
**УКРАЇНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЗАЛІЗНИЧНОГО ТРАНСПОРТУ**

**М. А. Мірошник, М. С. Курцев**

# АВТОМАТИЗАЦІЯ ПРОЕКТУВАННЯ ВБУДОВАНИХ СИСТЕМ І ПРОГРАМНИХ ЗАСОБІВ НА ПЛІС МОВОЮ ОПИСУ АПАРАТУРИ

***Навчальний посібник***

**Харків 2021**

**УДК 004. 067**
**М 31**

*Рекомендовано вченою радою Українського державного університету залізничного транспорту як навчальний посібник (витяг з протоколу № 4 від 20 квітня 2021 р.)*

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

**М 31** Мірошник М. А., Курцев М. С. Автоматизація проектування вбудованих систем і програмних засобів на ПЛІС мовою опису апаратури: Навч. посібник. – Харків: УкрДУЗТ, 2021. – 331 с., рис. 146, табл. 14.

ISBN

У навчальному посібнику розглядаються архітектурні особливості вбудованих цифрових систем. Наведено теоретичні відомості про системи та мережі на кристалі, вбудовані системи на базі ПЛІС, реконфігуровані системах. Описано сучасні технології проектування та реалізації вбудованих систем. Розглянуто методи синтезу базових компонент вбудованих цифрових пристроїв. Особливу увагу приділено синтезу вбудованих цифрових пристроїв з мікропрограмного управлінням. Подано аспекти впровадження елементів надійності на початкових стадіях проектування цифрових пристроїв, у тому числі і методи захисту цифрового контенту вбудованих систем. Наведено сучасні тенденції в галузі забезпечення захисту цифрових проектів і пристроїв.

Викладено теоретичні основи автоматизації процесу проектування вбудованих комп'ютерних систем, зокрема вбудованих на ПЛІС, а також програмних засобів за допомогою мови опису апаратури VHDL та VeryLog. Велика увага приділяється питанням автоматизації проектування програмних засобів за допомогою спеціалізованих САПР. Розглянуто питання, які виникають у процесі впровадження розроблених систем на залізничному транспорті.

Навчальний посібник призначено для студентів другого рівня навчання – магістр денної та заочної форм, які навчаються за освітньою програмою – «Спеціалізовані комп'ютерні системи галузь знань». Також буде корисний для науковців, фахівців з обчислювальної техніки і аспірантів відповідних спеціальностей.

Навчальний посібник містить спеціальний і додатковий матеріал з дисциплін «Теорія і проектування комп'ютерних систем», «Системи автоматизації проектування пристроїв і систем автоматики», «Проектування цифрових пристроїв на ПЛІС», «Автоматизоване проектування програмних засобів систем ЗАТ», «Апаратне і програмне забезпечення комп'ютерних систем», «Мови опису апаратури», «Вбудовані системи» та 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. 067



ISBN

# ЗМІСТ

## Основні скорочення

**ASIC** – Application Specific Integrated Circuit
**BILBO** – Built-In Logic Block Observer
**BRAM**– lock RAM
**BSR** – Boundary Scan Register
**CAD** – Computer Aided Design
**CDFG** – Control Data Flow Graph
**CLB** – Configurable Logic Block
**CPLD** – Complex Programmable Logic Device
**DCM** – Digital Clock Manager
**DDR** – Double Data Rate
**DRAM** – Dynamic Random Access Memory
**DSP** – Digital Signal Processing
**EDRAM** – Embedded DRAM
**EEPROM** – Electrically Erasable Programmable Read-Only Memory
**EPP** – Extensible Processing Platform
**FPGA** – Field Programmable Gate Array
**FSL** – Fast Simplex Link
**FSM** – Finite State Machine
**ETM** – External Test Mode
**FPU** – Floating Point Unit
**GAL** – Generic Array Logic
**GALS** – Globally Asynchronous Locally Synchronous
**GDS** – Graphic Database System
**GPR** – General Purpose Register
**HDL** – Hardware Description Language
**IOB** – Input/Output Block
**ISA** – Instruction Set Architecture
**ISP** – In-System Programming
**ITM** – Internal Test Mode
**JTAG** – Join Test Action Group
**LFSR** – Linear Feedback Shift Register
**LIFO** – Last In First Out
**LSSD** – Level-Sensitive Scan Design
**LUT** – Look-Up Table
**MAC** – Multiply-ACcumulate

**MIPS** – Million Instructions Per Second
**NoC** – Network on-a Chip
**PISO** – Parallel Input - Serial Output
**PRM** – Programmable Routing Matrix
**PROM** – Programmable Read-Only Memory
**PSoC** – Programmable System-on-a-Chip
**PUF** – Physical Unclonable Function
**RAM** – Random Access Memory
**RISC** – Reduced Instruction Set Computer
**RO-PUF** – Ring Oscillator PUF
**ROM** – Read-Only Memory
**RTL** – Register Transfer Level
**SB** – Switch Block
**SHIC** – Super High Information Content
**SoC** – System on-a Chip
**SRAM** – Static RAM
**TAP** – Test Access Port
**TDI** – Test Data Input
**TDO** – Test Data Output
**TMS** – Test Mode Select
**TOS** – Top Of the Stack
**UCF** – User Constraints File
**VHDL** – VHSIC Hardware Description Language
**VHSIC** – Very High Speed Integrated Circuit

**АЦП** – аналого-цифровий перетворювач
**БВВ** – блок введення/виведення
**БМК** – базовий матричний кристал
**ВС** – вбудовані системи
**ГПТП** – генератор псевдовипадкових тестових послідовностей
**ГСА** – граф-схема алгоритму
**ГТП** – генератор тестових послідовностей
**ДНФ** – диз'юнктивна нормальна форма
**ЕП** – елемент пам'яті
**ЗУ** – запам'ятовуючий пристрій
**ІС** – інтегральна схема

**КМОП** – комплементарний метал-окисел напівпровідник
**МК** – мікроконтролер
**МП** – мікропроцесор
**МПС** – матриця програмованих з'єднань
**МСА** – багатоканальний сигнатурний аналізатор
**ОЗП** – оперативний пристрій
**ПЗП** – постійний запам'ятовуючий пристрій
**ПЛІС** – програмована логічна інтегральна схема
**ПЛМ** – програмована логічна матриця
**ПЛП** – програмований логічний пристрій
**ПМЛ** – програмована матриця логіки
**ПЗ** – програмне забезпечення
**ППЗП** – перепрограмоване ПЗП
**ПСнК** – програмована система на кристалі
**ПСП** – псевдовипадкова послідовність
**РОН** – регістри загального призначення
**САПР** – система автоматизованого проектування
**СБИС** – надвелика інтегральна схема
**СДНФ** – досконала диз'юнктивна нормальна форма
**СнК** – система на кристалі
**СОЗП** – статичний ОЗП
**ФБ** – функціональний блок
**ЦАП** – цифро-аналоговий перетворювач
**ЦОС** – цифрова обробка сигналів

# ПЕРЕДМОВА

У навчальному посібнику викладено методи автоматизованого проектування вбудованих цифрових пристроїв і систем.

Раніше вбудованою цифровою системою прийнято було називати спеціалізовані пристрої управління, спроектовані на логічних інтегральних схемах малого і середнього ступеня інтеграції, а сучасне розуміння вбудованої системи визначає її як замовний НВІС, що за своєю суттю є спеціалізованою обчислювальною системою, технологічно виготовленою на одному кристалі.

Проектування сучасних вбудованих систем засноване на застосуванні високотехнологічних САПР цифрових пристроїв, що вимагає від розробників глибоких знань не тільки цифрової схемотехніки і архітектур обчислювальних систем, але і знання методів синтезу спеціалізованих пристроїв з мікропрограмним управлінням, високорівневих мов проектування і методів контролепридатного синтезу. Процес проектування сучасних вбудованих цифрових пристроїв і систем – це процес створення власних і використання стандартних цифрових компонент інтелектуальної власності, що являють собою не тільки схемотехнічні описи, але по суті є повноцінними проектними документаціями за функціональним і параметричним моделюванням, верифікацією та виготовленням з застосуванням конкретних технологій. Масове поширення мов проектування цифрових пристроїв, формування світового ринку компонент інтелектуальної власності визначило проблему захисту проектних описів і реалізованих цифрових компонент від несанкціонованого використання. Стрімкий прогрес у технологіях реалізації програмованих логічних інтегральних схем не тільки відновив інтерес інженерів і науковців до реконфігурованих обчислень, але і визначив нові завдання, такі як забезпечення захисту цифрових компонент від клонування.

Навчальний посібник присвячений сучасним методам проектування вбудованих цифрових пристроїв і систем з застосуванням високорівневих мов проектування. Основну увагу приділено методам синтезу базових цифрових пристроїв і спеціалізованих пристроїв з мікропрограмного управління,

контролепридатного синтезу і захисту цифрового контенту вбудованих систем. Розглядаються питання, пов'язані з методами і засобами захисту цифрових проектів і реалізованих компонент вбудованих систем.

Змістовна частина включає як класичні питання високорівневого проектування цифрових вбудованих систем, так і оригінальні результати.

Перший розділ присвячено сучасним архітектурам вбудованих систем. Зроблено огляд систем на кристалі, реконфігурованих цифрових систем, мереж на кристалі. Наводяться приклади вбудованих систем на базі ПЛІС.

Другий і третій розділи присвячені синтезу базових компонент вбудованих цифрових пристроїв і цифрових пристроїв з мікропрограмного управління. Наводяться приклади вихідних описів цифрових компонент мовою VHDL і результати їх синтезу для ПЛІС.

У четвертому розділі розглядаються питання контроле-придатного синтезу цифрових пристроїв, зокрема методи контролепридатного доступу. Наводяться приклади реалізації архітектури граничного сканування для цифрових пристроїв, що вбудовуються.

П'ятий розділ містить матеріал, присвячений методам, алгоритмам і засобам захисту цифрових проектів і пристроїв від несанкціонованого використання. Увага приділена методам захисту, заснованим на застосуванні фізично неклонованих функцій. Наводяться приклади захисту цифрових пристроїв, реалізованих на ПЛІС.

Навчальний посібник призначено для студентів денної та заочної форм навчання для освітніх програм першого та другого рівнів, які навчаються за спеціальністостями 123 «Комп'ютерна інженерія», 126 «Інформаційні системи і технології», 151 «Автоматизація та комп’ютерно-інтегровані технології», 273 «Залізничний транспорт», а також для інших спеціальностей відповідних напрямів і фахівців у галузі комп'ютерних систем і програмування.

Навчальний посібник містить спеціальний і додатковий матеріал з дисциплін «Теорія і проектування комп’ютерних систем», «Системи автоматизації проектування пристроїв і

систем автоматики», «Проектування цифрових пристроїв на ПЛІС», «Автоматизоване проектування програмних засобів систем ЗАТ», «Апаратне і програмне забезпечення комп'ютерних систем», «Мови опису апаратури», «Вбудовані системи» і може застосовуватися при виконанні курсових і розрахунково-графічних робіт.

Кожний розділ має приклади практичних завдань та контрольні питання.

# 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 за 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 найрізноманітніших сферах діяльності людей. Дивною властивістю цифрового комп'ютера є його універсальність, тобто застосовність до всього. Він може слідувати послідовності команд – програмі, що 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 – 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), бінарних (двійкових). Проектувальник 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 спроектувати 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 дискретизированими. Наприклад, коли змінні процесу подаються безперервними сигналами в реальному масштабі часу, вони дикретизуються аналого-цифровим перетворювачем. Фізична система, чия поведінка описується математичними рівняннями, моделюється в *цифровому комп'ютері* шляхом використання чисельних методів. А коли розв'язувана 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 проектування цифрових схем від простих комбінаційних вузлів до цифрових автоматів. Причому будуть розглядатися і класичні підходи, які дозволяли виконувати проектування цифрових схем вручну, і сучасні методи, що дозволяють за допомогою засобів автоматизованого проектування позбавити проектувальника рутинної «ручної праці».

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

Мета опанування дисципліни - набуття базових знань з організації процесу розроблення, програмування, тестування і налагодження програмних продуктів вбудованих інформаційно-комунікаційних систем з використанням сучасних технологій і підходів.

Завданням дисципліни є вивчення принципів побудови мікропроцесорних систем різного призначення та їхніх основних функціональних вузлів, набуття навичок проведення оптимального вибору технічних рішень залежно від технічної задачі та умов використання і експлуатації вбудованої системи (ВС).

Перелік компетенцій, яких набуде студент після опанування даної дисципліни: аналізувати, теоретично та експериментально досліджувати методи, алгоритми, програми апаратно-програмних комплексів і систем; аналізувати і обирати обчислювальні методи для вирішення завдань проектування вбудованих систем за критеріями мінімізації обчислювальних витрат, стійкості, складності; використовувати можливості локальних мереж та Інтернет-технології в системах проектування; реалізовувати і тестувати компоненти програмного забезпечення ВС.

