркало для синхросигналу. Управляючи виходом з мікросхеми і отримуючи його назад, DLL може бути використана для усунення спотворень між кількома пристроями Virtex.

CPLD є розширенням концепції PAL. У загальному випадку CPLD є мікросхемою, яка містить певну кількість PAL-подібних

логічних блоків з програмованою матрицею міжз’єднань. Кожен PAL-подібний блок має програмовану AND матрицю, яка живить макрокомірки, і виходи цих макрокомірок можуть бути проведені до входів інших логічних блоків у тій самій мікросхемі. Більшість CPLD електрично стираються і репрограмуються і, таким чином, їх іноді відносять до EPLD (стираний програмований логічний пристрій). Altera MAX серії 7000 є сімейством високоефективних CMOS CPLD (комплементарний металооксидний напівпровідник). На противагу Xilinx FPGA, Altera серії 7000 використовує засновані на EEPROM конфігурованих осередках пам'яті, таким чином, одна запрограмована конфігурація буде збережена, поки її не зітруть.

На рис. 8.38 показана основна архітектура серії 7000, яка складається з деякої кількості блоків логічних матриць (Logic Array Blocks, LABs), вхідних/вихідних контрольних блоків (I/O Control Block) і програмованих матриць міжз’єднань (Programmable Interconnect Matrix, PIA). Кожен LAB містить 16 макрокомірок, кожна з яких містить комбінаційну логіку і тригер. Кожен LAB має 36 входів з PIA і 16 виходів у PIA. Від 8 до 16 виходів з кожного LAB можуть бути спрямовані до вхідних/вихідних контактів через вхідний/вихідний контрольний блок. Від 8 до 16 входів з вхідних/вихідних контактів можуть бути спрямовані до PIA через вхідний/вихідний контрольний блок. Вхід глобальної синхронізації (GCLK) і вхід загального скидання (GCLRn) пов'язують всі макрокомірки. Два виходи сигналів дозволу (OEIn and OE2n) з'єднують всі вхідні/вихідні контрольні блоки. Кожна макрокомірка на рис. 8.39 включає логічну матрицю, матрицю вибору кон'юнктивного терму, який живить OR вентиль, і програмований тригер.

Вертикальні лінії в логічних матрицях, які є загальними для всіх макрокомірок у LAB, управляються програмованими сигналами міжз’єднань від PIA і спільно використовуваних розширювачів логіки. Кон'юнктивні терми формуються в логічних матрицях так само, і в PAL. Кожну макрокомірку забезпечує п'ять кон'юнктивних термів, і вони розміщуються за матрицями вибору кон'юнктивних термів. Кон'юнктивні терми можуть бути використані як вхід вентиля OR, вхід вентиля XOR, розширювач логіки або попереднє встановлення тригера, очищення тригера, синхронізація чи вхід дозволу. Тригер у кожній макрокомірці є D тригером з дозволом синхронізації і асинхронним установленням

і скиданням. Вхід синхронізації може управлятися або загальним входом синхронізації, або кон'юнктивним термом. Вхід дозволу синхронізації може управлятися або кон'юнктивним термом, або Vcc (завжди дозволений). Вхід скидання – загальне скидання або кон'юнктивний терм. Вхід установлення управляється кон'юнктивним термом. Вхід D завжди пов'язаний з виходом вентиля XOR.

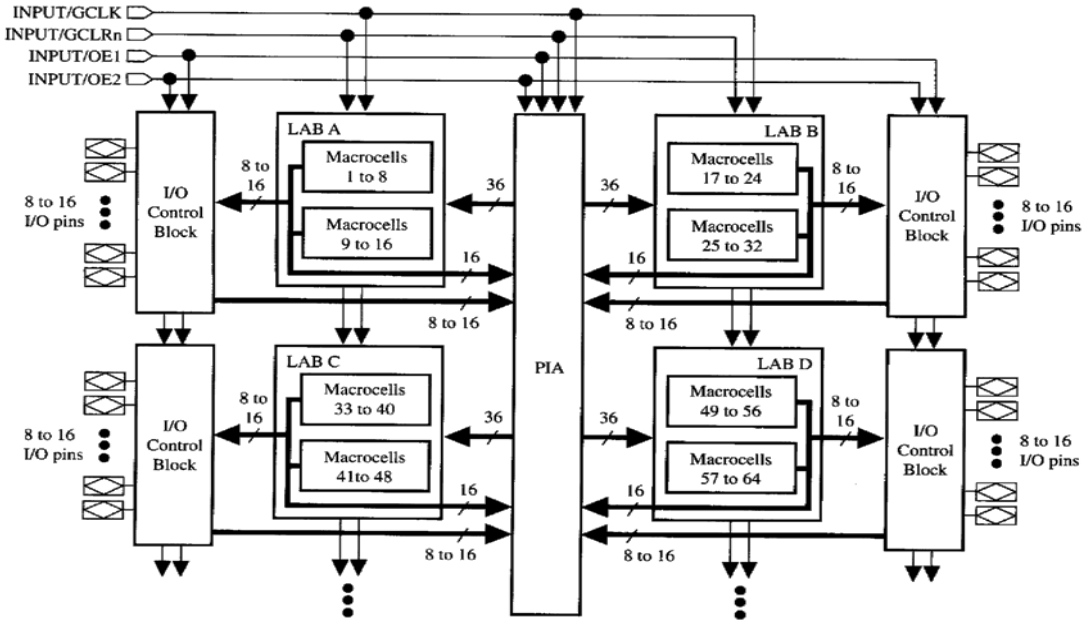


Рис. 8.38. Архітектура для пристроїв EPM Altera серії 7000

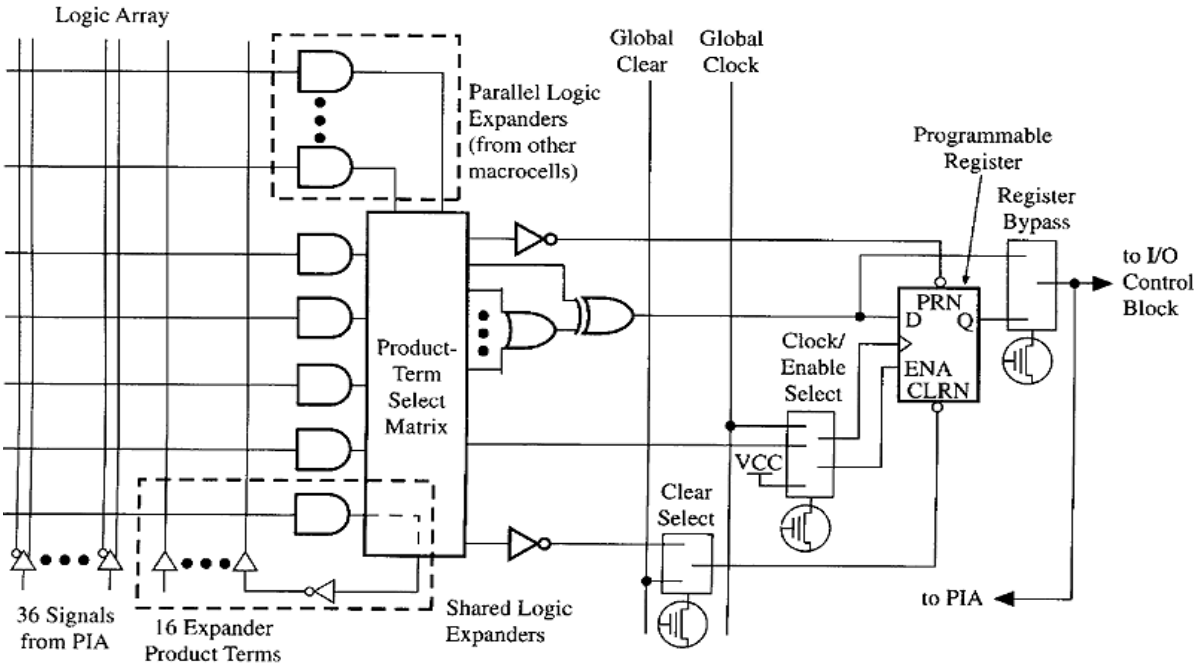


Рис. 8.39. Макрокомірки для пристроїв EPM

Мультиплексор Register Bypass (паралельний регістр) може вибирати або Q вихід тригера, або вихід з XOR. Обраний вихід може йти до PIA або на вхідний/вихідний контрольний блок. D тригер може бути перетворений у T-тригер за допомогою вентиля XOR. Оскільки характеристичне рівняння для T-тригера $Q^+=$, можна з'єднати один вхід XOR вентиля з Q і використовувати інший вхід для T. Використання T-тригера для реалізації лічильника або суматора часто вимагає меншої кількості вентилів, ніж використання D-тригера.

Хоча кожна клітинка має тільки п'ять кон'юнктивних термів, більш складні функції можуть бути реалізовані з використанням невикористовуваних кон'юнктивних термів з інших макрокомірок. Можливі два типи розширення кон'юктивних термів - загальний і паралельний розширювачі логіки. У кожному осередку один кон'юктивний терм може використовуватися як загальний розширювач (рис. 8.40).

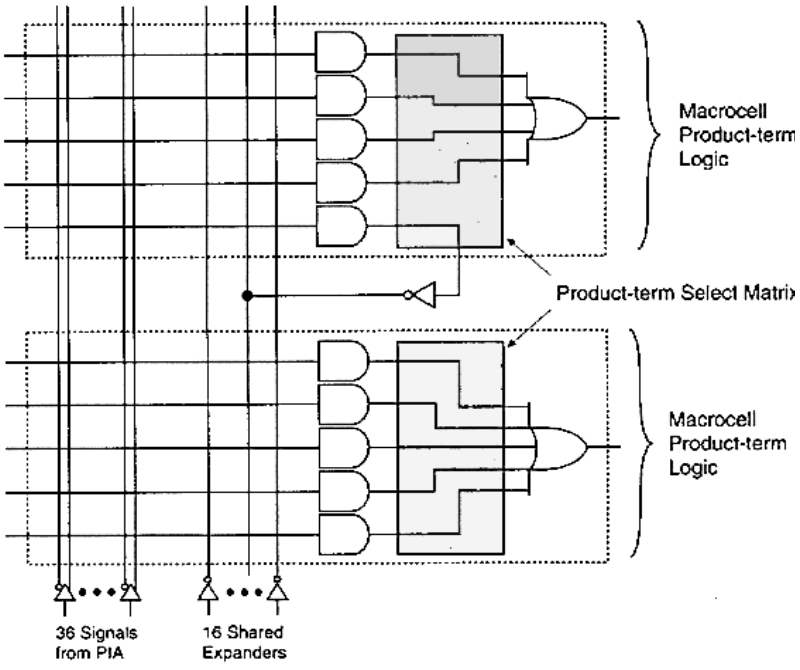


Рис. 8.40. Загальні розширювачі

Обраний кон'юктивний терм подається назад на логічну матрицю через інвертор, і інвертований кон'юнктивний терм може бути використаний як вхід будь-якого AND вентиля макрокомірки. Використання загального розширювача еквіва-лентно трирівневій схемі NAND-AND-OR. AND-OR логічний вираз з

більш ніж п'ятьма термами може часто бути розкладено для використання загального розширювача з інших макрокомірок. Наприклад, *P=AB+B'C+C'D+E'F+E'G+E'H+F'I+F'J=AB+B'C+ + C'D + E'(F'G'H')' + F'(I'J')'* може використовувати загальний розширювач для генерації *(F'G'H')'* і (*I'J'*)'. Вентиль XOR в осередку може використовуватися для доповнення функції, оскільки *F = F'*1. Іноді доповнення функції *(F')* вимагає менше термів, ніж оригінальна функція (F), у цьому випадку більш економічно реалізувати *F',* доповнити його, використовуючи вентиль XOR.

Паралельний розширювач (рис. 8.41) дозволяє незастосовуваним кон'юктивним термам бути використаними сусідніми осередками. Кон'юнктивні терми розширювача можуть з'єднувати одну клітинку з сусідньою всередині двох груп – макрокомірки 8 до 1 і 16 до 9. Коли паралельний розширювач використовується без загального розширювача, максимальна кількість кон'юнктивних термів у будь-якій логічній функції дорівнює 20, п'ять термів у самій макрокомірці і три додаткових групи з п'ти термів, з'єднаних з сусідніми макрокомірками.