Сфера реалізації набутих компетенцій у майбутній професії: використання сучасних програмних засобів для моделювання, дослідження та створення вбудованих систем; створення програм мовами програмування високого рівня для побудови та використання моделей сучасних вбудованих систем; використання сучасних засобів автоматизації проектування для вирішення різних завдань.

Навчальний посібник з дисципліни відповідає програмі дисципліни і методиці викладання дисципліни, яка склалася за останні роки на кафедрі спеціалізованих комп’ютерних систем Українського державного університету залізничного транспорту.

Тематика навчального посібника викладається згідно з затвердженим навчальним планом, що враховує паралельне проведення практичних занять, лабораторних робіт. Кожна тема виділена в окремий розділ. Розділи поділені на підрозділи, у яких наведено окремі питання. На початку кожного розділу приведено перелік основних питань тематики розділу. Кожний розділ закінчується переліком контрольних завдань і питань, самостійне опрацювання яких і знаходження відповідей сприяють більш глибокому засвоєнню матеріалу.

За сучасними вимогами навчання, половина часу, який виділяється на вивчення дисципліни, відводиться на самостійну роботу студента. Розділи навчального посібника дано з зазначенням орієнтовного обсягу годин для вивчення матеріалу в аудиторії і годин для засвоєння матеріалу, що виноситься на самостійне опрацювання. Таким чином, навчальний посібник може бути використаний для самостійного вивчення курсу.

Обсяг матеріалу посібника достатній для того, щоб при його засвоєнні не вдаватися до допоміжної або довідкової літератури. При підготовці курсу лекцій враховано, що студенти отримали необхідні знання з загальнотехнічних і спеціально-технічних дисциплін.

У розділі 1 зроблено огляд вбудованих систем.

У розділі 2 доступно викладено технології та автоматизація проектування за допомогою комбінаційною логіки.

У розділі 3 розглянуто технології та автоматизацію проектування базових компонент вбудованих пристроїв.

У розділі 4 розглянуто технології та автоматизацію проектування запам'ятовуючих пристроїв вбудованих систем.

У розділі 5 розглянуто етапи проектування вбудованих систем.

Кожний розділ має приклади практичних завдань і контрольні питання.

# 1. ОГЛЯД ВБУДОВАНИХ СИСТЕМ

У цьому розділі розглядається сучасне розуміння вбудованих систем як цифрових компонент систем на кристалі і методи їх проектування. Особливу увагу приділено проектним описам, які розглядаються як компоненти інтелектуальної власності. Детально вивчається питання реалізації вбудованих систем на прикладі програмованих логічних інтегральних схем, які є технологічним базисом для реконфігурованих цифрових систем. Наводиться огляд технологій проектування вбудованих систем з використанням високорівневих мовних описів і систем автоматизованого проектування.

## 1.1. Системи на одному кристалі

Традиційно під *вбудованою системою* розуміли і розуміють спеціалізовану обчислювальну систему, яка є невід'ємною частиною більшої системи або пристрою, якими вона управляє. Сфера застосування вбудованих систем сьогодні настільки широка, що охоплює практично всі сторони людського життя: від мобільних мультимедійних пристроїв до медичних чіпів, що імплантуються, та електронних паспортів [1].

З виникненням вбудовані системи проектувалися як спеціалізовані цифрові пристрої на базі інтегральних схем малого і середнього ступеня інтеграції. З появою мікроконтролерної і мікропроцесорної техніки, а пізніше програмованих логічних інтегральних схем (ПЛІС), поняття вбудованої системи сильно трансформувалося. Так, якщо мікропроцесорна вбудована система являє собою спеціалізовану систему, технологічно виготовлену на одній друкованій платі, яка має у своєму складі центральний процесорний блок (мікропроцесор), окремі інтегральні схеми контролерів периферійного обладнання, цифрових запам’ятовуючих пристроїв, то сучасна вбудована система може бути єдиною інтегральною схемою, що включає до свого складу всі перераховані вище компоненти. В англомовній літературі термін *System-On-a-Chip(SoC)* точніше передає тенденцію розвитку сучасних вбудованих систем [2, 3].

Прагнення розробників вбудованих систем інтегрувати всі цифрові компоненти на одному кристалі зумовлено багатьма чинниками. З одного боку, цьому сприяло збільшення ринку замовних інтегральних схем (ASIC, Application Specific Integrated Circuit), складених з великої кількості неоднорідних компонентів, що мають складну структуру і від яких було потрібно ефективне вирішення поставлених завдань. При цьому важливе значення мав час проектування і швидкість виходу готової системи на ринок. З іншого боку, динамічно розширюваний ринок замовних НВІС значно знизив їхню вартість, а споживачам стало вигідно замовляти невеликі партії вузькоспеціалізованих вбудованих систем. При моральному старінні вбудованої системи вже не йдеться про її часткову модернізацію. Простіше і вигідніше стає перепроектувати всю систему в цілому з подальшою появою нових цифрових пристроїв. Прикладом може служити ринок портативних цифрових пристроїв: смартфонів, електронних планшетів, мобільного мультимедійного обладнання тощо.

Таким чином, сучасне тлумачення замовних НВІС відображує і поняття *системи на кристалі* (SoC), під якою розуміють функціонально закінчену цифрову систему, компоненти якої технологічно розташовані на одному кристалі інтегральної схеми.

Розглянемо приклад типової сучасної системи на кристалі. На рис. 1.1 зображена внутрішня структура SoC сімейства ARM [3]. Можемо бачити, що структура поданої SoC складається з процесорного ядра, вбудованих блоків пам'яті, множини системних контролерів і схем управління, контролерів введення/виведення різних інтерфейсів і призначень, об'єднаних між собою периферійною та системною шинами.

Крім стандартних функціональних блоків, ця SoC містить блок замовної логіки (Application-Specific Logic), яка в різних варіантах може доповнювати функціональність стандартних компонент або реалізовувати специфічні апаратні функції системи.

Крім перерахованих функціональних блоків, системи на кристалі мають додаткові апаратні ресурси, що дозволяють забезпечувати доступ до внутрішніх компонентів з метою їх тестування і програмування блоків енергонезалежної пам'яті (компоненти JTAG Boundary Scan і Flash Programmer на рис. 1.1) [1].

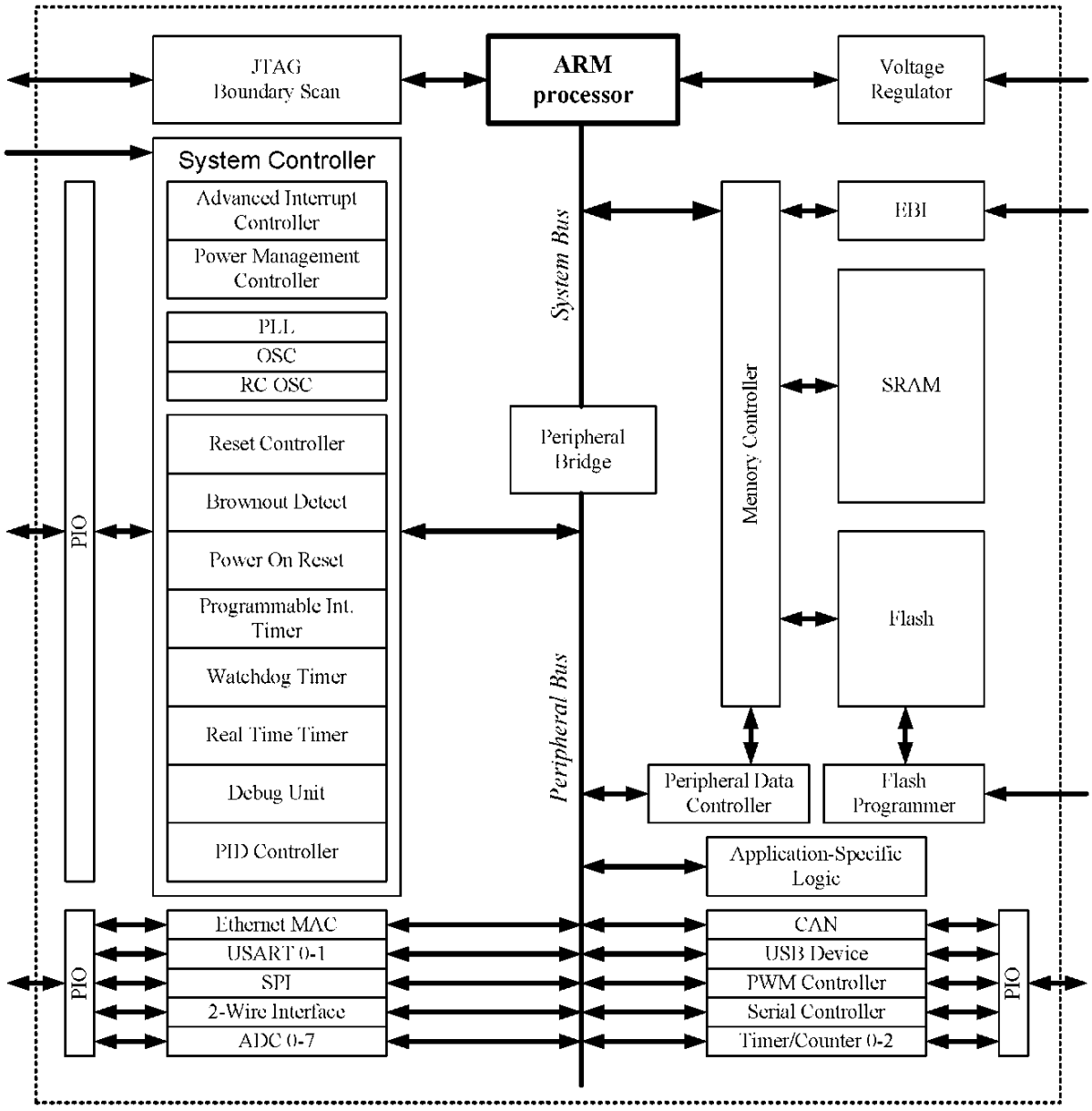



Рис. 1.1. Структура системи на кристалі сімейства ARM

Інтеграція великої кількості різнорідних функціональних блоків на одному напівпровідниковому кристалі визначила широкий спектр методик проектування і розроблення SoC, які, на перший погляд, здаються суперечливими. З точки зору споживача, система на кристалі має відповідати таким вимогам:

1) *висока швидкодія,* що має на увазі ефективне виконання різних призначених для користувача завдань;

2) *низьке енергоспоживання,* обумовлене використанням SoC у портативних пристроях, що працюють від автономних малопотужних джерел живлення;

3) *надійність,* користувач має бути переконаний у правильному функціонуванні цифрового пристрою;

4) *експлуатаційна гнучкість,* яка розуміє оперативну зміну та/або доповнення функціональних можливостей цифрового пристрою самим користувачем;

5) *великий ступінь інтеграції*, що визначає не тільки кількість транзисторів на одиниці площі кристала, але і може відображувати важливі характеристики цифрового пристрою і непрямо його габарити;

6) *прийнятна вартість*, що забезпечує ефективний продаж конкретної SoC на споживчому ринку.

Прагнення задовольнити всі перераховані вище вимоги користувачів до систем на кристалі забезпечується технологіями виготовлення та архітектурним рішеннями при їх проектуванні. Розглянемо як приклад можливі варіанти підвищення швидкодії системи на кристалі. На рис. 1.2 подано технологічні та аппаратно-програмні варіанти підвищення швидкодії мікропроцесорної системи на кристалі (МП SoC) [1].


Рис. 1.2. Варіанти підвищення швидкодії МП SoC

Удосконалення технологічних процесів виготовлення інтегральних схем тягне за собою зменшення топологічних розмірів транзисторів до декількох десятків нанометрів, що дозволяє в комплексі забезпечувати високі швидкості їх перемикання при менших енерговитратах, що опосередковано збільшує швидкодію системи на кристалі в цілому.

Ще одним поширеним варіантом технологічного підвищення швидкодії є збільшення системної частоти функціонування, під якою розуміють не тільки частоту роботи мікропроцесорного ядра, але і частоту передачі даних по системній і периферійній шинах. Однак штучне підвищення цих частот тягне за собою негативні наслідки, у першу чергу пов'язані зі збільшенням напруги живлення, температурою кристала і необхідністю примусового відведення тепла, що може бути конструктивно неможливим для портативних цифрових систем.

Швидкодія МП SoC сильно залежить від пропускної спроможності каналів передачі інформації, що зв'язують обчислювальне ядро і модулі запам'ятовуючих пристроїв, що зберігають коди виконуваних інструкцій і оброблювані дані. Для скорочення втрат часу при передачі інформації виробники інтегрують модулі запам'ятовуючих пристроїв безпосередньо в інтегральну схему SoC [4]. Це дозволяє скоротити не тільки протяжність сигнальних ліній, але і забезпечувати функціонування запам'ятовуючих пристроїв на частоті системної шини. Вбудовані модулі пам'яті (Embedded Memory) є не тільки пристроями оперативної пам'яті, а й виконують роль перепрограмованих постійних запам'ятовуючих пристроїв (ППЗП). За даними Semico Research Corp., частина вбудованих блоків пам'яті неухильно зростає [7], і, за прогнозами, до 2022 року становитиме близько 70 % площі напівпровідникових кристалів цифрових систем (рис. 1.3) [1].

Наприклад, система на кристалі Soc, використана для побудови суперкомп'ютера IBM Blue Gene/P, містить чотири мікропроцесорних ядра PowerPC 405 і чотири блоки вбудованої динамічної пам'яті EDRAM (Embedded DRAM) загальною ємністю 8 Мбіт [8]. Дана SoC містить 208 мільйонів транзисторів, 88 мільйонів з яких використані для реалізації вбудованої пам'яті, що становить близько 42 % загальної площі напівпровідникового кристала.

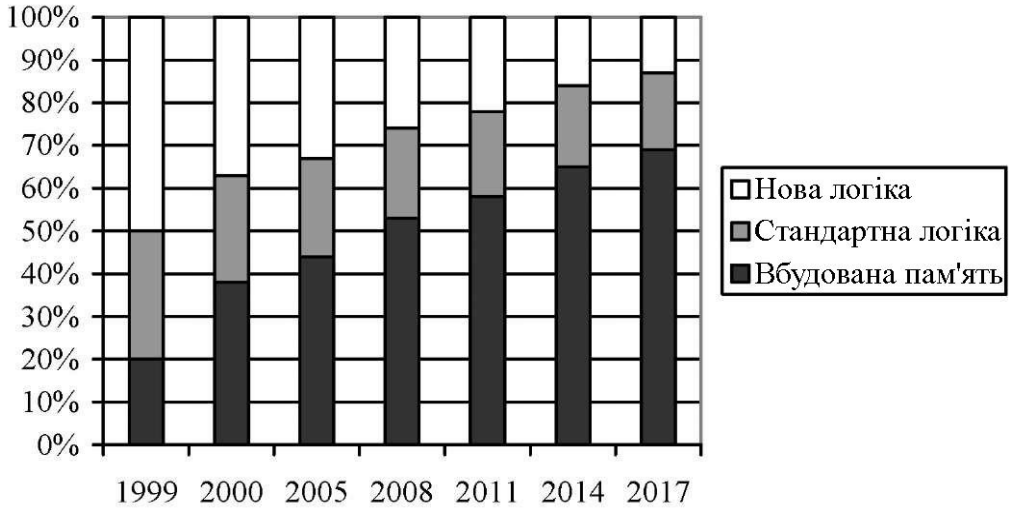



Рис. 1.3. Частина площі SoC, займаної вбудованою пам'яттю

Проектування спеціалізованих SoC змушує розробників ретельно вибирати мікропроцесорне ядро для ефективного вирішення необхідних завдань. Вибір відповідної обчислювальної архітектури буде прямо впливати на швидкодію всієї системи. Наприклад, для виконання системою на кристалі завдань обробки потокового відео буде потрібна наявність процесора цифрової обробки сигналів (ЦОС), як правило, з такими архітектурними особливостями, як розділена пам'ять кодів виконуваних інструкцій і даних, багатоступеневий конвеєр виконання інструкцій тощо. У свою чергу архітектура набору інструкцій (ISA, Instruction Set Architecture) обраного процесора накладає свої обмеження на реалізацію обчислювальних алгоритмів і швидкодію всієї системи. Вирішення завдань оптимізації потоку виконуваних інструкцій не може бути повністю покладено на компілятори, а більшою мірою визначається проектуванням і реалізацією обчислювальних алгоритмів системним програмістом з урахуванням архітектурних особливостей мікропроцесора.

Вимоги ринку до підвищення продуктивності цифрових пристроїв змушують розробників шукати рішення, які не вимагають тривалого часу для реалізації. Одним з таких рішень є розроблення багатоядерних МП SoC, що містять два і більше мікропроцесори [5]. Наприклад, двоядерна система на кристалі Coretex-A9 MPCore, на відміну від своїх одноядерних попередників, була розроблена для збільшення швидкодії з

можливістю застосування симетричної багатопроцесорної обробки даних [6]. Наявність декількох мікропроцесорних пристроїв на кристалі не тягне за собою багаторазового збільшення швидкодії, а лише дозволяє за допомогою системного програмування здійснювати розподілену обробку даних при вирішенні деяких завдань. При цьому приріст продуктивності безпосередньо залежить від програмної реалізації обчислювальних процесів. Обмежувальним чинником застосування багатоядерних МП SoC було і залишається значне споживання енергії, яке негативно впливає на тривалість функціонування цифрових пристроїв з автономними джерелами живлення.

Наступним варіантом підвищення швидкодії МП SoC можна вважати застосування вбудованих прискорювачів, які дозволяють ефективно втілювати в апаратурі найбільш трудомісткі обчислення, що вимагають значних витрат при їх програмній реалізації. Апаратне прискорення може варіюватися від невеликої функціональної одиниці до великого функціонального блока. Наприклад, для підвищення швидкодії SoC, що реалізує алгоритми цифрової обробки сигналів, використовують вбудовані прискорювачі, що являють собою апаратні блоки акумулючих помножувачів MAC [9], а для ефективного декодування сигналу цифрового телебачення та інших систем мультимедіа проектують спеціалізовані апаратні прискорювачі [10, 11].

Апаратні прискорювачі значно збільшують швидкодію систем на кристалі і, будучи повністю реалізовані в апаратурі, не мають можливості змінювати свою функціональність. Наявність перепрограмованої пам'яті виконуваних інструкцій у мікропроцесорних SoC дозволяє змінювати поведінку цифрової системи, що неможливо для SoC, які являють собою виключно апаратні блоки без можливості мікропрограмування. Зміна функціонування апаратного прискорювача або всієї SoC можлива лише шляхом внесення відповідних змін до проектних описів цифрових пристроїв з подальшим їх перепроектуванням і повторним виготовленням.

Компромісом у таких ситуаціях може служити застосування реконфігурованої цифрової апаратури. Під *реконфігурованістю* будемо розуміти можливість апаратних блоків змінювати свою функціональність і зв'язки між собою. Технологічним базисом

для реконфігурованої апаратури є програмовані логічні інтегральні схеми (ПЛІС), які являють собою, як правило, двомірні матриці конфігурованих логічних блоків, розташованих у налагоджуваному комунікаційному середовищі. Функціональність і комунікацію логічних блоків визначає пам'ять конфігурації ПЛІС. Власне наявність такої пам'яті і дозволяє задавати різну функціональність цифрових пристроїв, реалізованих на ПЛІС. Подібна властивість програмованої логіки вже давно використовується розробниками для швидкого прототипування цифрових пристроїв [12]. А зі зростанням ступеня інтеграції програмованих логічних пристроїв ПЛІС типу FPGA (Field Programmable Gate Array) стали застосовуватися як самостійна платформа для реалізації високопродуктивних систем на кристалі [13]. За своєю суттю реконфігурована апаратура є базисом для реалізації апаратних прискорювачів з можливістю зміни їхньої функціональності [14]. Здатність до реалізації довільних цифрових блоків дає можливість проектувати конфігуровані апаратні прискорювачі, які можуть адаптувати своє функціонування під завдання, що вирішується, і тим самим підвищувати швидкодію як мікропроцесорного ядра, так і всієї системи на кристалі в цілому [15, 16].

Перспективність використання технологій реконфігурованої апаратури для реалізації SoC обумовлюється не тільки можливістю підвищення швидкодії за рахунок реалізації апаратних прискорювачів, а й можливістю якісної зміни функціонування цифрових систем без їх перепроектування. Наявність цифрових блоків, що перебудовуються, дає можливість проектувальникам реалізовувати саморемонтовані системи на кристалі, у яких блок, що відмовив, заміняється шляхом реконфігурації резервного блока [17, 18, 19]. Властивість реконфігурування також дозволяє проектувати цифрові пристрої з контролем динамічної споживаної потужності шляхом оптимізації використання вбудованих цифрових модулів [20, 21].

Сучасні реконфігуровані інтегральні схеми, такі як FPGA, мають у своєму складі багато вбудованих цифрових блоків: процесорні блоки, блоки пам'яті, прискорювачі ЦОС і т. д., що робить їх свого роду програмованими системами на кристалі SoC (Programmable SoC) [22]. Розглянемо більш докладно архітектури сучасних ПSoC.

## 1.2. Програмовані логічні пристрої

Програмовані системи на кристалі належать до класу програмованих логічних пристроїв (ПЛП). На відміну від традиційних цифрових пристроїв, що мають фіксовану структуру апаратних блоків, ПЛП можуть змінювати не тільки поведінку окремих своїх модулів, але і перебудовувати зв'язки між ними. Це можливе завдяки наявності пам'яті конфігурації ПЛП і базового набору реконфігурованих блоків (рис. 1.4) [1].

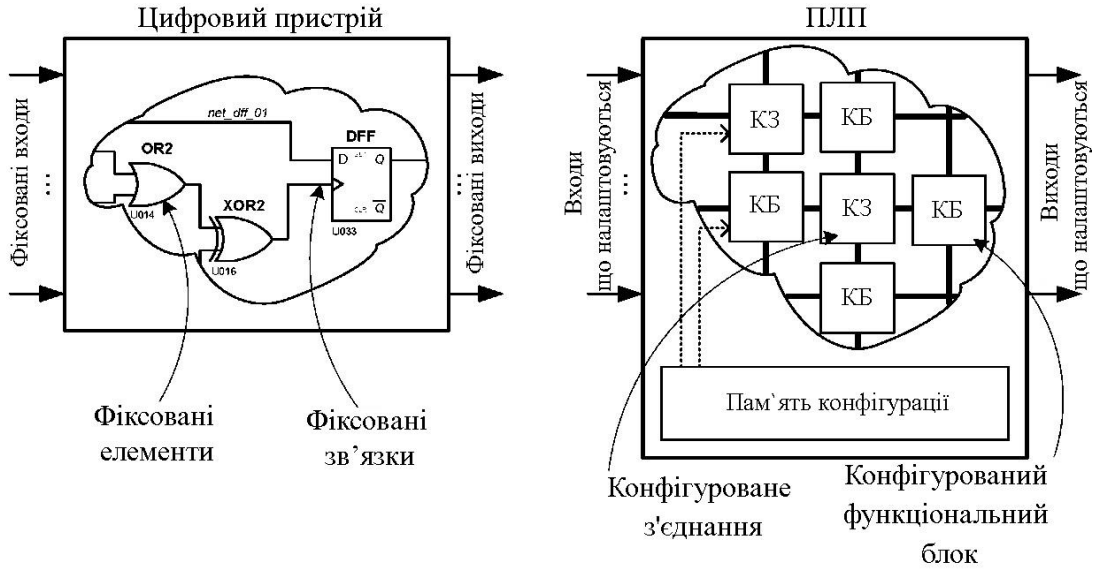



Рис. 1.4. Базове подання цифрових пристроїв з фіксованою (а) і конфігурованою (б) структурами

Довільний цифровий пристрій на логічному рівні ієрархії може бути поданий як схема, що складається зі взаємопов'язаних базових логічних елементів. У свою чергу базові логічні елементи класифікуються так: *комбінаційні логічні елементи* (комбінаційна логіка) і *послідовні логічні елементи* (елементи з пам'яттю). До комбінаційної логіки відносять як найпростіші логічні елементи типу AND2 (двовходовий елемент І), OR2 (двовходовий елемент АБО), так і більш складні функціональні блоки: суматори, мультиплексори, дешифратори і т. д. Послідовна логіка представлена елементами пам'яті, до найпростіших належать тригери і засувки, а до більш складних – регістри, лічильники, генератори кодів, реєстрові файли і т. д.

Функціонування довільної комбінаційної схеми можна описати за допомогою логічного виразу (або системи логічних виразів). Привівши вираз до диз'юнктивної нормальної форми (ДНФ), можна спроектувати схему пристрою в базисі логічних елементів (І, АБО, НІ).

У свою чергу подання логічної функції за допомогою таблиці істинності визначає умови, за яких функція набуває одиничних значень. Ці умови можуть бути записані також у вигляді логічного виразу, відтворенням якого є досконала ДНФ (ДДНФ) вихідної функції [23]. Таким чином, альтернативною реалізацією логічної функції може бути пристрій, на адресні входи якого поступають значення аргументів, на виході формуються значення самої функції, а запам'ятовуючі елементи при цьому виконують роль елементів таблиці істинності, у яких визначені значення початкової функції.

### 1.2.1. Постійні запам’ятовуючі пристрої

Розглянемо реалізацію логічної функції на прикладі комбінаційної схеми мультиплексора, що має два інформаційних входи A і B, вхід селективного сигналу S і вихід F. ДНФ логічної функції F являє собою вираз $F = A \times \bar{S} + B \times S$, для схемної реалізації якого знадобиться два логічних елементи AND2, логічний елемент OR2 і один інвертор INV.

З іншого боку, ДДНФ функції F має такий вигляд: $F = \bar{A} \times B \times S + A \times \bar{B} \times \bar{S} + A \times B \times \bar{S} + A \times B \times S$,, що визначає вхідні набори, на яких функція набуває одиничних значень.