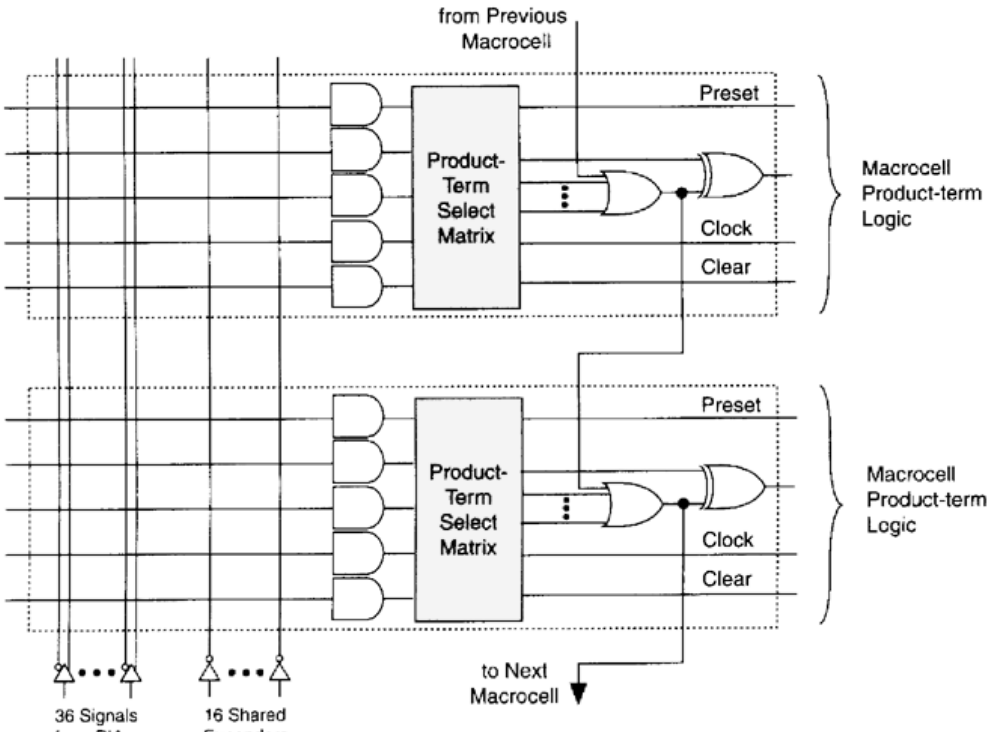


Рис. 8.41. Паралельні розширювачі

Рис. 8.42 зображує вхідний/вихідний контрольний блок для вхідних/вихідних контактів. Блок дозволяє кожному вхідному/вихідному контакту бути налаштованим як вхід або двонаправлений контакт. Мультиплексор OE control програмується для вибору Vcc, Gnd або одного з загальних виходів сигналу дозволу. Якщо вибирається Vcc, вихід макрокомірки з'єднується з вхідним/вихідним контактом. У разі вибору Gnd буфер буде відключений, і вхідний/вихідний контакт може бути використаний як вхід. Інакше буфер може бути керованим OE1n або OE2n. Програмне забезпечення фірми Altera може бути використано для оптимізації і поділу проєкту для розміщення його в логічних осередках і створення міжз'єднань між осередками.

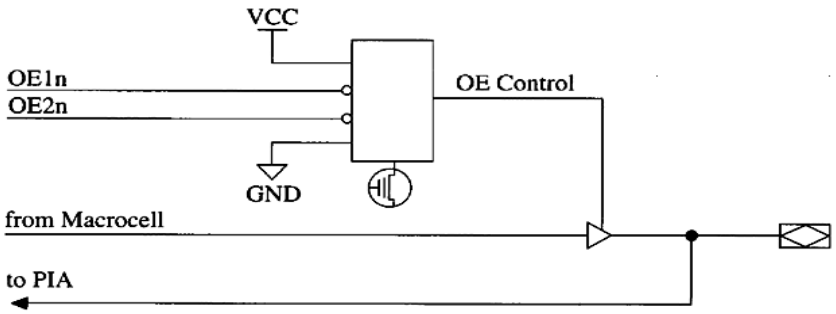


Рис. 8.42. Вхідний/вихідний контрольний блок для EPM

Наприклад, якщо використовувати програмне забезпечення Altera для реалізації двох біт-повного суматора, використовуючи рівняння, подані раніше, програма спочатку визначить, що T-тригер буде вимагати менше вентилів, ніж D-тригер, і потім розкладе рівняння для використання загальних розширювачів.

Результуючі рівняння мають вигляд

```
C3 = A1 A2 X01 + B1 C1 X02 + A1 B2 X01 + A2 B2,
```

де X01 = B1+C1 и X02 = A2+B2 – виходи загальних розширювачів;

```
T2 = Ad A1 B2'C1 + AdB1 B2'X03+Ad B1'B2 X04+Ad
A1'B2C1',
```

де X03 = A1+C1 и X04 = A1'+C1' – виходи загальних розширювачів;

```
T1 = Ad B1 C1' + Ad B1'C1.
```

Кожне логічне рівняння має менше або рівно п'ять термів, таким чином вони підходять для одного логічного осередка. Реалізація цих рівнянь вимагає три логічних осередки і чотири загальних рівняння. Використовуючи програмне забезпечення Altera, ми ввели схеми для пристрою гри в карти і скомпілювали проєкт. Результуючі рівняння вимагають 23 логічних осередків і вісім загальних розширювачів, і вони підходять для EPM7032 CPLD. Інші приклади використання Altera CPLD буде дано далі.

Altera виробляє кілька інших серій CPLD. Серія MAX 7000S подібна до серії MAX 7000, за винятком, що вона вбудована, програмована швидше, ніж вимагає програматор. Серія MAX 9000 є розширеною версією серії MAX 7000S, має вищу щільність і додаткові ресурси маршрутизації. Серії FLEX 8000 і FLEX 10K використовують осередки з заснованою на RAM конфігурованою пам'яттю замість заснованих на EEPROM осередках. Altera FLEX 10K (рис. 8.43) логічного сімейства вбудованого програмування забезпечує високу щільність логіки і RAM пам'ять у кожному пристрої. Логіка і міжз'єднання програмуються за допомогою конфігурованих RAM-осередків аналогічно, як і в Xilinx FPGA. На рис. 3.43 зображено блок-схему пристрою FLEX 10K.

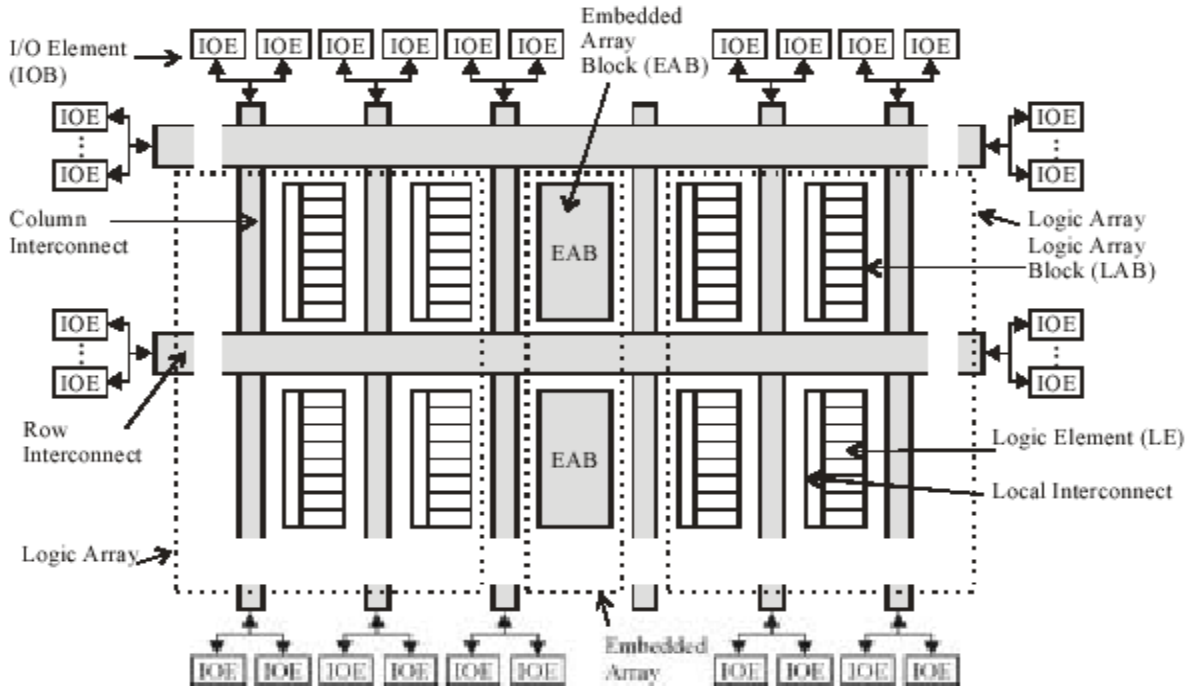


Рис. 8.43. Блок-схема пристрою FLEX 10K

Кожен LAB містить вісім логічних елементів і локальний з'єднувальний канал. EAB містить 2048 біт RAM пам'яті. LAB і EAB можуть з'єднуватися через швидкі горизонтальні і вертикальні сполучні канали. Кожен вхідний/вихідний елемент (IOE) може використовуватися як вхід, вихід або двонаправлений контакт. Кожен IOE містить двонаправлений буфер і тригер, який може бути використаний для збереження або вхідних, або вихідних даних. Пристрій FLEX 10K дає від 72 до 624 LAB, від 3 до 12 EAB і до 406 lOE. У типовому застосуванні він може використовувати від 10 до 100 тисяч еквівалентних вентилів при звичайній реалізації.

Рис. 8.44 зображує блок-схему FLEX 10K, яка містить вісім логічних елементів (LE). Локальні сполучні канали мають 22 або більше входів з горизонтальними з'єднаннями і вісім входів, які подають значення назад з LE виходів. Кожен LE має чотири входи даних з локальних сполучних каналів, як і додаткові входи управління. LE-виходи можуть бути направлені через горизонтальні і вертикальні з'єднання.

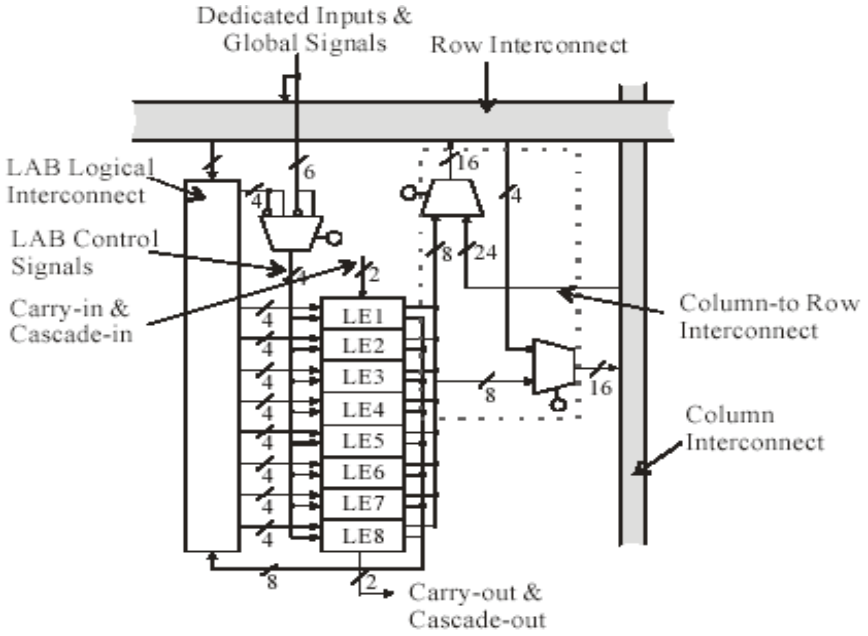


Рис. 8.44. Блок логічної матриці FLEX 10K

Кожен логічний елемент (рис. 8.45) містить функціональний генератор, який може реалізовувати будь-яку функцію чотирьох змінних за допомогою таблиці перетворення (lookup table, LUT).

Каскадне з'єднання (cascade chain) забезпечує з'єднання сусідніх LE, таким чином можуть бути реалізовані функції більше чотирьох змінних. Каскадне з'єднання може бути використано для AND або OR конфігурацій, як ілюструє рис. 8.46.