Для наведеного прикладу, перепрограмуючи вміст ПЗП, можлива реалізація $2^8 = 256$ різних булевих функцій трьох змінних. Наприклад, зміна вмісту пам'яті на таке значення *ROM* = {0,1,1,0,1,0,0,1} призведе до реалізації на виході *F* логічної функції *F* = *A* 0 B 0 S, що еквівалентно тривходовому елементу виключного АБО (XOR3).

Припустимо, що є постійний запам'ятовуючий пристрій (ПЗП) з трирозрядною адресною шиною *ADR*, вісьмома запам’ятовуючими елементами і однорозрядним виходом *DO*. Позначимо розряди адреси так: ADR[2] = A, ADR[1] = B та ADR[0] = S. Після цього запрограмуємо пам'ять за правилом: у комірці з адресами ADR = {011,100,110,111} запишемо значення

'1', в інші – "0". Таким чином, вміст пам'яті буде являти собою запис значень функції F з таблиці істинності. Поданий на адресні входи пам'яті двійковий набір буде сприяти появі на виході DO значення функції F згідно з її ДДНФ. На рис. 1.5 наведено два варіанти реалізації логічної функції F за допомогою фіксованої схеми на логічних елементах і постійної пам'яті ROM 8 x 1 [1].

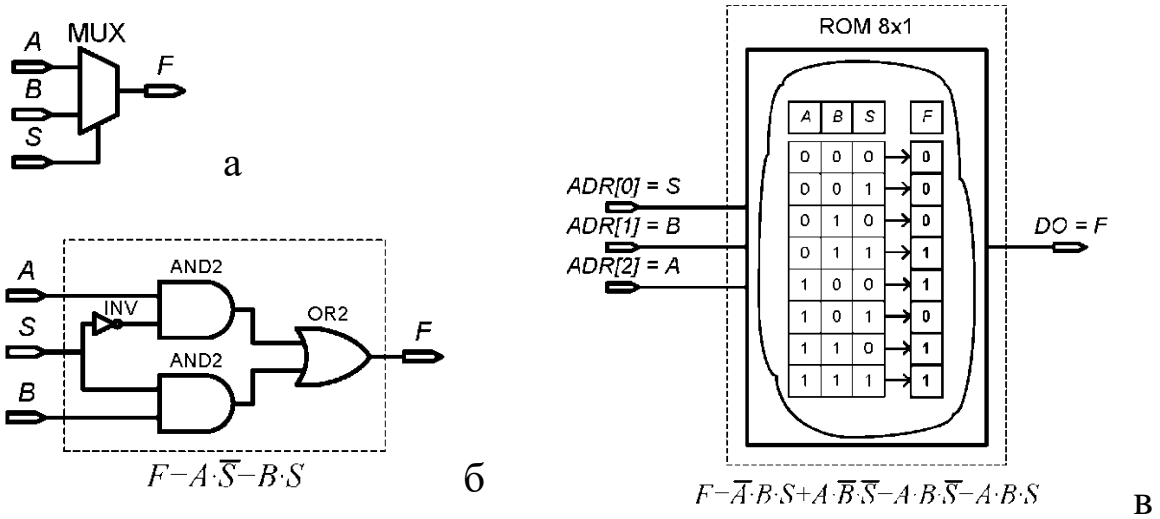



Рис. 1.5. Двовходовий мультиплексор (а), його схемна реалізація на логічних елементах (б) і на ПЗП (в)

У загальному випадку застосування запам'ятовуючих пристроїв з організацією $2^m \times x \times n$ ($m$ – розрядність адресної шини, $n$ – розрядність вихідної шини даних) для реалізації комбінаційної логіки можна схематично зобразити так, як на рис. 1.6.

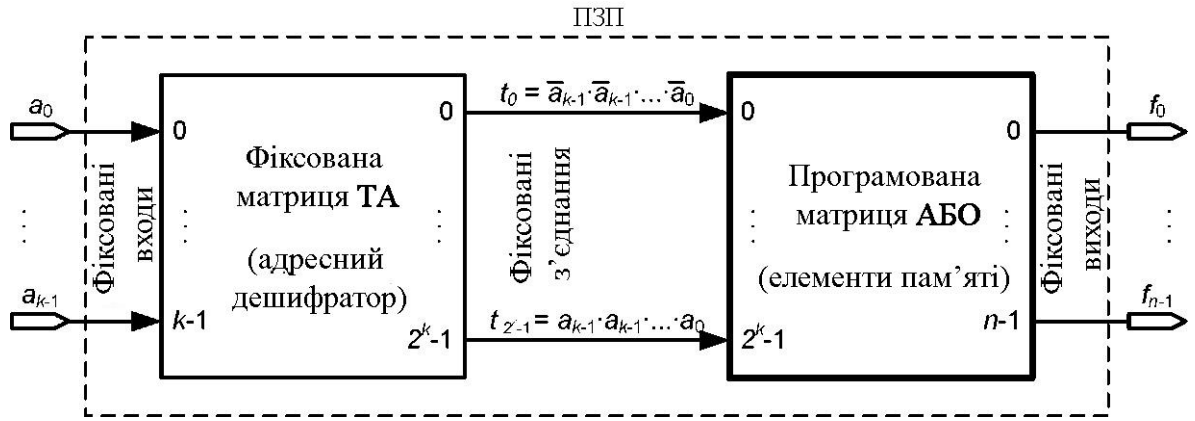



Рис. 1.6. Подання ПЗП для реалізації комбінаційної логіки

Адресний дешифратор пам'яті, що має $k$ фіксованих входів $a_i$, – фіксована матриця, що виробляє всі можливі $2^k$ терми $t_j$. У свою чергу масив запам'ятовуючих елементів пам'яті подається як програмована матриця АБО, що здійснює диз'юнкцію відповідних термів і вироблення n можливих логічних функцій. Число n визначається розрядністю адресованих слів пам'яті. Таким чином, існує можливість реалізувати довільну логічну функцію з $2^k \times n$ можливих від $k$ аргументів.

Ідея використання перепрограмованих ПЗП (EEPROM, Electrically Erasable Programmable Read – Only Memory) як генератори логічних функцій часто застосовувалася в мікроконтролерній техніці [24], а модифікація цієї методики досі використовується в сучасних ПЛП.

### 1.2.2. Програмовані логічні матриці

Подальшим розвитком архітектур ПЛП є клас пристроїв під назвою програмовані логічні матриці (ПЛМ). Основу ПЛМ утворюють дві програмовані матриці кон'юнкторів і диз'юнкторів, здатні реалізувати ДНФ логічних функцій. Окрім того, ПЛМ мають вхідні і вихідні каскади, що містять програмовані інвертори, що дозволяють виробляти інверсні значення вхідних аргументів і вихідні значення функцій. На відміну від запам'ятовуючих пристроїв, для ПЛМ нема необхідності реалізовувати ДДНФ логічних функцій. На виходах програмованої матриці І можлива реалізація термів, що являють собою кон'юнкцію довільного набору з вхідних аргументів і їхніх інверсних значень, що виробляються інверторами вхідного каскаду ПЛМ. Загальна структура ПЛМ подана на рис. 1.7 [1].

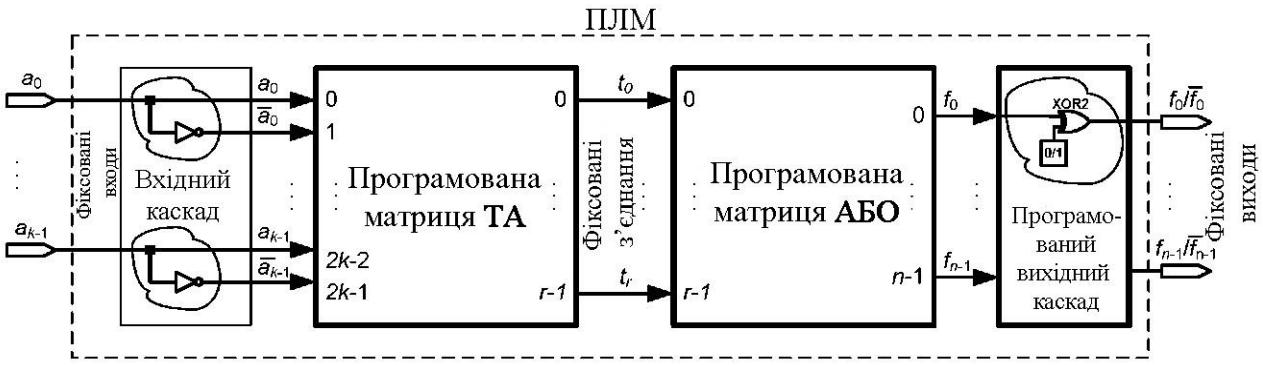



Рис. 1.7. Загальна структура ПЛМ

Крім набору вхідних інверторів, ПЛМ має у своєму складі програмовані інвертори, включені до складу вихідного каскаду для можливості вироблення інверсних значень логічних функцій. Функцію програмованих інверторів виконують двовходові елементи XOR2, на перші входи яких подаються значення функцій, що виробляються, а другі входи є програмованими з можливістю фіксування значення 0 або 1. При фіксуванні значення 0 елемент XOR2 виконує роль повторювача рівня сигналу на своєму першому вході, в іншому випадку – роль інвертора. Подібна властивість програмованості часто використовувалася виробниками ПЛМ [25] і досі використовується в структурах сучасних ПЛП.

### 1.2.3. Програмовані матриці логіки

Подальшим еволюційним етапом розвитку конфігурованих цифрових пристроїв була поява програмованих матриць логіки (ПМЛ), які, на відміну від ПЛМ, мали фіксовану матрицю диз'юнкторів.

Як показала практика застосування ПЛМ, ресурси програмованої матриці АБО неефективно використовувалися при реалізації більшості логічних схем. У зв'язку з цим виробники ПЛП стали заміняти програмовану матрицю АБО на множину фіксованих багатовхідних елементів АБО, що не тільки спрощувало структуру ПЛП, а й полегшувало процес отримання конфігурацій для їх програмування.

Ще однією важливою відмінністю ПМЛ від ПЛМ стало застосування складних вихідних каскадів, які отримали назву *макрокомірки* (macrocell). До складу макрокомірки включалися тристабільні буферні елементи, які могли переводити вихідні лінії ПМЛ у високоімпедансний стан і тим самим реалізовувати функції додаткових вхідних портів, пов'язаних зі входами програмованої матриці І. Крім того, макрокомірка дозволяла реалізовувати синхронну і асинхронну буферизацію вихідних сигналів за допомогою багатофункціональних тригерних елементів, що налагоджуються. Таким чином, наявність зворотних зв'язків і елементів пам'яті в складі ПМЛ дозволили розробникам реалізовувати не тільки комбінаційні, але і послідовні цифрові схеми.

На рис. 1.8 наведена загальна структура програмованих пристроїв типу ПМЛ, а на рис. 1.9 – внутрішня структура макрокомірки вихідного каскаду ПМЛ AMD 22 V 10 [26].