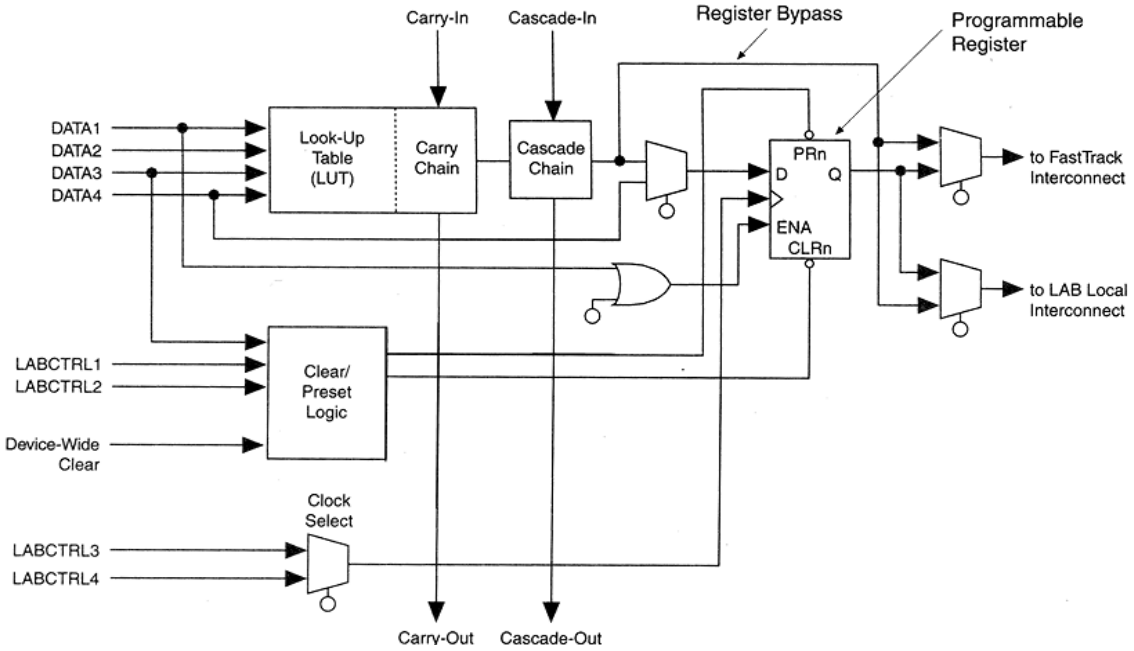


Рис. 8.45. Логічний елемент FLEX 10K

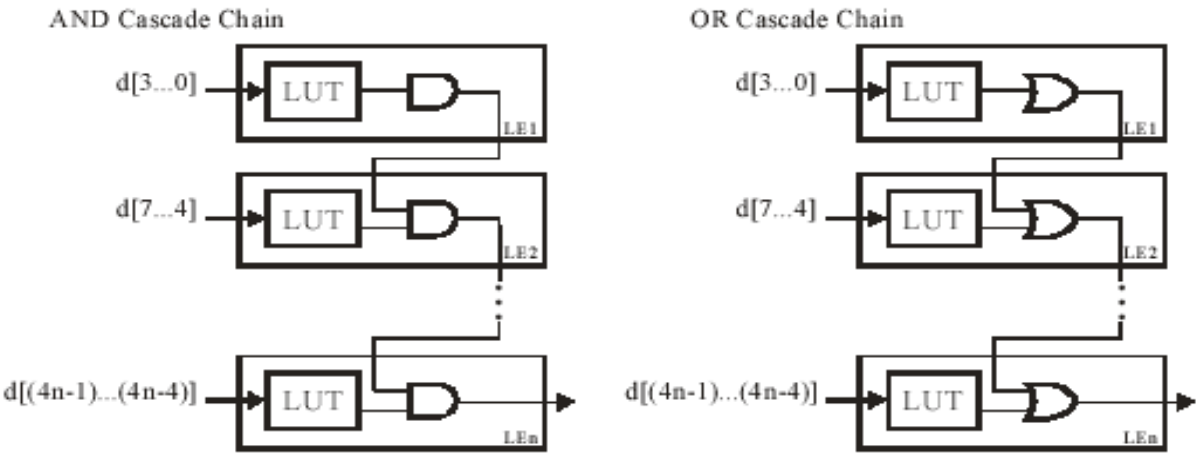


Рис. 8.46. Операція каскадного з'єднання

Коли використовується арифметичний режим, LE може реалізовувати суму і перенесення для одного біта повного суматора. Ланцюг перенесення забезпечує передачу перенесень між сусідніми осередками. Кожен LE містить один D-тригер із

входами дозволу синхронізації і асинхронного скидання і установлення. Вихід LE може йти з тригера або прямо з комбінаційної логічної схеми. Функції більш ніж чотирьох змінних вимагають для реалізації кілька LE. Наприклад, функція шести змінних Z (a, b, c, d, e, f) може бути реалізована за допомогою шести LE. Застосовуючи теореми розкладання, $Z(a, b, c, d, e,f) = a'b'Z_0(c,d,e,f)+a'bZ_1(c,d,e,f)+ab'Z_2(c,d,e,f)+abZ_3(c,d, e,f)$.

Кожна функція Z_0, Z_1, Z_2 і Z_3 може бути реалізована на одному LE. Виходи цих LE можуть бути реалізовані на одному LE. Виходи цих LE можуть бути з'єднані з входами інших LE через локальне з'єднання. Кожна функція чотиризмінних $Y_0 = a'b'Z_0 + a'bZ_1$ і Y_1= ab'Z_2 + ab'Z_3 вимагає ще LE. Y_0 і Y_1 можуть бути з'єднані через OR, використовуючи каскадне з'єднання, таким чином, не потрібно додаткових LE. Рис. 8.47 зображує вкладений блок матриці. Входи від горизонтальних з'єднань йдуть через EAB локальні з'єднання і можуть бути використані як входи даних або адресні входи до EAB.

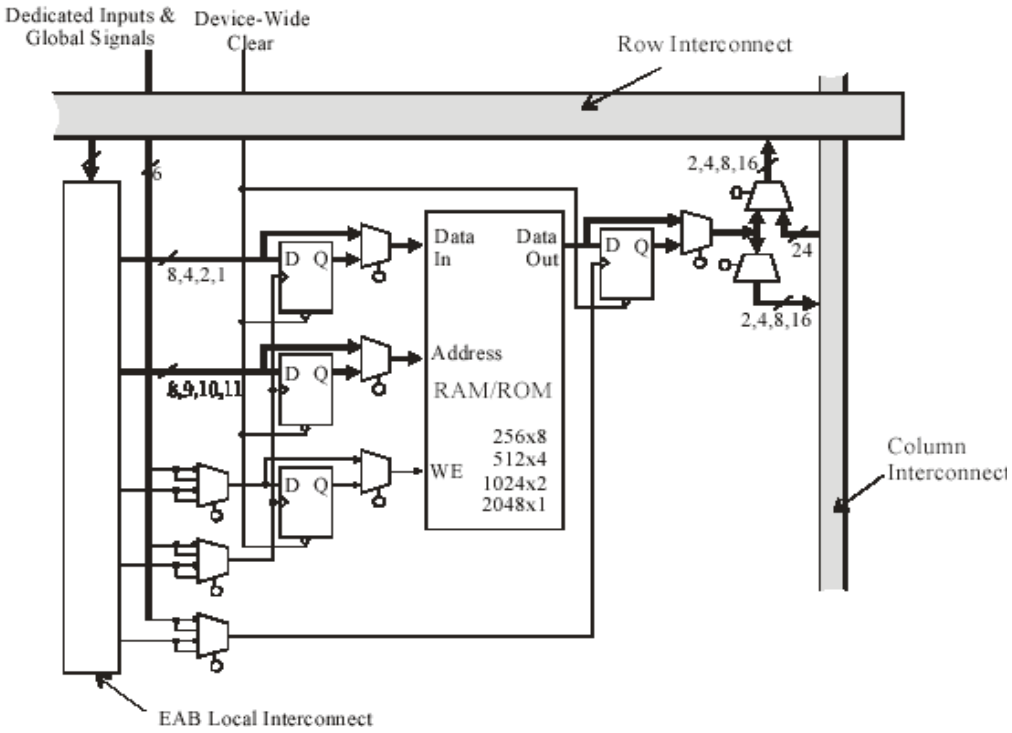


Рис. 8.47. Вбудований блок масиву FLEX 10K

Масив внутрішньої пам'яті може бути використаний як RAM або ROM розмірами 256x8, 512x4, 1024x2 або 2048x1. Кілька EAB можуть бути використані разом для формування більшої пам'яті. Виходи даних пам'яті можуть бути направлені або через горизонтальні, або вертикальні з'єднання. Всі входи і виходи пам'яті з'єднуються з регістрами, таким чином, пам'ять може бути керованою в синхронному режимі. Інакше регістри можуть бути блоковані, і пам'ять функціонує як асинхронна.

Використання таких CPLD, як серія FLEX 10K, дозволяє виконувати складні цифрові системи на одній мікросхемі.

Описано декілька типів FPGA і CPLD і процедури проєктування з використанням цих пристроїв. Використання відповідних програмних засобів CAD полегшує декомпозицію проєкту за логічними блоками, розміщення цих логічних блоків у логічній матриці і маршрутизацію з'єднань між блоками.

Контрольні запитання

1. Що являє собою принцип 3М при побудові мікропроцесорних систем?
2. Що являє собою принцип агрегатування?
3. Організація одно- і багатомагістральних мікропроцесорних систем.
4. Призначення схеми синхронізації та початкового скидання.
5. Організація доступу до пристроїв введення/виведення.
6. Поясніть поняття діагностики несправності, налагодження.

9. СИНТЕЗ З ВИКОРИСТАННЯМ МОВИ ОПИСУ АПАРАТУРИ VERILOG

9.1. Синтез конструкцій у Verilog

У табл. 9.1 наведено методи побудови конструкцій у Verilog за допомогою ключових слів або опису.

Таблиця 9.1

Методи побудови конструкцій у Verilog за допомогою ключових слів або опису

Конструкції	Ключове слово або опис	Примітки
Порти	input, inout, output	
Параметри	parameter	
Визначення модуля	module	
Сигнали і змінні	wire, reg, tri	Дозволені вектори
Реалізація компонентів	реалізація модуля, реалізація вентильного примітива	Наприклад, mymux m1(out, i0, i1, s); nand (out, a, b);
Функції і завдання	function, task	Ігноруються часові параметри
Поведінковий опис	always, if, then, else, case, casex, casez	Initial не підтримується
Процедурні блоки	begin, end, іменні блоки, disable	Дозволяється використання оператора для іменних блоків
Data flow	assign	Інформація про затримки ігнорується
Цикли	for, while, forever	Цикли while і forever мають містити конструкції @(posedge clk) або (@(negedge clk)

Далі розглянемо приклади використання операторів assign і always для комбінаційної і послідовної логіки у Verilog синтезі.

Оператор assign реалізується комбінаційною логікою: а) assign out=(a & b)|c; б) assign {c_out, sum} = a + b + c_in; в) assign out=(s)? a: b. Результати синтезу наведено на рис. 9.1.

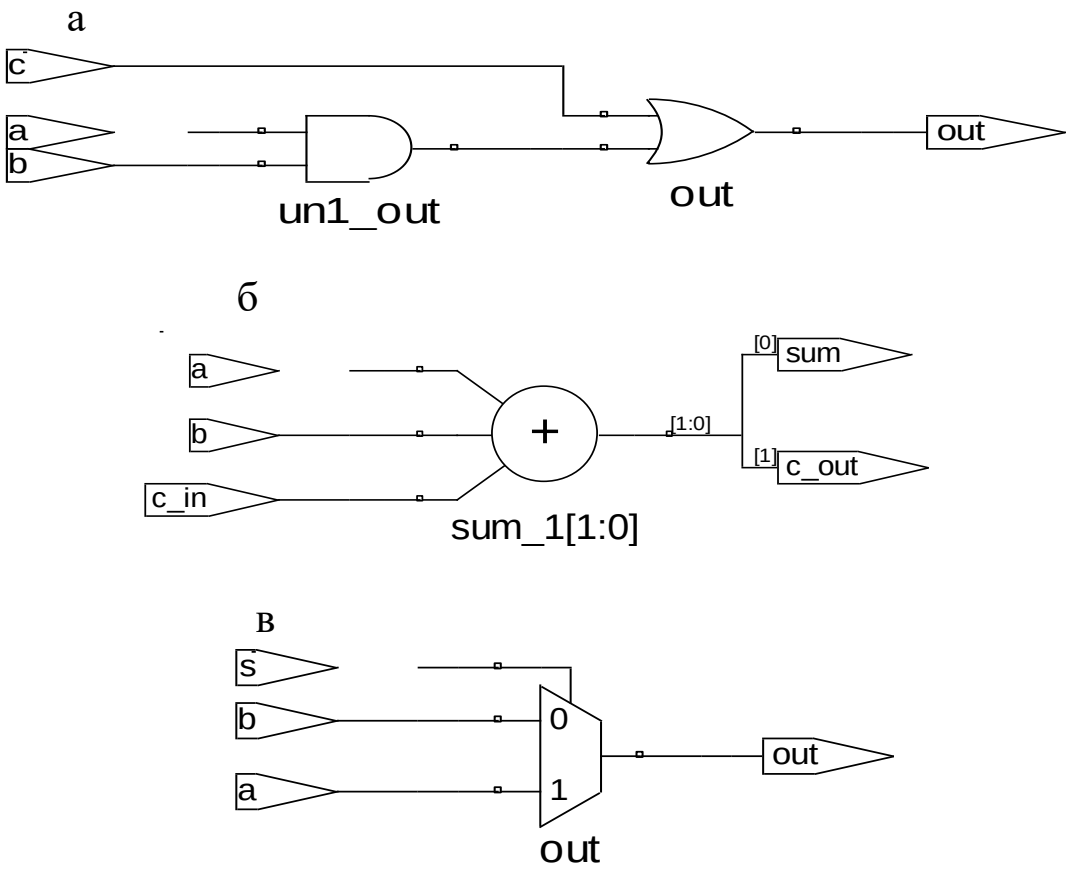


Рис. 9.1. Результат синтезу

Використання блока always для комбінаційної логіки: always @(a or b or c_in) {c_out, sum} = a + b + c_in; результат синтезу наведено на рис. 9.2.