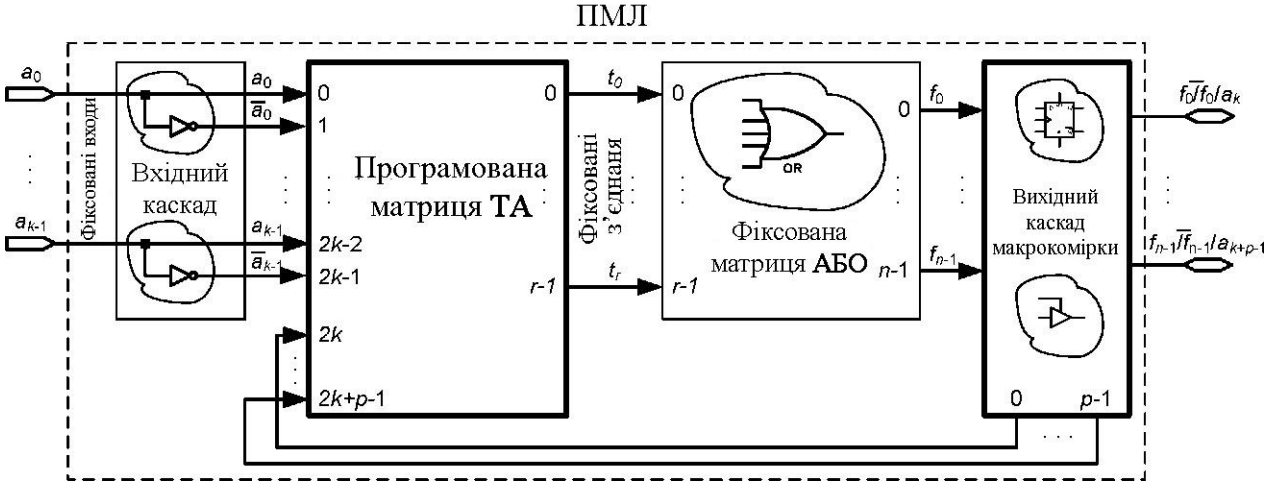



Рис. 1.8. Загальна структура ПМЛ

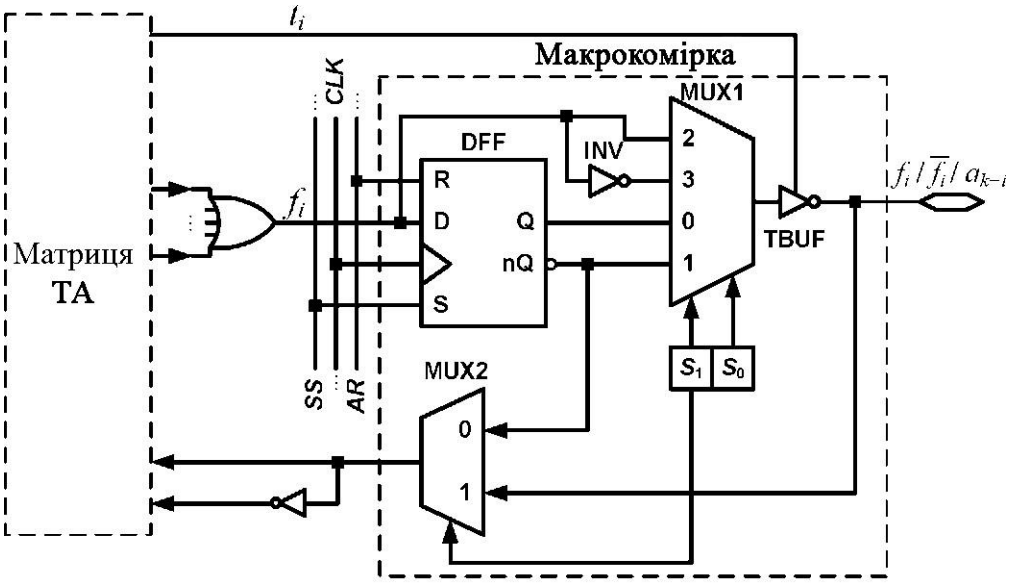



Рис. 1.9. Структура макрокомірки AMD 22V10

До складу макрокомірки входять DFF-D-тригер, стробувальний фронтом глобального сигналу CLK з можливістю синхронного установлення і асинхронного скидання; програмовані мультиплексори MUX1 і MUX2, керовані елементами пам'яті So і Si; INV — інвертор; TBUF — вихідний тристабільний буфер-підсилювач. Управління макрокоміркою

здійснюється за допомогою двох запам'ятовуючих елементів So і Si, значення сигналу $t$ і керуючого буфером TBUF, значень глобальних сигналів синхронізації CLK, синхронного установлення SS і асинхронного скидання *AR*.

Так, при значеннях $t_i = 1$, So = 1 і Si = 1 на вихід макрокомірки транслюватиметься значення функції $f_i$, а її копія буде подана по лініях зворотного зв'язку до матриці I.

При значеннях $t_i = 0$ і $S_1 = 1$ і-й порт ПМЛ буде виконувати роль додаткового входу $a^k + i$ , а при $t_i = 0$ і $S_1 = 0$ макрокомірка виконує роль елемента пам'яті при реалізації послідовної логіки на ПМЛ.

У 1985 році корпорація Laticce Semiconductor удосконалила виготовлення ПМЛ, зробивши матрицю I і макрокомірки багаторазово програмованими, поєднавши на одному кристалі технологію КМОП і EEPROM. Це нововведення дозволило розробникам використовувати ПМЛ для налагодження прототипів майбутніх цифрових пристроїв, а нова технологія ПЛП отримала назву GAL (Generic Array Logic). Крім того, до складу структури GAL були включені апаратні блоки внутрішньосистемного програмування (ISP, In – System Programming), а наявність додаткових портів граничного сканування JTAG (Join Test Action Group), відповідного кабелю, і програмного забезпечення дало можливість перепрограмувати ПЛП, не видаляючи його зі складу цифрової системи [27].

### 1.2.4. Складні ПЛП

На початку 90-х років минулого століття вдосконалення технологій виготовлення напівпровідникових інтегральних схем дозволило розмістити на одному кристалі безліч ПМЛ, об'єднаних програмованими зв'язками. Такі мікросхеми отримали назву «складні ПЛП», або CPLD (Complex Programmable Logic Devices).

Загальна структура CPLD подана на рис. 1.10 [1].

CPLD складається з двомірної матриці функціональних блоків (ФБ), побудованих за принципом перепрограмованого ПМЛ, яка складається з налагоджуваної матриці I та програмованих макрокомірок. У свою чергу кожен функціональний блок з'єднаний з матрицею програмованих з'єднань (МПЗ) і з налагоджуваними блоками введення/виведення (БВВ).

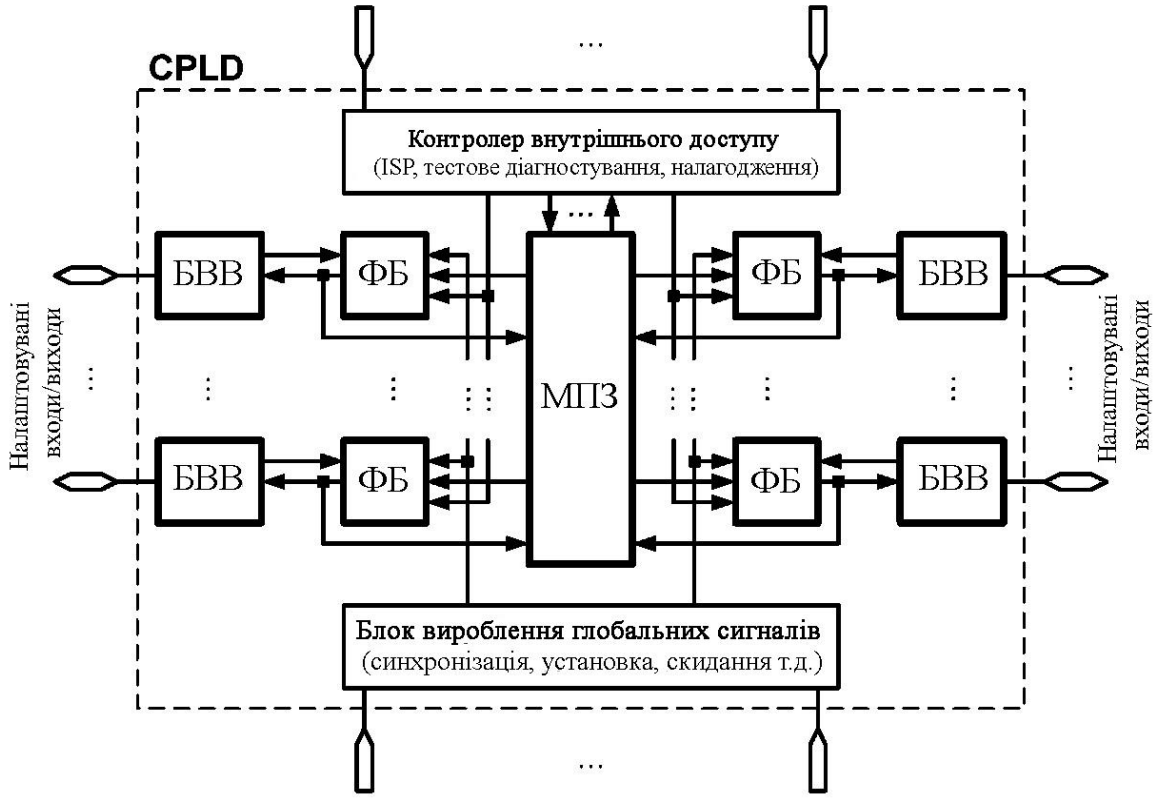



Рис. 1.10. Загальна структура ПЛП типу CPLD

Контролер внутрішнього доступу призначений для проведення процедур внутрішньосистемного програмування CPLD і для забезпечення контролеро-придатного доступу до внутрішніх ресурсів з метою проведення тестування, діагностики та налагодження програмованого пристрою.

Практично всі внутрішні блоки CPLD є програмованими, при цьому зручні налагодження ресурсів забезпечуються енергонезалежною пам'яттю, виконаною, як правило, за Flash-технологією.

Сучасні CPLD являють собою відносно дешеві і багатофункціональні енергонезалежні ПЛІС і застосовуються в основному для реалізації цифрових пристроїв, що не вимагають частої заміни і реконфігурації. Основними сферами застосування CPLD є інтерфейсні контролери, програмовані матриці комутацій цифрових ліній, пристрої управління, арбітри системних і локальних шин мікропроцесорних систем, контролери периферійних пристроїв і т. п.

Сучасні CPLD є досить продуктивними пристроями, що дозволяють реалізовувати складні цифрові системи управління і

навіть користувацькі мікроконтролерні системи. Наприклад, корпорація Xilinx Inc. пропонує готове рішення у вигляді 8-розрядного мікроконтролера PicoBlaze, що налагоджується, для реалізації на CPLD серії CoolRunner [28]. У свою чергу CPLD CoolRunner-II XC2C512 має такі характеристики: 512 макрокомірок, 270 портів введення/виведення, що налагоджуються, системна тактова частота 180 МГц, кристал виготовлений за 0,18-мікронним КМОП техпроцесом, пам'ять конфігурації підтримує близько 1000 циклів стирання/перепрограмування, тривалість зберігання конфігурації – близько 20 років.

### 1.2.5. Програмовані користувачем вентильні матриці

Одночасно з розвитком архітектури ПЛП трансформувалися і удосконалювалися архітектури напівзамовних ІС вентильних матриць з масковим програмуванням. Наприклад, базові матричні кристали (БМК), маючи регулярну структуру, програмувались безпосередньо на етапі виробництва. НВІС БМК являли собою набір базових цифрових блоків, при цьому налагодження зв'язків між ними здійснювалося на етапі трасування сигнальних ліній кристала. Реалізація напівзамовної логіки подібного типу полегшує процес їх виробництва, проте не завжди задовольняє проектувальників з точки зору проектної та експлуатаційної гнучкості розроблюваних цифрових пристроїв. У 1985 році фірма Xilinx Inc. спроектувала і почала серійне виробництво НВІС вентильних матриць, неодноразове програмування яких здійснювалося користувачами, а не на етапі виробництва. Цей вид ПЛП отримав назву FPGA (Field Programmable Gate Array), що в перекладі означає «програмована користувачем вентильна матриця». Поява і подальший розвиток технології FPGA забезпечило великий прорив в архітектурних рішеннях сучасних ПЛП і призвело до виникнення таких термінів, як *програмовані системи на кристалі* (ПSoC) і *реконфігуровані обчислення.*

Взагалі структура FPGA являє собою двомірний масив конфігурованих логічних блоків CLB (Configurable Logic Block), оточений по периферії налагодженими блоками введення/виведення IOB (Input/Output Block). Якщо для БМК трасування міжз’єднань здійснюється на стадії виготовлення, то

для кристалів FPGA налагодження з'єднань і конфігурація блоків CLB і IOB здійснюється користувачем. Всі блоки CLB розташовані в полі комутуючої матриці міжз'єднань PRM (Programmable Routing Matrix), яка забезпечує конфігурацію з'єднань різних блоків між собою. Матриця PRM є множиною горизонтальних і вертикальних трасувань каналів, на перетині яких розташовані блоки перемикання SB (Switch Block). Узагальнена структура кристала FPGA подана на рис. 1.11 [1].

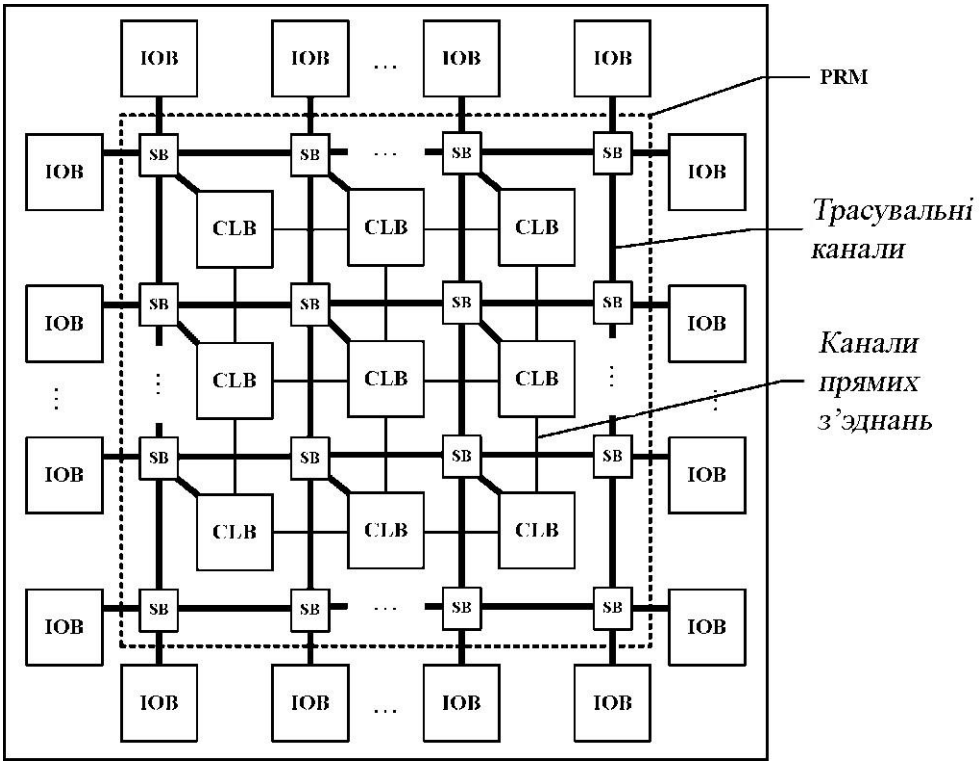



Рис. 1.11. Загальна структура ПЛП типу FPGA

Комутація з'єднань, що налагоджуються, здійснюється за допомогою транзистора або тристабільного буфера, здатних забезпечувати трансляцію сигналів або блокувати. Управління відкриттям або закриттям одного каналу передачі даних здійснюється одним конфігураційним бітом, що є частиною всієї пам'яті конфігурації FPGA. Множина подібних елементів формує блок SB, який дозволяє комутувати різні канали між собою для трансляції сигналів від одних блоків до інших.