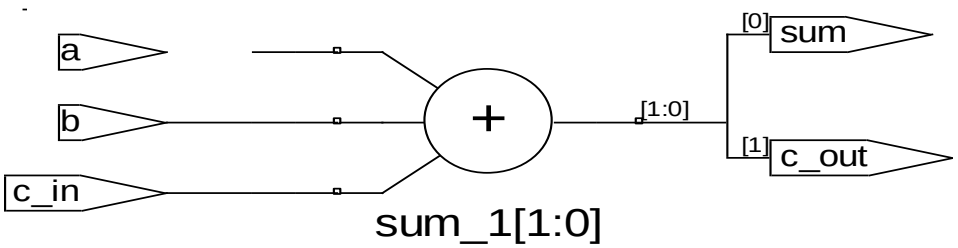


Рис. 9.2. Результат синтезу

Використання always для послідовної логіки: синхронний D-тригер без скидання наведено на рис. 9.3, а, синхронний D-тригер зі скиданням наведено на рис. 9.3, б.

а

```
module flop (C, D, Q);
input C, D;
output Q;
reg Q;
always @(posedge C)
      Q = D; endmodule
```

б

```
module flop1 (C, D, CLR, Q);
input C, D, CLR;
output Q; reg Q;
always @(negedge C or posedge CLR)
        if (CLR)
              Q = 1'b0;
        Else Q = D; endmodule
```

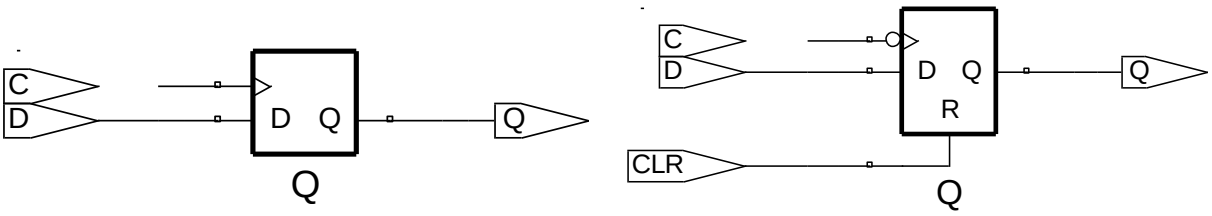


Рис. 9.3. Результат синтезу: а – синхронний D-тригер без скидання; б – синхронний D-тригер зі скиданням

Використання блокових і неблокових операторів у Verilog синтезі. Не допускається використовувати обидва типи операторів для одного сигналу.

1) блокові оператори

```
always @(in1)
begin
   if (in2) out1 = in1;
   else out1 <= in2;
end
```

2) неблокові оператори

```
if (in2) begin
   out1[0] = 1'b0;
   out1[1]  <=  in1;  end
else begin
   out1[0] = in2;
   out1[1] <= 1'b1; end
```

Використання оператора IF і CASE у Verilog синтезі. Використання паралельного оператора CASEX у Verilog синтезі, результати синтезу оператора наведено на рис. 9.4.

```
module SelectOneOf (A, B, Z);
      input [1:0] A, B; output [1:0] Z; reg [1:0] Z;
      always @(A or B)
            if (A>B)   Z=A;
            else       Z=B;
      endmodule
```

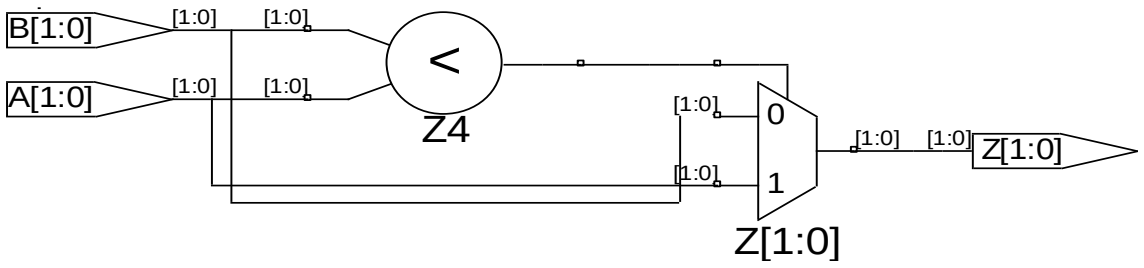


Рис. 9.4. Результат синтезу

Побудова тригера з оператора if: module SelectOneOf (A, B, Z); результат синтезу оператора if наведено на рис. 9.5.

```
module SelectOneOf (A, B, Z);
        input A, B;
        output Z;
        reg Z;
                always @(A or B)
                                if (A) Z=B;
        endmodule
```

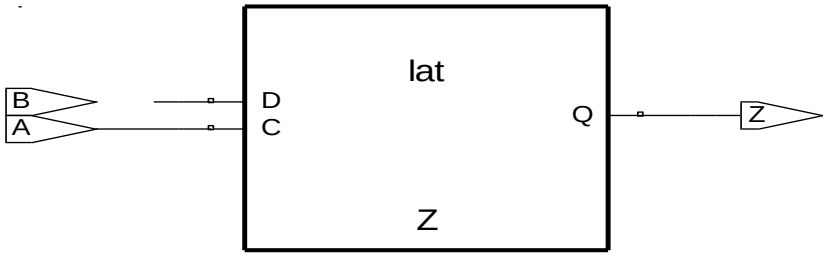


Рис. 9.5. Результат синтезу

Використання оператора CASE у Verilog синтезі, результат синтезу оператора CASE наведено на рис. 9.6.

```
  module ALU (Op, A, B, Z);
        input [1:2] Op; input [0:1] A, B;
        output [0:1] Z; reg [0:1] Z;
        parameter ADD = 'b00, SUB = 'b01, MUL = 'b10, DIV = 'b11;
        always @(Op or A or B)
                        case (Op)
                                ADD : Z = A + B;
                                SUB : Z = A - B;
                                MUL : Z = A * B;
```

```
            DIV : Z = A / 2;
        endcase
    endmodule
```

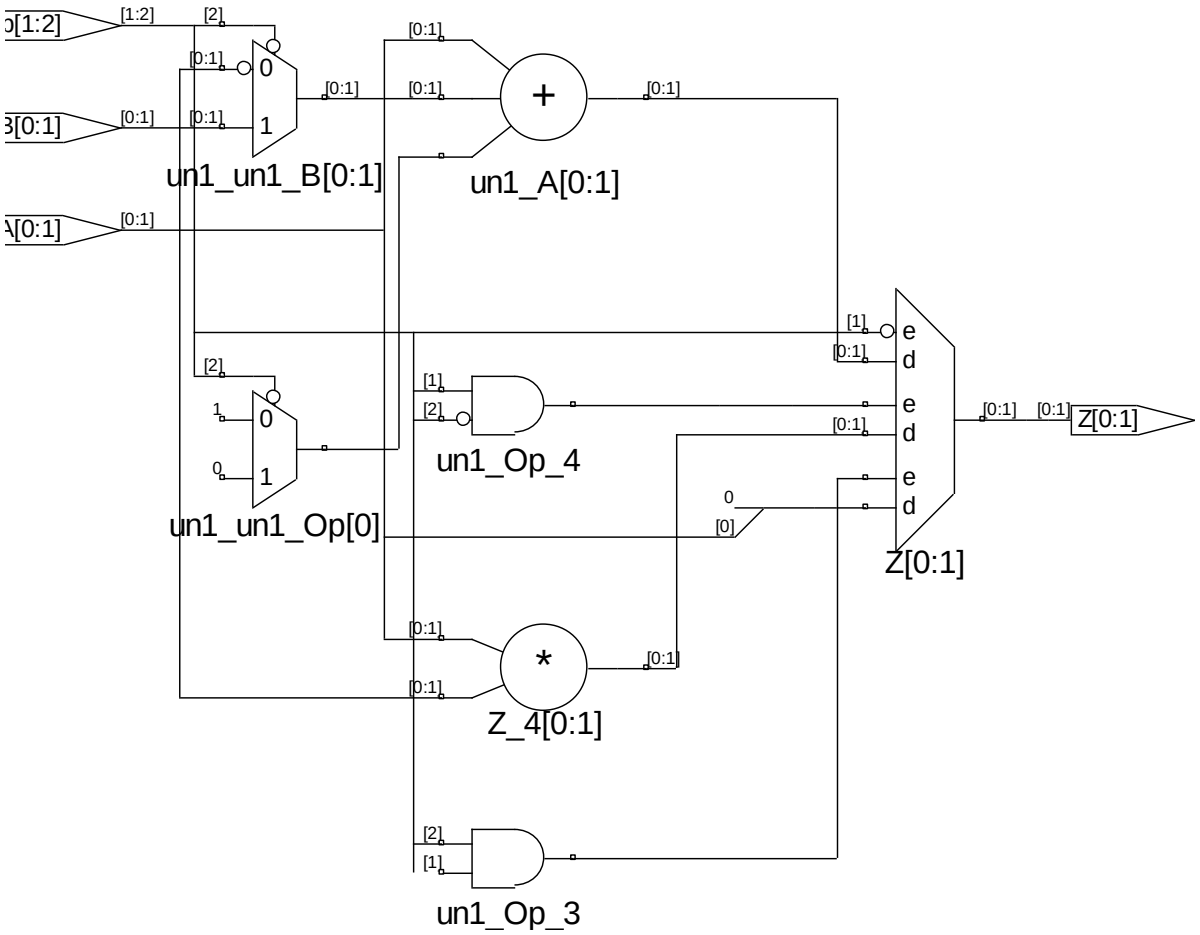


Рис. 9.6. Результат синтезу оператора CASE

Побудова тригера з оператора CASE module module NextStateLogic; результат синтезу оператора CASE наведено на рис. 9.7.

```
    module NextStateLogic (NextToggle, Toggle);
        input [1:0] Toggle; output [1:0] NextToggle; reg [1:0] NextToggle;
        always @(Toggle)
            case (Toggle)
                2'b01: NextToggle = 2'b10;
                2'b10: NextToggle = 2'b01;
            endcase
    endmodule
```

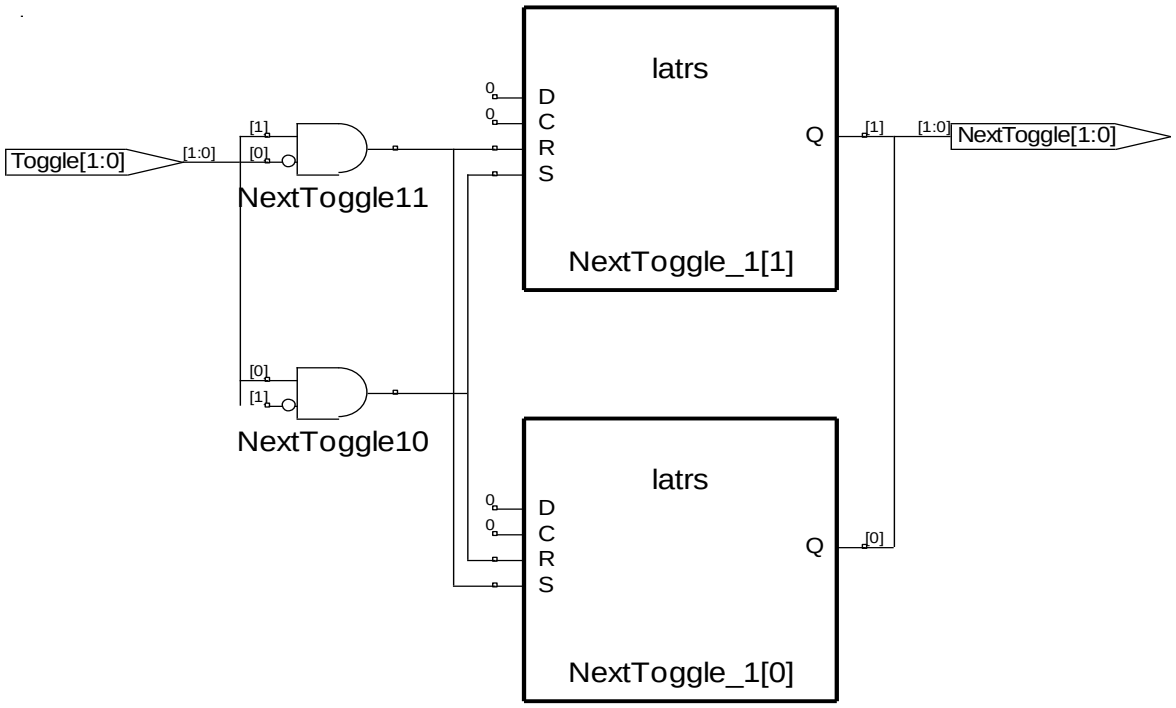


Рис. 9.7. Результат синтезу тригера з CASE

Побудова комбінаційної схеми з CASE: always @(Toggle); результат синтезу наведено на рис. 9.8.

```
always @(Toggle)
        case (Toggle)
                2'b01: NextToggle = 2'b10;
                2'b10: NextToggle = 2'b01;
                default: NextToggle = 2'b01;
        endcase
```

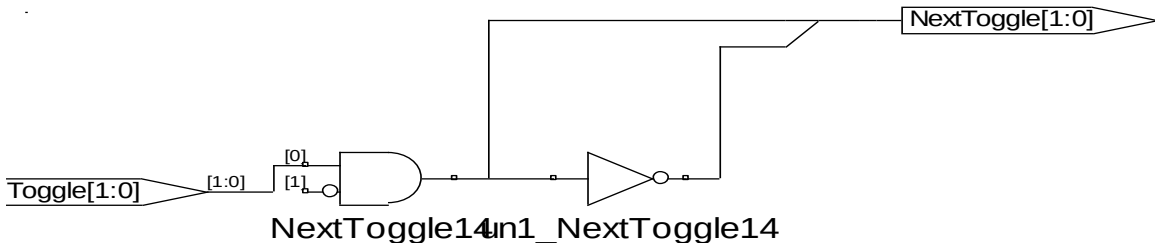


Рис. 9.8. Результат синтезу оператора комбінаційної схеми з CASE

Побудова тригера з оператором FULL CASE: always @(Toggle); результат синтезу оператора FULL CASE наведено на рис. 9.9.

```
always @(Toggle)
        case (Toggle)          //synthesis full_case
            2'b01: NextToggle = 2'b10;
            2'b10: NextToggle = 2'b01;
        endcase
```

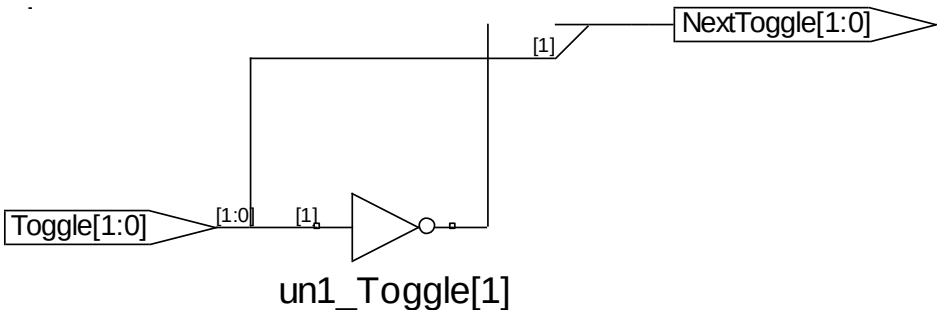


Рис. 9.9. Результат синтезу FULL CASE

Використання паралельного оператора CASEX у Verilog синтезі module ParallelCase (NextToggle, Toggle); результат синтезу наведено на рис. 9.10.

```
module ParallelCase (NextToggle, Toggle);
    input [2:0] Toggle; output [2:0] NextToggle; reg [2:0] NextToggle;
always @(Toggle)
        casex (Toggle)          // synthesis parallel_case
            3'bxx1: NextToggle = 3'b010;
            3'bx1x: NextToggle = 3'b110;
            3'b1xx: NextToggle = 3'b001;
            default: NextToggle = 3'b000;
        endcase
    endmodule
```

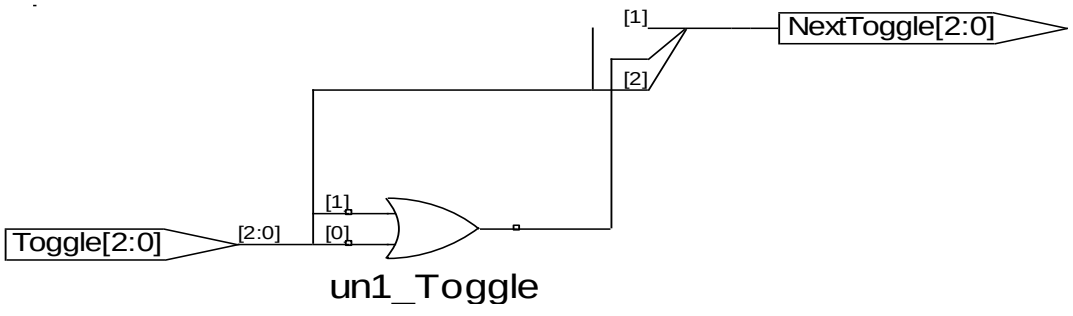


Рис. 9.10. Результат синтезу

Verilog-шаблон синтезованого послідовного пристрою з асинхронними і синхронними керуючими сигналами.

```
always @(posedge A or negedge B or negedge C or posedge Clock)
        if (A)                  //posedge A
            <statement>             //asychronous logic
        else if (!B)            // negedge B
            <statement>             //asynchronous logic
        else if (!C)            // negedge C
            <statement>             // asynchronous logic
        . . .           //Любое количество конструкций else if
        else                // posedge Clock
            <statement>             //synchronous logic
```

Verilog-шаблон послідовного пристрою з синхронними керуючими сигналами.

```
always @(posedge Clock)
        if (A)                  //posedge A
            <statement>             //sychronous logic
        else if (!B)            // negedge B
            <statement>             //synchronous logic
        else if (!C)            // negedge C
            <statement>             //synchronous logic
        . . .           //Любое количество конструкций else if
        else                // posedge Clock
            <statement>             //synchronous logic
```

Приклад синтезуючої Verilog-моделі лічильника.

```
module AsyncPreClrCounter (Clock, Preset, UpDown, Clear, PresetData, Counter);
        parameter Num_bits = 2;
        input Clock, Preset, UpDown, Clear;
        input [Num_bits-1:0] PresetData;
        output [Num_bits-1:0] Counter;
        reg [Num_bits-1:0] Counter;
```

Контрольні запитання

1. Принцип модульної організації МПС.
2. Магістральний спосіб обміну інформацією в МПС.
3. Мікропрограмна реалізація управління в МПС.
4. Типова структура тришинної МПС.
5. Організація доступу до пам'яті МПС.
6. Призначення таймера в МПС.
7. Організація прямого доступу до пам'яті в МПС.
8. Призначення контролера переривань у МПС.
9. Перелічіть основні етапи проєктування МПС.

10. ІМПЛЕМЕНТАЦІЯ ПРИСТРОЇВ І АНАЛІЗ ЧАСОВИХ ПАРАМЕТРІВ

10.1. Методи та етапи тестопридатного проєктування обчислювальних систем і пристроїв на ПЛІС

Завдання синтезу схем на ПЛІС були сформульовані і вирішувались стосовно реалізації схем на програмованих логічних матрицях. У них проєктування логічних схем, як правило, зводиться до розв'язування логічних матричних рівнянь, вибору мінімальної множини простих імплікантів, які покривають у найкоротшій формі задану множину функцій виходів і збудження елементів пам'яті. Були розроблені алгоритми синтезу, у яких використовувалися відомі в теорії булевих функцій систематичні та евристичні прийоми для знаходження МДНФ, вирішення завдань економічного і протигоночного кодування [2-4].

Різні завдання логічного проєктування ДП на програмованих матричних логіках (ПМЛ) формуються і вирішуються з використанням логічних матричних рівнянь. Розв’язання конкретної задачі синтезу зводиться до знаходження коренів деякого рівняння, у якому одні величини вважаються заданими, а інші потрібно знайти. Запропоновано варіант протигоночного кодування станів автомата, який не завжди забезпечує його економічну реалізацію на ПМЛ.

Основні процедури алгоритмів синтезу зведені до впорядкування множини інтервалів (кон'юнкція), на яких задана система ЛФ. У цій області визначення полегшується вибір простих імплікант, які покривають у найкоротшій формі задану систему ЛФ. В алгоритмах кодування внутрішніх станів використовується метод Трейсі, який хоча і забезпечує протигоночне кодування автомата з найменшою кількістю змінних станів, проте пов'язаний з трудомісткими процедурами пошуку дихотомій пар станів, знаходженням множин максимально сумісних дихотомій і розв’язанням задачі покриття, що вимагає більшого обсягу обчислень.

Синтез синхронних послідовних схем у тому чи іншому вигляді зводиться до розв’язання задачі економічного або

протигоночного кодування внутрішніх станів автомата. Загальним недоліком перерахованих робіт є те, що запропоновані методи і алгоритми синтезу дискретних пристроїв на ПЛІС зводяться в основному до проблем синтезу систем логічних функцій і кодування внутрішніх станів автомата і базуються на великій кількості джерел і результатів у сфері структурних методів цифрових автоматів. Не оцінюється складність пропонованих алгоритмів, які найчастіше подані в узагальненій формі, далекій від практичної реалізації, і описані з використанням великої кількості понять і термінології, важко сприймаються на інженерному рівні. На етапі кодування внутрішніх станів автоматних моделей ДУ не бралися до уваги вимоги тестопригодного проєктування.

Алгоритм синтезу тестів для логічних схем на ПЛІС

Перевагою такого способу отримання тесту є виключення процедури моделювання для визначення повноти покриття заданого класу несправностей. При подачі тестових наборів з множини Т0 за відсутності несправності на виході секції ПЛІС завжди спостерігається значення логічного 0, а за наявності несправності – логічної 1. При подачі наборів з безлічі Т1 значення на виході секції протилежні.

Алгоритм синтезу тестів T_0 для однієї секції ПЛІС:

1. Ввести список термів секції ПЛІС.
2. Для кожного терму отримати список термів розширення.
3. Видалити з отриманих списків надлишкові терми розширення.
4. Отримати з даних списків термів розширення список взаємно непересічних термів розширення.
5. У кожному термі розширення отриманого списку визначити вільний мінтерм.
6. Отримані вільні мінтерми включити до списку тестів.
7. Множину вільних мінтермів на кроці 6 утворює множина T_0.

Алгоритм синтезу тестів T_1 для однієї секції ПЛІС:

1. Ввести список термів секції.
2. Для кожного терму отримати список вільних підтермів, шляхом додавання букв, відсутніх у вихідному термі.

3. Для кожного з множини вільних підтермів цього терму отримати тестові набори.

4. Отримані вільні мінтерми включити до списку тестів.

5. Множину вільних мінтермів на кроці 4 утворює множина T_1.

При каскадному з'єднанні секцій ПЛІС для синтезу множини тестів всієї схеми необхідно ранжувати її за секціями. Багатовихідна однорангова схема має кількість гілок, дорівнювану кількості секцій. Для такої схеми множину тестів отримують шляхом простого склеювання тестових наборів секцій. Наступний клас схем, реалізованих на ПЛІС, - це багаторангові схеми. Такою схемою, що має лише одну гілку, є каскадована схема. Схема з гілками, що сходяться, має один вузол, у якому сходяться дві або більше гілок. Схема з розгалуженнями має вузол з розбіжними гілками на дві або більше гілок. У схемі з обома видами розгалужень гілки сходяться в одному вузлі і розходяться з нього. Крім розглянутих типів схем, існують різні види комбінацій цих схем. З огляду на ці структурні особливості, алгоритм синтезу тестів для схеми, реалізованої на ПЛІС, може бути сформульовано таким чином.

Алгоритм:

1. Ввести опис схеми з розбиттям її за секціями.

2. Провести ранжування схеми з визначенням секцій з максимальним рангом.

3. Вибрати секцію, що має максимальний ранг.

4. Для даної секції отримати множину тестів T_0.

5. Для даної секції отримати множину тестів T_1.

6. Якщо множина тестів для даної області порожня, то занести в нього отримані T_0 і T_1, інакше провести склеювання отриманих наборів з наявними.

7. Якщо в даній області ще є секції, то перехід до кроку 4, інакше вивести множину тестів для даної області.

8. Якщо є ще секції з максимальним рангом, то перехід до кроку 3.