Існує три основні типи SB-блоків: універсальний (universal), непересічний (disjoint) і Уілтон (Wilton) [29, 30]. На рис. 1.12

подано приклади SB-блоків різних типів і можливі варіанти комутації каналів [1].

Для кожного типу SB-блока необхідна наявність 30 бітів конфігурації, які дозволяють здійснювати різні види комутацій цифрових каналів. Наприклад, для універсального SB-блока вхідний канал 4L може бути з'єднаний з одним з трьох каналів: 0B, 4R, 4B. З'єднання CLB і IOB блоків між собою може бути здійснене за допомогою налагоджуваних каналів матриці PRM або за допомогою прямих з'єднань, наприклад для організації ліній прискореного перенесення при реалізації схем суматорів/помножувачів.

Крім описаних вище блоків, кристали FPGA можуть містити додаткові компоненти: вбудовані блоки багатопортових ЗП, мікропроцесорні компоненти, апаратні акселератори алгоритмів цифрової обробки сигналів (DSP), інтерфейсні контролери і т. д.

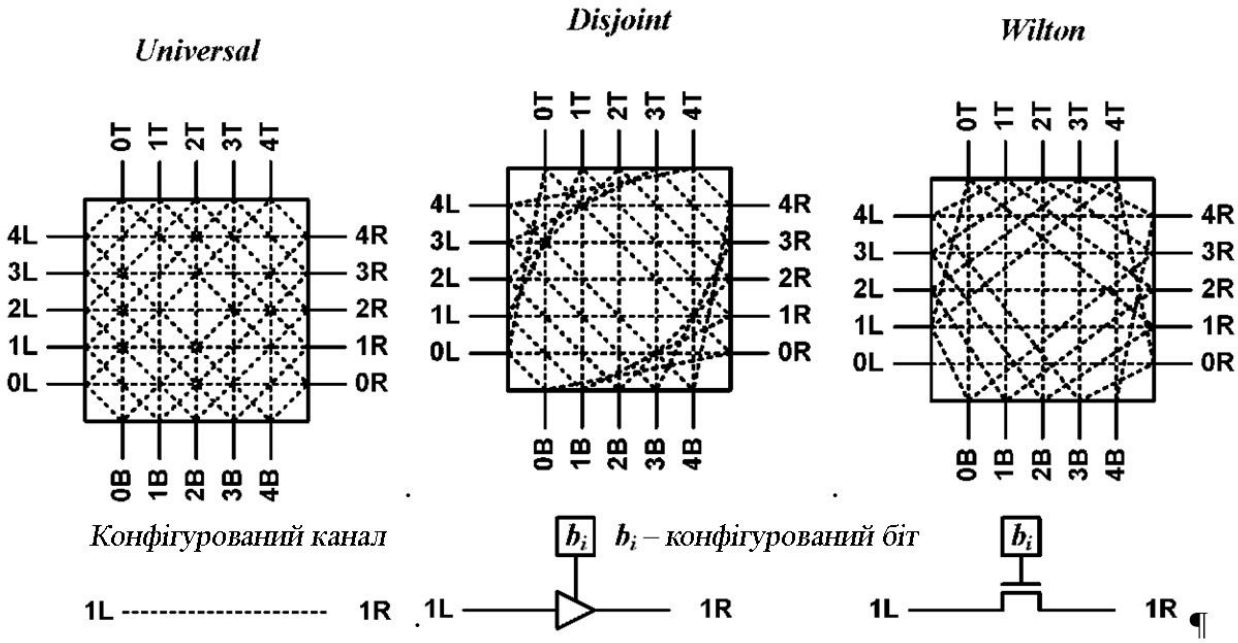



Рис. 1.12. Типи SB-блоків

Крім описаних вище блоків, кристали FPGA можуть містити додаткові компоненти, такі як вбудовані блоки багатопортових ЗП, мікропроцесорні компоненти, апаратні акселератори алгоритмів цифрової обробки сигналів (DSP), інтерфейсні контролери і т. д.

Розглянемо архітектурні особливості кристала FPGA на прикладі серії SPARTAN-3E фірми Xilinx Inc. [31]. Цей тип FPGA виготовляється за технологічним процесом 90 нм SRAM і у своїй максимальній конфігурації має 376 контактів введення/виведення з підтримкою різних номіналів напруги для цифрових сигналів (від 1,2 до 3,3 В). Двомірна матриця CLB-блоків у своїй максимальній конфігурації (кристал XC3S1600E) містить 3688 конфігурованих логічних блоків, 376 блоків IOB, вбудовані блоки статичної пам'яті BRAM (Block RAM) загальною інформаційною ємністю 648 бітів, 36 апаратних блоків конфігурованих помножувачів 18x18 і 8 блоків DCM (Digital Clock Manager), призначених для налагодження сигналів системної синхронізації, а також здійснення множення і ділення частот (у діапазоні від 5 до 333 МГц).

На рис. 1.13 подано внутрішню структуру блока IOB FPGA SPARTAN-3E. Кожен блок введення/виведення з'єднаний з фізичним контактом мікросхеми FPGA (I/O Pin) і може бути налагоджений як вхідний, вихідний або двонаправлений порт [1].

Основу кожного блока IOB складають вхідний буфер-підсилювач IBUF і вихідний тристабільний буфер-підсилювач TOBUF. Управління буферами здійснюється за допомогою трьох програмованих секцій: T-Section (формує керуючий сигнал для буфера TOBUF), O-Section (формує значення вихідного сигналу), I-Section (здійснює захоплення зовнішнього сигналу з контакту I/O Pin для подальшої трансляції до CLB-блоків). Кожна секція містить два елементи пам'яті, які можуть функціонувати як синхронні D-тригери або асинхронні засувки (Latch). Керуючі сигнали асинхронного встановлення/скидання SR та інверсії поточного стану елемента пам'яті REV є загальними для всіх секцій.

Синхронізація елементів пам'яті секцій здійснюється чотирма сигналами: *OTCLK1*, *OTCLK2*, *ICLK1* і *ICLK2*. Сигнали дозволу синхронізації *TCE*, *OCE* та *ICE* є загальними для пари елементів пам'яті в кожній секції.

Розглянемо структуру секції T. Існує п'ять джерел сигналу, керуючого тристабільним буфером TOBUF, які вибираються за допомогою програмованого мультиплексора MUXT: два комбінаційні сигнали, що формуються CLB-блоками та обрані в обхід елементи пам'яті, два сигнали, що формуються на виходах Q елементів пам'яті TFF1 (TTF2), і сигнал на виході блока DDR MUX.

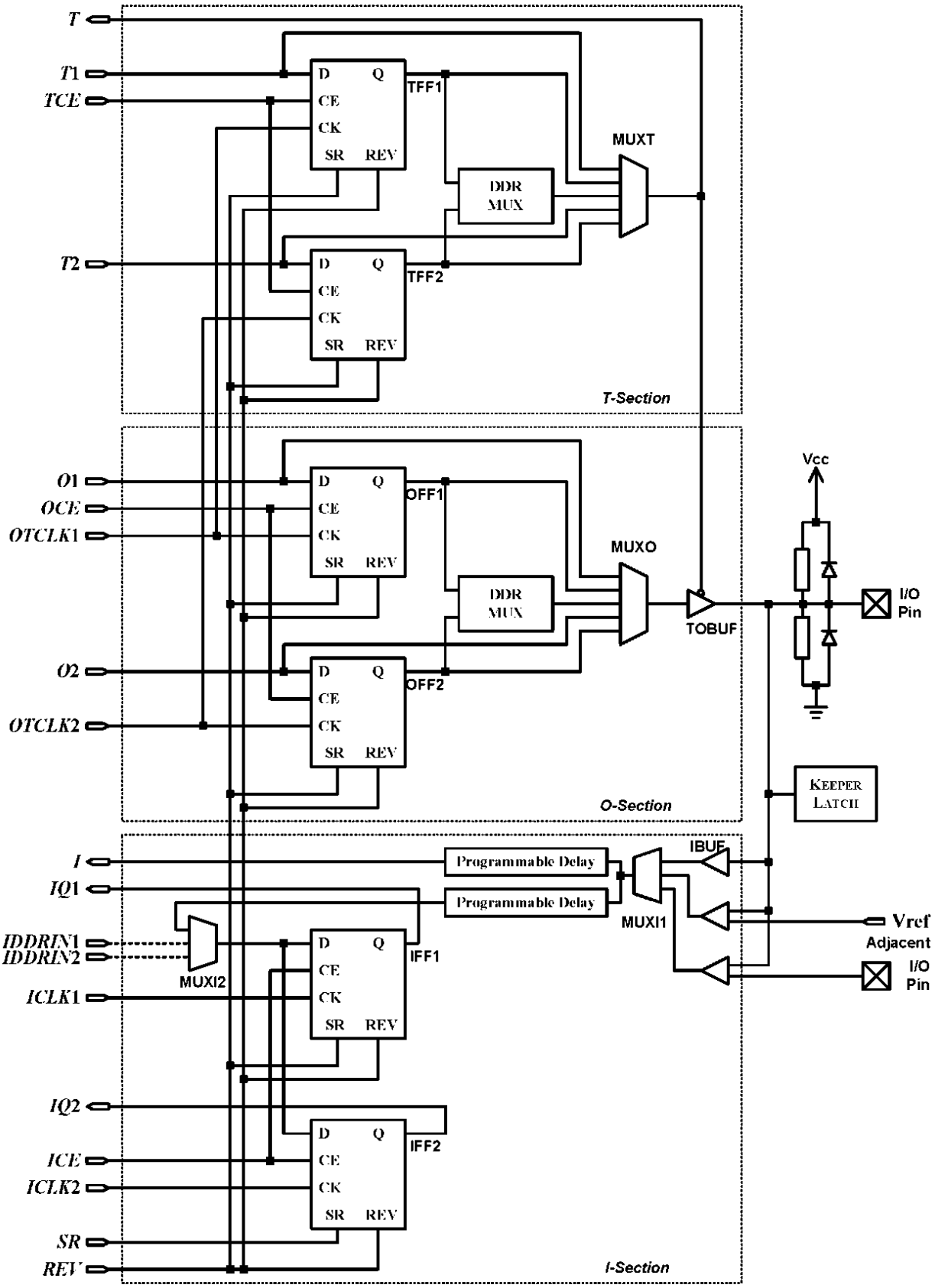



Рис. 1.13. Структура блока
введення/виведення FPGA SPARTAN-3E

У разі реалізації синхронного управління буфером TOBUF передані дані і керуючий сигнал формуються відповідно до одного джерела синхронізації (OTCLK1 або OTCLK2). Блок DDR MUX (DDR, Double Data Rate) призначений для формування значення керуючого сигналу секції T і вихідних даних секції O з подвоєною частотою. Використання блока DDR MUX передбачає два варіанти синхронізації елементів пам'яті. Перший варіант полягає в застосуванні вбудованого блока DCM, за допомогою якого здійснюється зсув фази вихідного сигналу синхронізації на 180°, при цьому вихідний сигнал підключається до входу *CK* першого елемента пам'яті, а другий – до входу *CK* другого елемента пам'яті. Таким чином, формування даних може здійснюватися як під час настання фронту, так і в момент настання спаду сигналу синхронізації. Альтернативним варіантом подвоєння частоти передачі даних є безпосереднє підключення інверсного значення сигналу синхронізації на вхід *CK* другого елемента пам'яті. При цьому DCM не використовується, а інверсія сигналу здійснюється засобами CLB-блоків. Сформовані значення керуючого сигналу і сигналу вихідних даних можуть бути доступні CLB-блокам за допомогою використання комутованих ліній T і I, підключених до виходу програмованого мультиплексора MUXT і виходу елемента Programmable Delay (транзитне підключення через буфер IBUF і мультиплексом MUXI1).