9. Якщо отримано множину тестових наборів, то виконати операцію склеювання даних наборів для отримання мінімальної множини тестів.

10. Сформувати множину тестів для всієї схеми.

10.2. Місце імплементації в процесі проєктування. Використання пакета Xilinx ISE

Огляд потоку проєктування Xilinx наведено на рис. 10.1.

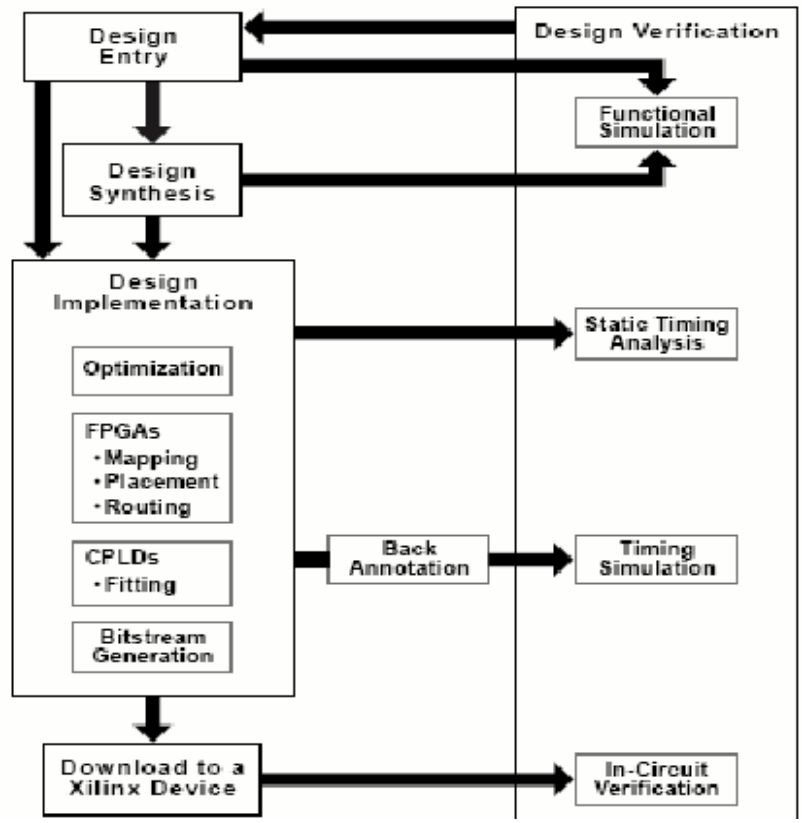


Рис. 10.1. Етапи проєктування Xilinx

Після синтезу формується проєкт, що складається з визначених цільовим пристроєм вентилів. Нехай цей проєкт верифікований на даному рівні. Для того щоб розмістити цей проєкт, у мікросхемі необхідно виконати імплементацію – операцію Device Implementation.

Етапи імплементації для FPGA: Translate; Map; Place and Route; Configure;

Етапи імплементації для CPLD: Translate; Fit – розподілити проєкт за макрокомірками; Configure – створення JED-файла для програмування;

Translate – інтерпретація проєкту і перевірка його на правильність опису ('design rule check'). Складається з набору різних програм, які використовуються для імпорту netlist проєкту

і підготовки його до розміщення. Ці програми можуть змінюватися від виробника до виробника. У загальному випадку вони можуть включати оптимізацію, трансляцію до фізичних елементів, перевірку правильності моделі (наприклад чи існує в мікросхемі необхідна кількість синхробуферів тощо). Етап трансляції зазвичай закінчується створенням звіту, який містить попередження і помилки.

Map – Обчислення і розподіл ресурсів цільової мікросхеми для проєкту.

Place and Route – розміщення CLB в логічні позиції мікросхеми і створення засобів маршрутизації між ними. Більшість розробників пропонують автоматичні інструменти place and route, таким чином користувачеві нема потреби детально вникати в деталі цього процесу. Деякі мають інструменти, які дозволяють виконувати операції place and/or route вручну для найбільш критичних шляхів, зберігаючи кращі властивості автоматичних інструментів. Одним із прикладів таких інструментів є програма Floorplanner з пакета Xilinx ISE. Процес Place and Route вимагає великого проміжку часу для виконання, оскільки розміщення великого проєкту за компонентами мікросхеми, виконання міжз'єднань, гарантуючи їх правильність і відповідність вимогам проєктувальника є досить складним завдання.

Fit (Fitting) – даний етап виконується при проєктуванні на CPLD і означає підгонку (Fit) проєкту до цільового пристрою, що має фіксовану структуру, з якої вибираються вентилі і з'єднання для формування проєкту. Це зазвичай швидкий процес. Потенційним джерелом проблем у даному випадку є попередній розподіл вхідних/вихідних контактів, зазвичай так зване Pin Locking (часто вони формуються на попередніх ітераціях проєктування і залежать від розведення друкованої плати). Архітектури (подібні Xilinx XC9500 і CoolRunner CPLD), що підтримують Pin Locking, мають перевагу. Вони дозволяють проєктувальнику зберігати вихідний розподіл вхідних/вихідних контактів, не дивлячись на будь-яку кількість внесених змін. Pin locking є важливою властивістю при програмуванні в системі (In-System Programming, ISP).

Configure – створення виконуваного файла конфігурації FPGA або JED-файла для програмування CPLD.

Downloading or Programming – це завершальний етап імплементації. Термін Download зазвичай стосується енергозалежних пристроїв, таких як SRAM FPGAs. У цьому випадку виконується завантаження конфігурації мікросхеми в пам'ять пристрою. Передаваний Bitstream містить всю інформацію про використовувані ресурси і є індивідуальним для кожного проєкту. Оскільки SRAM-пристрої втрачають свою інформацію після вимкнення живлення, конфігурація має зберігатися в іншому місці. Зазвичай для цього використовується послідовне ПЗП. Program використовується для програмування енергонезалежних програмованих логічних пристроїв, включаючи послідовні ПЗП. Програмування виконує ті самі функції, що і завантаження, за винятком того, що інформація про конфігурацію зберігається після вимкнення живлення. Для антиф'юзних пристроїв програмування виконується тільки один раз (звідси термін – однократно програмований пристрій (One-Time Programmable, OTP)).

Програмування CPLD може бути виконано в системі In-System через інтерфейс JTAG або з використанням відповідного для пристрою вибору програм, наприклад Data I/O (рис. 10.2). JTAG Boundary Scan – також відомий як стандарт IEEE/ANSI 1149.1_1190 – являє собою набір правил проєктування для спрощення процесу тестування, програмування і налагодження мікросхеми, плати або системи. Перевага програмування в системі в тому, що змінити конфігурацію пристроїв можна не виймаючи мікросхему з плати.

Звіти, що формуються в процесі імплементації (пакет Xilinx ISE0):

- i. Translate Report – показує помилки в проєкті або UCF (User Constraints File);

- ii. Map Report – показує ресурси, використовувані в проєкті. Докладний звіт map може бути обраний у властивостях для map. Докладний звіт описує trimmed (усічену) і merged (поєднану) logic. Також точно описує, де і яка частина проєкту розміщується в реальному пристрої;

- iii. Post-Map Static Timing Report – показує тільки затримки елементів, не враховуючи затримки ліній. Якщо затримки логіки не задовольняють часовим параметрам, то додані до неї затримки зв'язків тільки погіршать ситуацію;

- iv. Place and Route Report – показує покроково процес виконання етапу Place & Route. Інструменти, які виконують Place & Route, мають отримати інформацію про часові вимоги. Це може бути It will list the given constraints and report how comfortably the design fell within or how much it failed the constraints;

- v. Asynchronous Delay Report – містить затримки гіршого шляху в проєкті, що враховують затримки елементів і шляхів;

- vi. Pad Report – містить список розподілу зовнішніх контактів з інформацією про drive strength і стандарті сигналу;

- vii. Post Place and Route Static Timing Report – враховує затримки міжз'єднань.

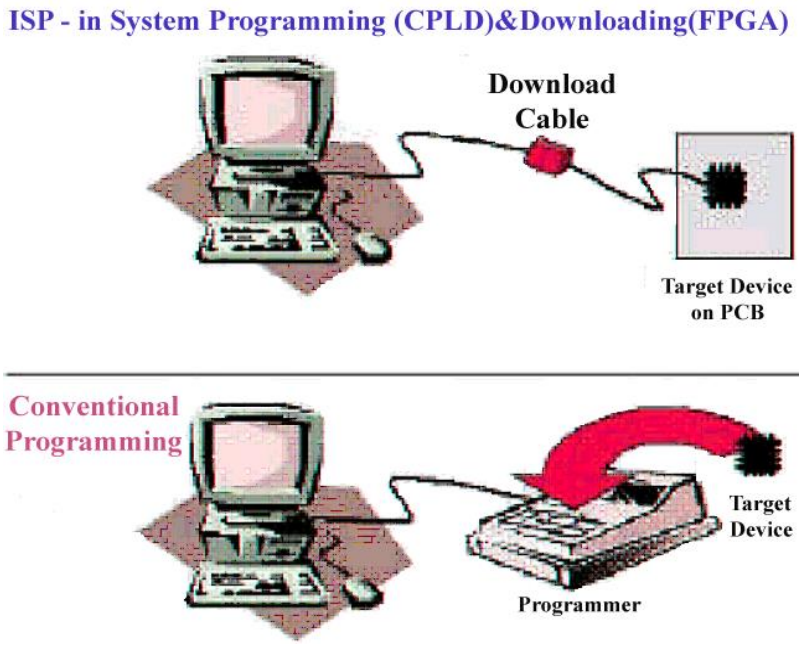


Рис. 10.2. Програмування та завантаження даних у кристал

Тестування пристрою на різних стадіях проєктування (пакет Xilinx ISE)

Functional Simulation використовується для перевірки поведінки пристрою. Якщо на цьому етапі виявляються помилки, проєктувальник виправляє їх у моделі і повторює моделювання. На цей етап витрачається 5 0% часу розроблення пристрою.

Після того як пристрій починає працювати правильно, виконують синтез схеми пристрою. Після синтезу виконується моделювання поведінки пристрою, що виконує перевірку правильності синтезу схеми з RTL. Виконується аналіз часових параметрів, що враховує затримки елементів.

На етапі Place & Route програма Timing-Driven Place & Route (TDPR) дозволяє користувачеві вказувати часові параметри, які будуть використовуватися під час розміщення проєкту.

Static Timing Analyser – зазвичай ця частина розробляється користувачем програмного забезпечення, дозволяє отримати інформацію про шляхи в проєкті. Ця інформація є точною і може бути переглянута різними способами (наприклад відображення всіх шляхів проєкту і сортування їх від найдовшого до найкоротшому).

На додачу до цього, користувач може використовувати докладну інформацію про розміщення проєкту в мікросхемі, яка може бути передана назад у програму моделювання разом з точною інформацією про часові параметри системи. Цей процес називається Back-Annotation і його перевага в точному описі часових параметрів одночасно з поведінкою проєкту. В обох випадках часові параметри будуть відображувати затримки логічних блоків, точно так, як і міжз'єднань. We will do Timing Simulation a little later in the design flow.

10.3. Стандарт IEEE P1076.4: Timing methodology (VITAL) (методологія часового аналізу)

Одним з найважчих завдань, що вирішується в процесі проєктування, є часовий аналіз. З цієї причини механізми обробки часових параметрів займають значну частину мови VHDL. Але він пропонує тільки спосіб і не має методик для детального опису або тестування часової поведінки схеми. Для компенсації цього недоліку організувалася група розробників VHDL Initiative TOWARDS ASIC Libraries (VITAL), яка запропонувала спосіб включення в VHDL-моделі докладної часової інформації і стандартний інтерфейс для back-annotation часових даних від layout інструментів. Була створена бібліотека примітивних компонентів (вентилі, тригери, регістри тощо), які відповідають правилам методики. Результатом є можливість виконання докладного тестування часових параметрів, що гарантує те, що створений пристрій буде функціонувати, як і його модель.

Ідея створення VITAL-бібліотек виникла в 1992 році на Міжнародному форумі користувачів VHDL (VHDL International

Users 'Forum). Тоді великим недоліком VHDL була відсутність ASIC-бібліотек і єдиного ефективного методу для опису і моделювання часових характеристик. У той час як це завдання вже було вирішено в інших мовах.

Було запропоновано VITAL-стандарт, представлений пакетом, що містить типи для докладного опису pin-to-pin і distributed затримок, часові константи, включаючи максимальне, мінімальне та номінальні значення. Пакет також містить процедури для виконання часового аналізу. Описано імена і типи generic-констант, які використовуються для створення коректної поведінки і верифікації часових констант. Замість безпосереднього внесення в модель часових параметрів методологія передбачає їх зчитування програмою моделювання з файла формату Standard Delay File (SDF). Причина цього рішення полягає в тому, що цей формат на той час вже широко використовувався і таке читання даних виконується швидше, ніж обробка VHDL опису і реконфігурація моделі.

VHDL Initiative TOWARDS ASIC Libraries (VITAL) була сформована на Design Automation Conference'92. Під час розроблення VITAL опис перейшов до робочої групи P1076.4 Working Group, яка виконала стандартизацію результуючої методики (Стандарт – IEEE P1076.4: Timing Methodology (VITAL)). До організаційного комітету входили представники VHDL Technology Group, Ryan & Rayn, Synopsys, Cadence, Hewlett-Packard і Texas Instruments. Крім того, у роботі брали участь понад 45 компаній.

Для розроблення VITAL-стандарту були використані існуючі:

а) формат Stantard Delay File (SDF);

б) деякі елементи з пакета Std_Timing групи VHDL Technology Group, а також спеціальні часові і поведінкові техніки, запропоновані Rian & Rian (фірма);

в) існуючі ASIC бібліотеки.

10.4. VITAL-бібліотеки у VHDL

VITAL-бібліотеки у VHDL для ASIC-розробників:

• поліпшення перенесення проєктів між різними EDA-проєктами;

• підвищення швидкості моделювання без виходу з середовища VHDL-проєкту (у 10-15 разів;)

• можливість використання стандартних VITAL-підпрограм для призначених для користувача моделей.

VITAL-бібліотеки в VHDL для постачальників напівпровідникової продукції:

• єдині бібліотеки, підтримувані всіма основними середовищами моделювання EDA;

• точна підтримка часових параметрів для субмікронної продукції;

• скорочення вартості розроблення та експлуатації.

VITAL-бібліотеки у VHDL для розробників EDA:

• концентрація на створенні інструментів і проєктів, а не бібліотек;

• стандартні методи створення моделей дозволяють підвищити оптимізацію і скоротити складність проєкту.

VITAL-бібліотеки у VHDL для проєктувальників мікросхем і систем:

• окремі VITAL-підпрограми доступні всім користувачем;

• стандартизація пакетів гарантує однакову поведінку на різних інструментах і платформах;

• VITAL-примітиви дозволяють спростити створення VHDL-моделей;

• підпрограми управління VITAL pin-to-pin затримкою дозволяють легко додати часові параметри в поведінкову модель;

• часова перевірка VITAL може бути використана для пошуку помилок функціонування проєкту;

• використання SDF back-annotation може бути необов'язковим, усі часові параметри.

Елементи VITAL-бібліотеки. Розроблено п'ять областей стандартизації, які використовуються для визначення так званих VITAL-сумісних моделей (VITAL compliant models) і VITAL-сумісних програм моделювання (VITAL compliant simulators).

VITAL_TIMING. Пакет, що містить типи даних і підпрограми для розроблення часових моделей макрокомірок. Осередок VITAL ASIC відповідає VHDL-інтерфейсу. До пакета також включені підпрограми для вибору затримок, перевірки часових порушень і збоїв. Для перевірки часових порушень

запропоновані процедури VITAL_Timing: VITALSetupHoldCheck, VITALRecoveryRemovalCheck і VITALPeriodPulseCheck.

VITAL_PRIMITIVE. Пакет визначає множину зазвичай використовуваних комбінаційних примітивів. Примітиви запропоновані у формі функцій і паралельних процедур для підтримки поведінкового і структурного стилю створення моделей. Процедурні примітиви підтримують відділення pin-to-pin затримки шляху, також як і виявлення збоїв GlitchOnEvent. Також пакет містить таблиці істинності або станів, що використовуються для опису примітивів.

VITAL_SDFMAP. Правила перетворення SDF даних у значення VHDL Generic-константа для VITAL-моделей. Розробники засобів моделювання використовують цей стандарт для створення інструментів, які автоматично повертають back-annotate-опис – VHDL-модель з включеними часовими параметрами.

VITAL compliant Level 0 / Level 1 Models. Присвячена визначенню стилю для створення VITAL-сумісних моделей.

Семантика пакетів VITAL_Timing і VITAL_Primitives визначена VHDL-описом згідно з IEEE 1076-87 LRM. Опис інтерфейсу наведено в декларації пакета, а функціональність – у тілі пакета (рис. 10.3). Розробники підтримують будь-яку оптимізацію можливостей пакета семантика, якого узгоджується з VHDL-описом.

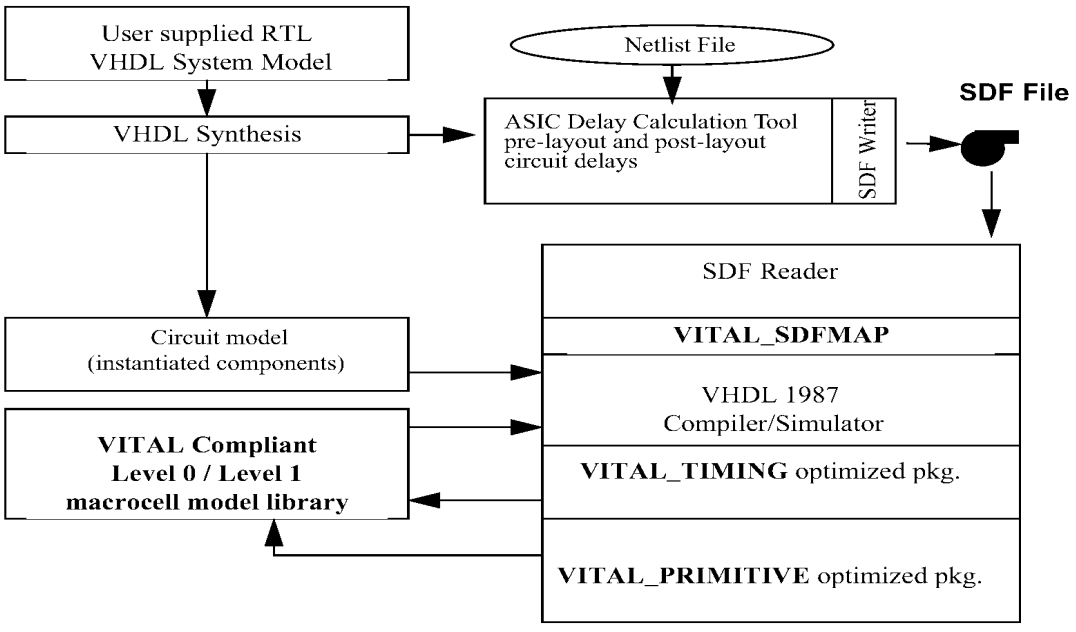


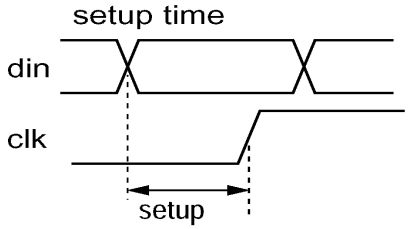
Рис. 10.3. Декларація пакета

10.5. Оцінювання часових параметрів проєктованого пристрою

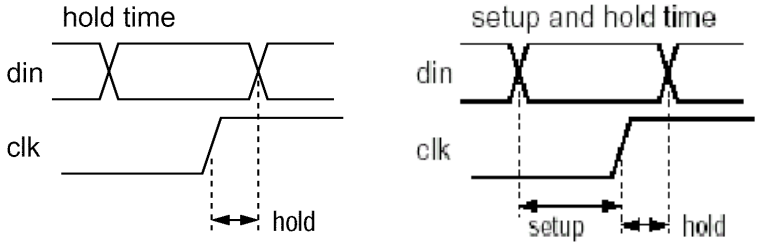
Терміни:

– Propagation Delay (затримка поширення) – час між надходженням вхідного сигналу і появою відповідного входу;

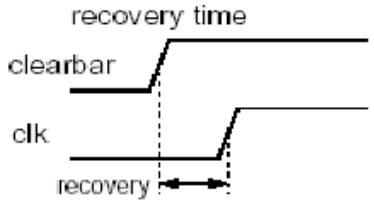
– Setup Time (час установлення) – період часу до появи фронту синхросигналу, протягом якого описаний вхідний сигнал не має змінювати значення;



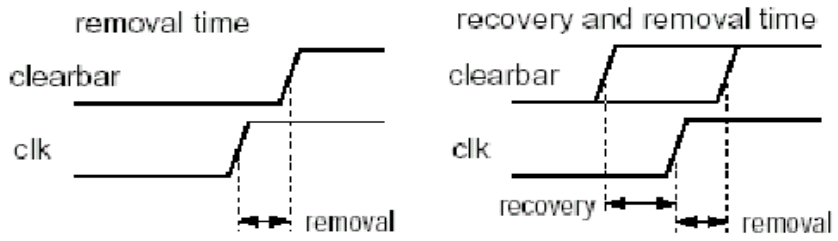
– Hold Time (час зберігання) – період часу, наступний за фронтом синхроімпульсу, протягом якого вхідний сигнал не може змінювати значення;



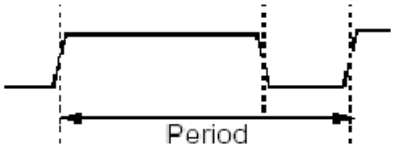
– Release Time (час витримки) – мінімальний проміжок часу між перемиканням асинхронного (set, reset) вхідного сигналу в неактивний стан і появою синхрофронту;



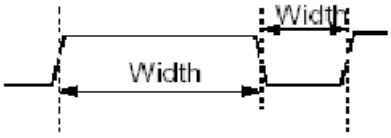
– Removal Time – мінімальний проміжок часу між появою синхрофронту і установленням асинхронного (set, reset) вхідного сигналу в активне значення;



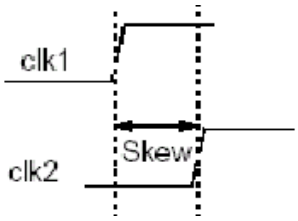
– Period – час між активними фронтами синхросигналу;



– Width – мінімальна ширина імпульсу;

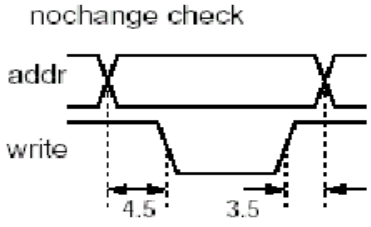


– Skew – зсув між парою сигналів;



– Nochange – описує мінімальний параметр для nochange timing check.

Ця перевірка споріднена перевірці ширини імпульсу. Період setup встановлюється до керуючого імпульсу, а hold після. Сигнал, що перевірявся відносно керуючого сигналу, має залишатися стабільним протягом setup period, ширини імпульсу і hold period. Зазвичай використовується в часових моделях пам'яті, у яких адресна лінія не має змінюватися під час появи сигналу записи, плюс додатковий час до і після цього сигналу.



Interconnect path delay – зовнішня затримка лінії до порту, або затримка міжз'єднань, спрощена до простої затримки лінії.

Timing generic port specification – описує порт або ім'я копії компонента, до якого належить часовий параметр. Підпорядковується синтаксису ідентифікаторів у VHDL, не допускається тільки використання символу підкреслення.

Timing generic suffix – відповідає комбінації SDF-конструкцій, що являють собою умови або фронти. Для опису цих SDF-суфіксів використовується таке правило:

```
    <SDFSimpleConditionAndOrEdge> ::=
        <ConditionName> | <Edge>|
<ConditionName>_<Edge>
<SDFFullConditionAndOrEdge> ::=
       <ConditionNameEdge>
[ _<SDFSimpleConditionAndOrEdge> ]
       <ConditionName> ::= simple_name
       <Edge> ::= posedge | negedge| 01| 10| 0z
| z1| 1z| z0
       <ConditionNameEdge> ::=
[ <ConditionName>_ ] <Edge>| [ <ConditionName>_
] noedge
```

Condition name – лексичний елемент, задає умову, пов'язану з цією часовою інформацією, і відповідний condition expression з SDF-файла.

Edge ідентифікатор – задає фронт, пов'язаний з даним часовим параметром, і відповідає edge name з файла SDF.