Секція I також містить два елементи пам'яті IFF1 (IFF2), які можуть застосовуватися для захоплення і тимчасового зберігання значення вхідного сигналу з подальшою передачею на лінії 1Q1 і 1Q2. Лінія I служить для безпосередньої передачі значення вхідного сигналу до блоків CLB. Елемент програмованої затримки (Programmable Delay) являє собою множину послідовно з'єднаних буферних елементів (елементів затримки), виходи яких надходять на входи програмованих мультиплексорів. Ставлячи різні конфігурації селективних сигналів для мультиплексорів, користувач може управляти значенням часу попереднього встановлення вхідного сигналу на лінії I та на входах елементів IFF1 і IFF2 (рис. 1.14).

Перейдемо до розгляду структури CLB-блоків. Логічні блоки FPGA SPARTAN-3E призначені для реалізації комбінацій-

ної і послідовної логіки. Кожен CLB-блок складається з чотирьох секцій (slices), згрупованих так, як показано на рис. 1.15 [1].

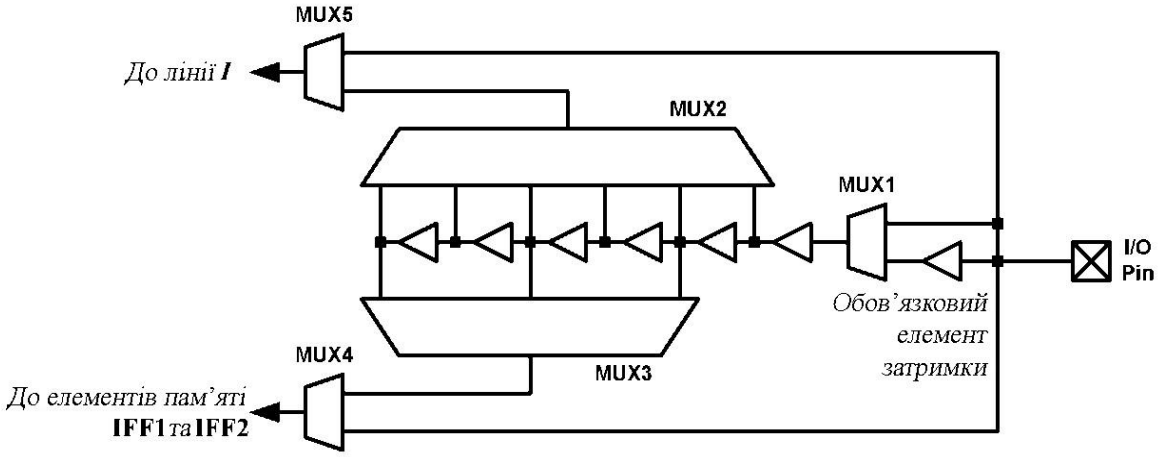



Рис. 1.14. Структура елемента програмованої затримки

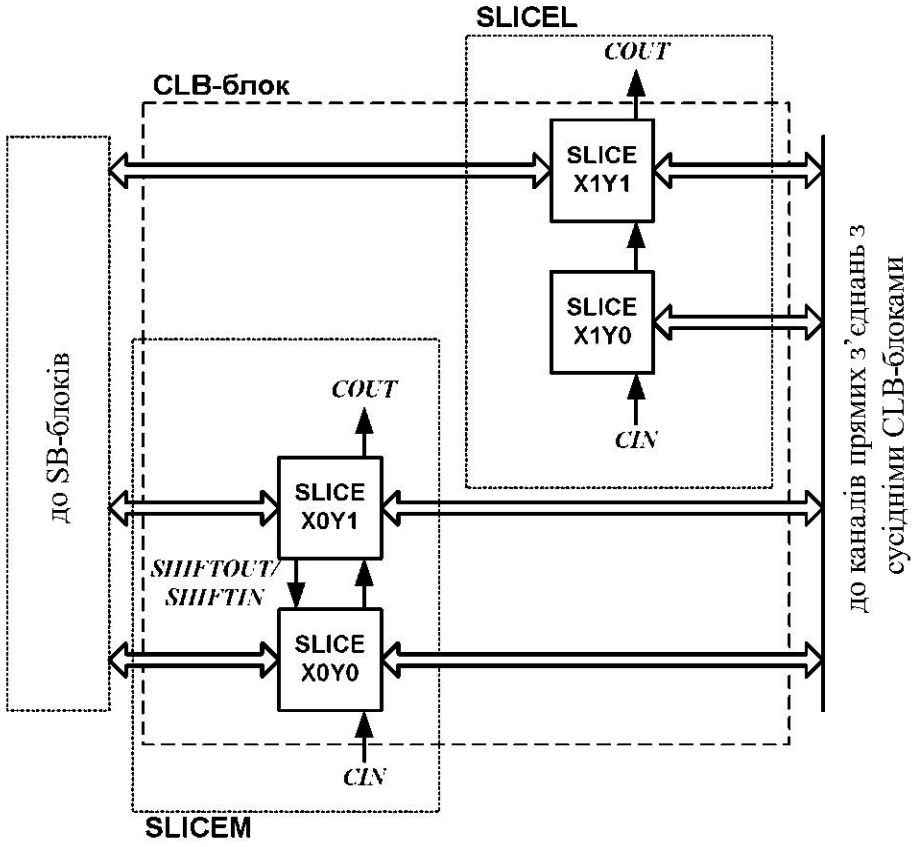



Рис. 1.15. Структура CLB-блока

Кожна з чотирьох секцій X0Y0, X0Y1, X1Y0 і X1Y1 призначена для реалізації комбінаційної логіки і найпростіших

елементів пам'яті, а пара частин X0Y0 і X0Y1 здатна реалізовувати елементи розподіленої пам'яті (Distributed RAM) і зсувні регістри. У зв'язку з цим пара секцій X0Y0 і X0Y1 носить назву SLICEM (Memory), а пара секцій X1Y0 і X1Y1 – SLICEL (Logic).

У свою чергу кожна секція містить два генератори функцій LUT (Look-Up Table), здатних реалізовувати довільну перемикальну функцію чотирьох змінних, два універсальних елементи пам'яті, які можуть бути використані як синхронні тригери або асинхронні засувки, фіксовані логічні блоки для можливості реалізації схем прискореного перенесення і програмовані мультиплексори для можливості конфігурації сигнальних каналів у межах використовуваної секції. На рис. 1.16 зображено можливі конфігурації LUT-блоків, мультиплексорів і елементів пам'яті для однієї секції CLB-блока [1].

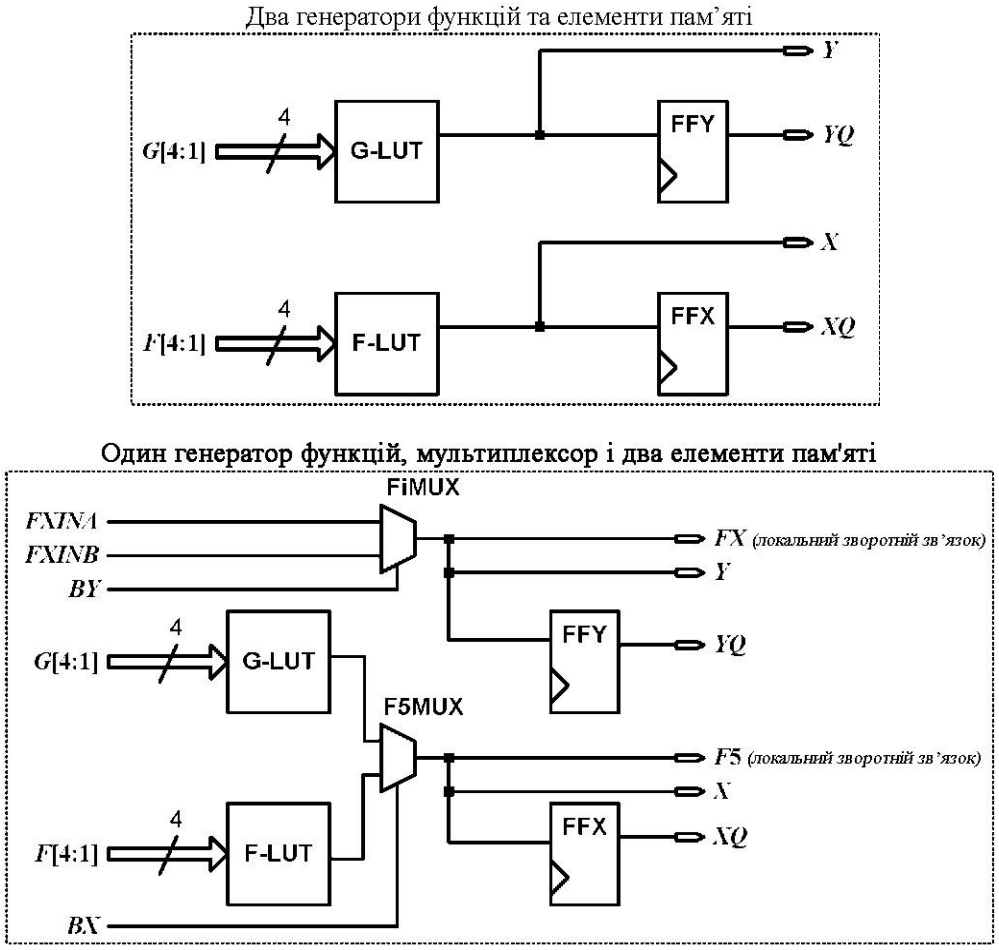



Рис. 1.16. Приклади конфігурацій однієї секції CLB-блока

Реалізація схеми мультиплексора може бути виконана або на одному з LUT-блоків, або за допомогою наявного мультиплексора FiMUX. У разі реалізації комутаційної функції п'яти змінних можуть бути використані генератори функцій G-LUT і F-LUT однієї секції, на входи яких подаються значення чотирьох змінних, а значення п'ятої змінної – на селективну лінію *BX* мультиплексора F5MUX, на виході якого і буде формуватися значення функції. Вихідні лінії FX і F5 використовуються для організації локальних зв'язків з сусідніми блоками, не використовуючи загальні ресурси матриці комутуючих з'єднань.

У блоках типу SLICEM генератори функцій також можуть бути сконфігуровані для реалізації елементів розподіленої пам'яті з інформаційною ємністю 16x1 біт або для реалізації 16-розрядного зсувного регістра (рис. 1.17) [1].

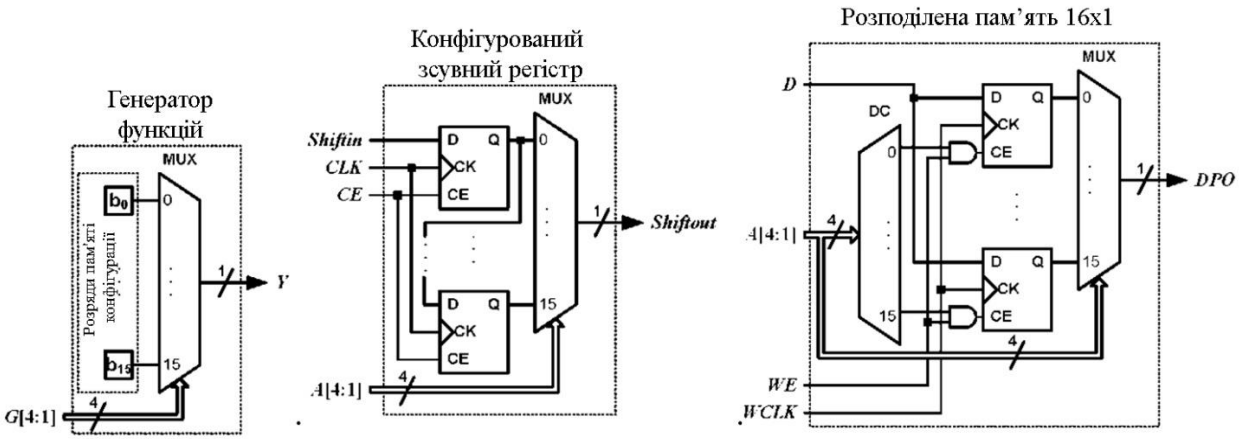



Рис. 1.17. Приклади конфігурацій одного LUT-блока секції SLICEM

Спеціалізовані схеми прискорених перенесень, що входять до складу кожної секції CLB-блоків, містять програмовані мультиплексори і фіксовані логічні елементи AND і XOR, які можуть бути використані для реалізації схем повних однорозрядних суматорів і помножувачів. Наявність наскрізних каналів перенесень, наявних у кожній секції CLB-блоків, дозволяє використовувати FPGA SPARTAN-3E як продуктивну платформу для реалізації широкого спектра каскадованих схем.