Лістинг 10.1. Фрагмент інтерфейсу компонента тригера X_FF в пакеті VCOMPONENTS

```
component X_FF
-- synopsys translate_off
generic(tpd_SET_O:VitalDelayType01 := (0.100 ns,
0.100 ns);
tpd_RST_O : VitalDelayType01 := (0.100 ns, 0.100
ns);
tpd_CLK_O:VitalDelayType01 := (0.100 ns, 0.100
ns);
tsetup_I_CLK_posedge_posedge  : VitalDelayType :=
0.010 ns;
```

```
tsetup_I_CLK_negedge_posedge  : VitalDelayType :=
0.010 ns;
thold_I_CLK_posedge_posedge:VitalDelayType :=
0.010 ns;
thold_I_CLK_negedge_posedge:VitalDelayType :=
0.010 ns;
tsetup_CE_CLK_posedge_posedge : VitalDelayType
:= 0.010 ns;
tsetup_CE_CLK_negedge_posedge : VitalDelayType
:= 0.010 ns;
thold_CE_CLK_posedge_posedge  : VitalDelayType
:= 0.010 ns;
thold_CE_CLK_negedge_posedge  : VitalDelayType
:= 0.010 ns;
          trecovery_RST_CLK_negedge_posedge :
         VitalDelayType := 0.010 ns;
thold_RST_CLK_negedge_posedge : VitalDelayType
:= 0.010 ns;
          trecovery_SET_CLK_negedge_posedge :
         VitalDelayType := 0.010 ns;
thold_SET_CLK_negedge_posedge : VitalDelayType
:= 0.010 ns;
          ticd_CLK:   : VitalDelayType := 0.000
ns;
          tisd_I_CLK  : VitalDelayType := 0.000
ns;
          tisd_CE_CLK   : VitalDelayType :=
0.000 ns;
t         isd_SET_CLK   : VitalDelayType :=
0.000 ns;
          tisd_RST_CLK   : VitalDelayType :=
0.000 ns;
         tpw_CLK_posedge  : VitalDelayType :=
0.010 ns;
         tpw_RST_posedge  : VitalDelayType :=
0.010 ns;
tpw_SET_posedge       : VitalDelayType := 0.010
ns;
tpw_CLK_negedge       : VitalDelayType := 0.010
ns;
tipd_I    : VitalDelayType01 := (0.000 ns,
```

```
0.000 ns);
tipd_CLK : VitalDelayType01 := (0.000 ns, 0.000
ns);
tipd_CE  : VitalDelayType01 := (0.000 ns, 0.000
ns);
tipd_RST : VitalDelayType01 := (0.000 ns, 0.000
ns);
tipd_SET : VitalDelayType01 := (0.000 ns, 0.000
ns) );
-- synopsys translate_on
        port( O    : out   STD_ULOGIC;
          CE     : in   STD_ULOGIC;
          CLK  : in   STD_ULOGIC;
          I    : in   STD_ULOGIC;
          RST  : in   STD_ULOGIC;
          SET  : in   STD_ULOGIC);
        end component;
```

Часові параметри, що формуються Xilinx ISE:

1. Setup/Hold to clock clk показує час, необхідний для установлення і зберігання вхідних сигналів (тобто скільки сигнали мають бути стабільними) до фронту синхросигналу clk.

2. Clock clk to pad показує час, необхідний для формування вихідних сигналів.

3. Clock to Setup on destination clock clk показує час, необхідний між фронтами синхросигналу clk.

4. Pad to Pad.

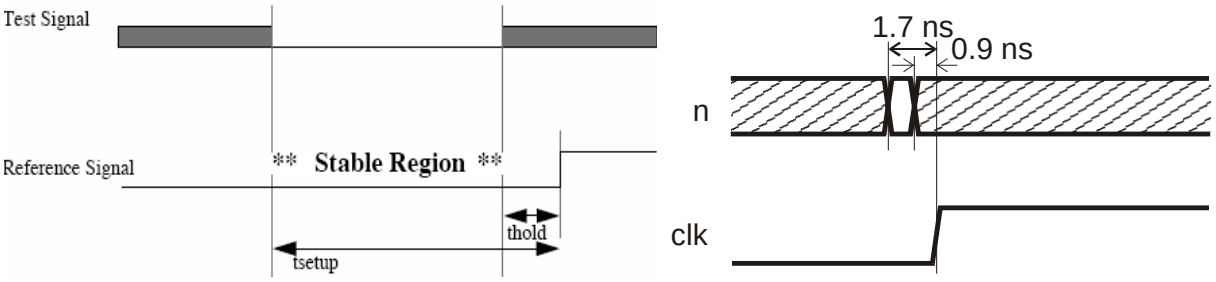
Приклад оцінювання часових параметрів для реального пристрою після імплементації Setup/Hold to clock:

Setup/Hold to clock clk

Source	Setup to clk (edge)	Hold to clk (edge)	Internal Clock(s)	Clock Phase
n	1.660(R)	-0.856(R)	clk_bufgp	0.000
x<0>	1.304(F)	-0.523(F)	clk_bufgp	0.000
x<1>	1.424(F)	-0.643(F)	clk_bufgp	0.000
x<2>	1.569(F)	-0.788(F)	clk_bufgp	0.000
x<3>	1.329(F)	-0.548(F)	clk_bufgp	0.000
y<0>	1.628(F)	-0.847(F)	clk_bufgp	0.000
y<1>	1.469(F)	-0.688(F)	clk_bufgp	0.000
y<2>	1.536(F)	-0.755(F)	clk_bufgp	0.000
y<3>	1.082(F)	-0.301(F)	clk_bufgp	0.000

Setup Time (час установлення) – період часу до появи фронту синхросигналу, протягом якого описаний вхідний сигнал не має змінювати значення. Його максимальне значення 1,660 нс (округлимо до 1,7 нс).

Hold Time (час зберігання) – період часу, наступний за фронтом синхроімпульсу, протягом якого вхідний сигнал не може змінювати значення. Його максимальне значення -0,856 нс (округлимо до -0,9 нс) (від'ємне значення терміну придатності).



Оскільки в нашому прикладі Hold Time від'ємне (-0.9 нс), воно буде входити в проміжок Setup Time (1,7 нс). Приклад Clock to pad:

Clock clk to Pad

Destination	clk (edge) to PAD	Internal Clock(s)	Clock Phase
result<0>	10.343(F)	clk_bufgp	0.000
result<1>	9.907(F)	clk_bufgp	0.000
result<2>	10.386(F)	clk_bufgp	0.000
result<3>	10.099(F)	clk_bufgp	0.000
result<4>	10.702(F)	clk_bufgp	0.000

Максимальний час затримки вихідного сигналу тут 10,702 нс. Саме максимальний час затримки вихідного сигналу враховується для оцінювання періоду (частоти) синхросигналу clk. Приклад Clock to Setup on destination clock:

Clock to Setup on destination clock clk

Source Clock	Src:Rise Dest:Rise	Src:Fall Dest:Rise	Src:Rise Dest:Fall	Src:Fall Dest:Fall
clk	3.797		6.500	4.626

Максимальна вимога до часових параметрів для clk – напівперіод має бути 6,5 нс (час між переднім і заднім фронтом). Тому мінімальний період clk дорівнює 13 нс.

З огляду на те, що максимальне значення Setup Time 1,7, максимальний час затримки вихідного сигналу тут 10,702 і мінімальний період clk дорівнює 13 нс, у цілому для спроєктованого пристрою період мінімальний період clk дорівнює 13 нс.

Контрольні запитання

1. Перелічіть основні етапи проєктування МПС.
2. Назвіть концептуальні рівні опису МПС під час проєктування і розроблення.
3. Перелічіть основні методи контролю правильності проєктування МПС.
4. Якими властивостями має володіти проєктована МПС для виконання етапу її налагодження?
5. Перелічіть види несправності під час проєктування МПС.
6. Назвіть причини фізичної і суб'єктивної несправностей МПС.
7. Поясніть поняття діагностики несправності, налагодження.

ВИСНОВКИ

Знання основ автоматизації проєктування та вміння працювати з засобами автоматизованого проєктування потрібно практично будь-якому інженеру-розробнику. Комп'ютерами насичені проєктні підрозділи, конструкторські бюро та офіси. Сучасні програми автоматизованого проєктування призначені не тільки для створення креслень – вони мають набагато широкі можливості та використовуються практично у всіх галузях техніки. За їх допомогою можна створювати моделі різних конструкцій і перевіряти властивості моделі на комп'ютері до початку виробництва, що істотно знижує кількість помилок проєктування та прискорює появу виробу на ринку.

Сучасний рівень розвитку обчислювальної техніки і засобів передачі інформації відкриває широкі можливості з розроблення і впровадження технічної бази автоматизації. Існуючі ж математичні засоби дають змогу формалізувати основні завдання управління.

Аналіз сучасних розподілених систем на основі застосування мереж показує, що останнім часом істотно зросла роль факторів, які визначаються часом, що витрачається системою на доведення інформації про стан керованого процесу до пунктів на обробку, що надходить, і прийняття рішення на основі прийнятої інформації і доведення прийнятого рішення до виконавчих органів.

Сучасні складні системи характеризуються винятковою складністю умов, у яких здійснюється управління. До них можна віднести значний обсяг завдань, що раптово виникають, жорсткий ліміт часу, відведеного на прийняття (уточнення) рішення щодо їх виконання; велику інтенсивність інформаційних потоків між різними ланками управління; високий динамізм зміни обстановки; обмеженість ресурсу, призначеного для розв’язання задач. Існують об'єкти, у яких час доведення інформації про стан керованого процесу до пунктів становить одиниці секунд. У таких системах час є найважливішим параметром, бо дуже часто потребує формування керуючих дій у реальному часі.

Бібліографічний список

1. Listrovoy S. V. On the class of NP-complete problems and approach Baptism of discrete optimization and graph theory. LAP LAMBERT Academic Publishing GmbH & Co. KG. 2014. 100 p.

2. The comparative analysis of a multiprobe microwave multimeters with involvement of processing by the Kalman filtering and the least-squares methods with regard for Re-reflection of probes / M. A. Miroschnik, O. B. Zaichenko, I. I. Klyuchnik, R. I.Tzekhmistro. *Telecommunications and radio engineering*. 2015. Vol. 74, № 74 (1). P. 79-86.

3. Мірошник М. А., Лістровий С. В., Клименко Л. А. Теорія автоматичного керування, штучний інтелект і автоматизація процесу прийняття рішень: навч. посіб. Харків: ХУПС, 2018. 188 с.

4. Listrovoy S. V., Minykhin S. V. General Approach to Solving Optimization Problems in Distributed Computing Systems and Theory of Intelligence Systems Construction. *Journal of Automation and Information Science.* 2010. Vol. 42, N. 3. P. 30-45.

5. Listrovoy S. V., Butenko V. M. Algoritm of Sub Exponential Complexity for the SAT. *International Journal of Computer and Information Technology* (ISSN: 2279-0764). September 2013. Vol. 2. Is. 05. P. 211-215.

6. Теорія графів у задачах розподілу ресурсів. Кн. 1. Алгоритми та методи обчислень: підручник / Ф. О. Демченко, С. В. Лістровий, М. І. Луханін, Р. В. Семчук. Харків: УкрДАЗТ, 2008. 119 с.

7. Miroschnik M. A., Kotukh V. G., Selevko S. N. Application of software complex for query processing in the database management system with a view of dispatching problem solving in Grid systems. *Telecommunications and radio engineering*. 2013. Vol. 27, № 10. P. 875-891.

8. Listrovoy S. V., Minykhin S. V. Znakhur Investigation of the Scheduler for Heterogeneous Distributed Computing Systems based on Minimal Cover Method. *Internation Journal of Computer Aplication* (0975-8887). August 2012. Vol. 51, No. 19. P. 123–127.

9. Listrovoy S. V. Jn Cjrrelation of P and NP-Classes. *I. J. Modern Education and Computer Science.* 2012. № 3. P. 21-27.

10. Model of influences of sensor reflections on the accuracy of microwave reflectometer / M. Miroschnik, I. Klyuchnyk, R. Tsekhmistro, Z. Warsza, O. Zaichenko. *PAK (Pomiary Automatyka Kontrola).* 2014. Vol. 60. P. 223-225.

11. Pseudoexhaustive tpg based on nonlinear feedback shift registers / M. Berezhna, L. Derbunovsch, M. Ryzhskova, D. Tatarenko. *Інформаційно-керуючі системи на залізничному транспорті*. Харків, 2005. № 5. С. 54-59.

12. Мирошник М. А. Проектирование диагностической инфраструктуры вычислительных систем и устройств на ПЛИС: монография. Харьков: ХУПС, 2012. 188 с.

Навчальний посібник

Мірошник Марина Анатоліївна,
Клименко Любов Анатоліївна,
Корольова Яна Юріївна

Технології та автоматизація проєктування цифрових пристроїв складних комп'ютерних систем на ПЛІС

Відповідальний за випуск Мірошник М. А.

Редактор Ібрагімова Н. В.

Підписано до друку 18.11.19 р.
Формат паперу 60х84 1/16. Папір писальний.
Умовн.-друк. арк. 12,0. Тираж 75. Замовлення №

Видавець та виготовлювач Український державний університет
залізничного транспорту,
61050, Харків-50, майдан Фейєрбаха, 7.
Свідоцтво суб'єкта видавничої справи ДК № 6100 від 21.03.2018 р.