Якщо на початку своєї появи мікросхеми FPGA застосовувалися як нова альтернатива БМК, то в подальшому, завдяки своїй експлуатаційній гнучкості і продуктивності, вони стали витісняти серійно випускні і замовні НВІС. Вимоги сучасного ринку вбудованих систем за скороченням часу, відведеного на проектування і виготовлення нових модифікацій цифрових пристроїв, змушують виробників поступово відмовлятися від стандартної елементної бази на користь програмованих логічних інтегральних схем. У свою чергу ці зміни послужили поштовхом в архітектурному розвитку ПЛІС і появі таких понять, як програмовані системи на кристалі PSoC (Programmable System-on-a-Chip), динамічно реконфігуровані системи (Run-time Reconfigurable Systems) і реконфігуровані обчислення (Reconfigurable Computing).

### 1.2.6. Програмовані системи на кристалі PSoC

Досягнення сучасної напівпровідникової промисловості дозволяють виготовляти інтегральні схеми цифрових пристроїв з надвеликим ступенем інтеграції. Виробники ПЛІС типу FPGA збільшують не тільки кількість і щільність компонування логічних блоків, а й ь розміщують на програмованому кристалі такі функціональні блоки, як мікропроцесорні ядра, вбудовані запам'ятовуючі пристрої, блоки арифметичних прискорювачів, інтерфейсні контролери стандартного устаткування тощо, перетворюючи тим самим ПЛІС у програмовані системи на кристалі PSoC (Programmable System-on-a-Chip).

Одним із прикладів програмованої системи на кристалі є FPGA сімейства ZYNQ-7000 фірми Xilinx Inc. Інтегральна схема ZYNC-7000 EPP (Extensible Processing Platform) виконана за технологічним процесом SRAM 28 нм і являє собою FPGA з вбудованими додатковими функціональними блоками [32]. Крім 350 тисяч логічних блоків (для FPGA Z-7045), кристал містить багатоядерний блок обчислювачів, побудований на базі двох ядер процесора ARM Cortex-A9, 2 Мбіт вбудованої багатопортової пам'яті, контролери зовнішніх динамічних оперативних запам'ятовуючих пристроїв (DDR2i DDR3), контролери пристроїв flash-пам'яті (NAND і NOR), два вбудованих блоки високо-швидкісних 12-розрядних аналого-цифрових перетворювачів,

вбудований контролер інтерфейсу PCIe, вбудовані контролери спеціалізованих інтерфейсів, таких як $I^2C$, USB 2.0, Gigabit Ethernet, UAR, CAN, SPI та ін. (рис. 1.18) [1].

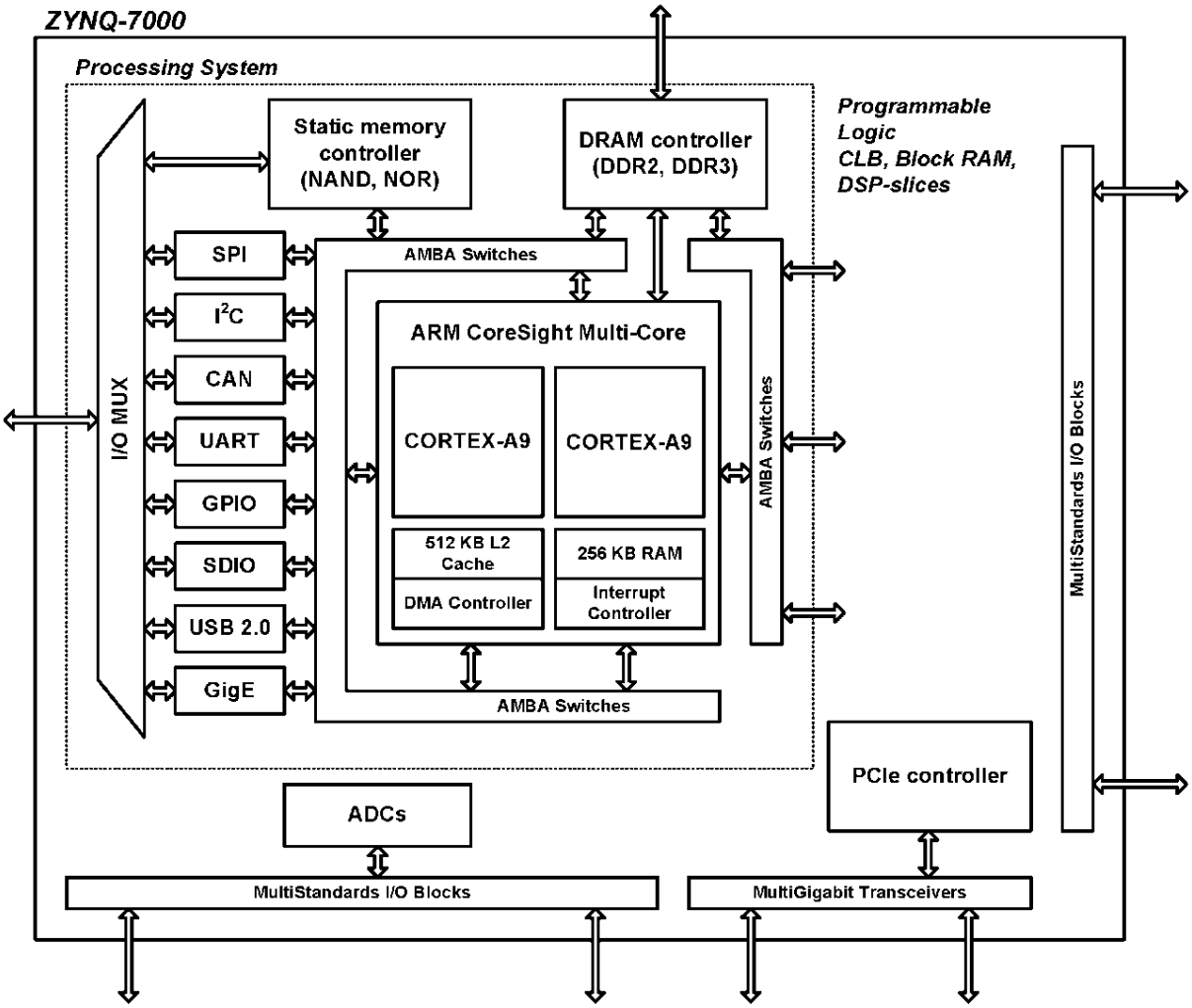



Рис. 1.18. Структура програмованої системи
на кристалі ZYNQ-7000

Програмовані ресурси дозволяють користувачеві реалізовувати різні цифрові пристрої, у тому числі і власні мікропроцесорні ядра. Подібні рішення отримали назву *софт-процесор*, оскільки, на відміну від процесорів, виготовлених як замовні НВІС, для них існує можливість вносити зміни в апаратну реалізацію. Прикладом такого процесора може служити IP-компонента MicroBlaze фірми Xilinx Inc. [33]. Дана компонента спроектована з урахуванням ефективної реалізації для таких серій ПЛІС, як Spartan-2, Spartan-3, Virtex-II Pro і

Virtex-4 і може ефективно застосовуватися і для більш пізніх серій FPGA.

MicroBlaze – 32-розрядне мікропроцесорне RISC-ядро, побудоване за гарвардською архітектурою. Триступенева конвеєрна система разом із тридцятьма двома 32-бітними регістрами загального призначення (РЗП) визначають MicroBlaze як досить ефективну компоненту при реалізації програмованих систем на кристалі. Запам'ятовуючі пристрої для зберігання коду та даних можуть розташовуватися як у модулях BRAM FPGA, так і зовнішніх підключуваних НВІС. Ряд внутрішніх системних блоків MicroBlaze є конфігурованими і можуть бути модифіковані проектувальником на свій розсуд (рис. 1.19) [1].

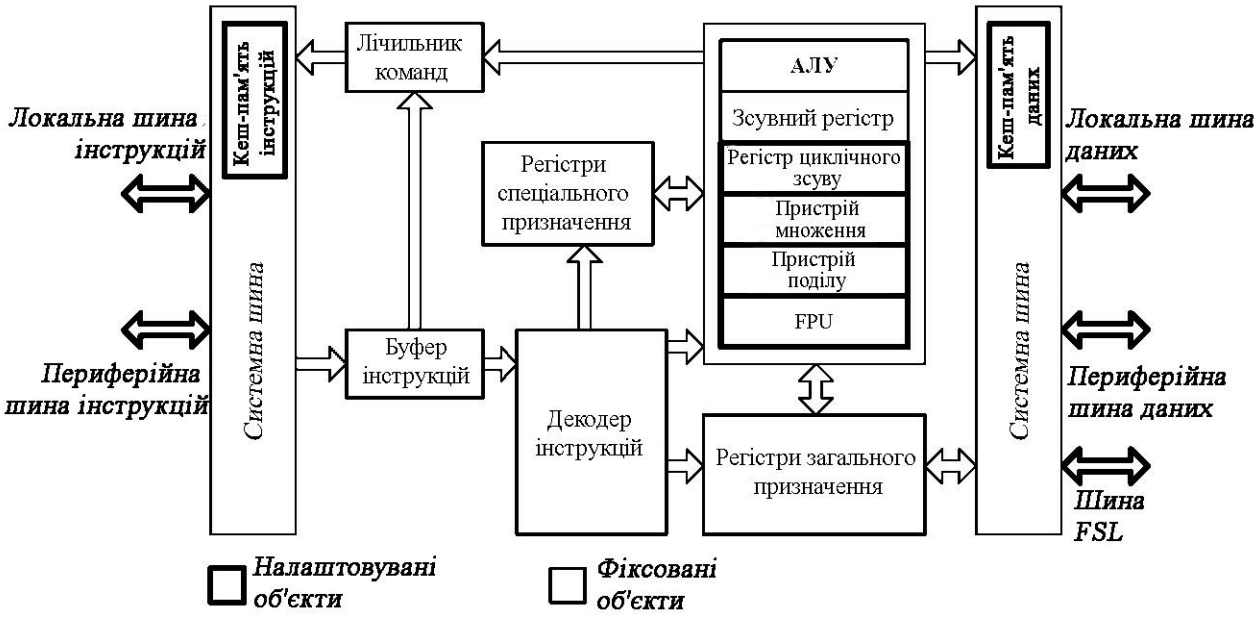



Рис. 1.19. Налагоджувана архітектура софт-процесора MicroBlaze

Реалізована для ПЛІС компонента MicroBlaze має кілька інтерфейсних шин, що дозволяють підключати двопортові ЗП BRAM, периферійні пристрої, контролери яких можуть бути реалізовані ресурсами самої FPGA.

Спеціальна інтерфейсна шина FSL (Fast Simplex Link) дає можливість підключати до конвеєрної структури MicroBlaze розроблені користувачем модулі акселерації різних функцій. Модуль обробки даних з рухомою точкою (FPU, Floating Point Unit), що входить до складу MicroBlaze та підтримує стандарт

IEEE 754, дозволяє використовувати софт-процесор для керуючих вбудованих систем, які потребують застосування алгоритмів цифрової обробки сигналів.

Реалізована компонента MicroBlaze для різних типів FPGA має різні апаратурні витрати і показники продуктивності. Так, для FPGA серії Spartan, що функціонують на частоті 100 МГц, MicroBlaze досягає пікової продуктивності в 92 MIPS (Million Instructions Per Second), а для FPGA серії Virtex-4 продуктивність оцінюється як 166 MIPS для частоти 200 МГц. Показники витрат апаратури варіюються від 900 до 2600 LUT-блоків при мінімальній і максимальній конфігурації IP-компоненти.

Ще одним прикладом програмованих систем на кристалі може служити ряд інтегральних схем, що випускаються корпорацією Cypress Semiconductor [34]. Відмінною особливістю таких інтегральних схем є наявність трьох основних частин: вбудований мікропроцесор, множина цифрових конфігурованих блоків і множина аналогових конфігурованих блоків. На рис. 1.20 подана узагальнена архітектура програмованих систем на кристалі, вироблена Cypress Semiconductor [1].

Наприклад, інтегральна схема PSoc5:CY8C55 [35] містить такі основні компоненти: 32-розрядне мікропроцесорне ядро ARM CORTEX-M3, 256 кбайт незалежної пам'яті, 64 кбайт статичного ОЗП, 128 байт кеш-пам'яті, 24 цифрових універсальних конфігурованих блоки, вбудовані контролери USB 2.0, SPI, $I^2C$, UART чотири вбудованих 16-розрядних таймери, конфігуровану дельта-сигму АЦП, чотири 8-розрядних модулі ЦАП, що налагоджуються, чотири модулі аналогових компараторів, чотири конфігурованих аналогових модулі, здатних виконувати функції аналогового підсилювача з програмованим коефіцієнтом підсилення, трансімпедансний підсилювач, схеми захоплення і утримання сигналів.

Багата номенклатура вбудованих цифрових і аналогових блоків разом з мікропроцесорним ядром дозволяє проектувальникам реалізовувати широкий спектр цифрових і аналого-цифрових систем на одному кристалі. Наявність можливості статичної реконфігурації дозволяє одноразово спроектовану систему налагоджувати на виконання різних завдань, не вдаючись до її повторного проектування і виробництва.

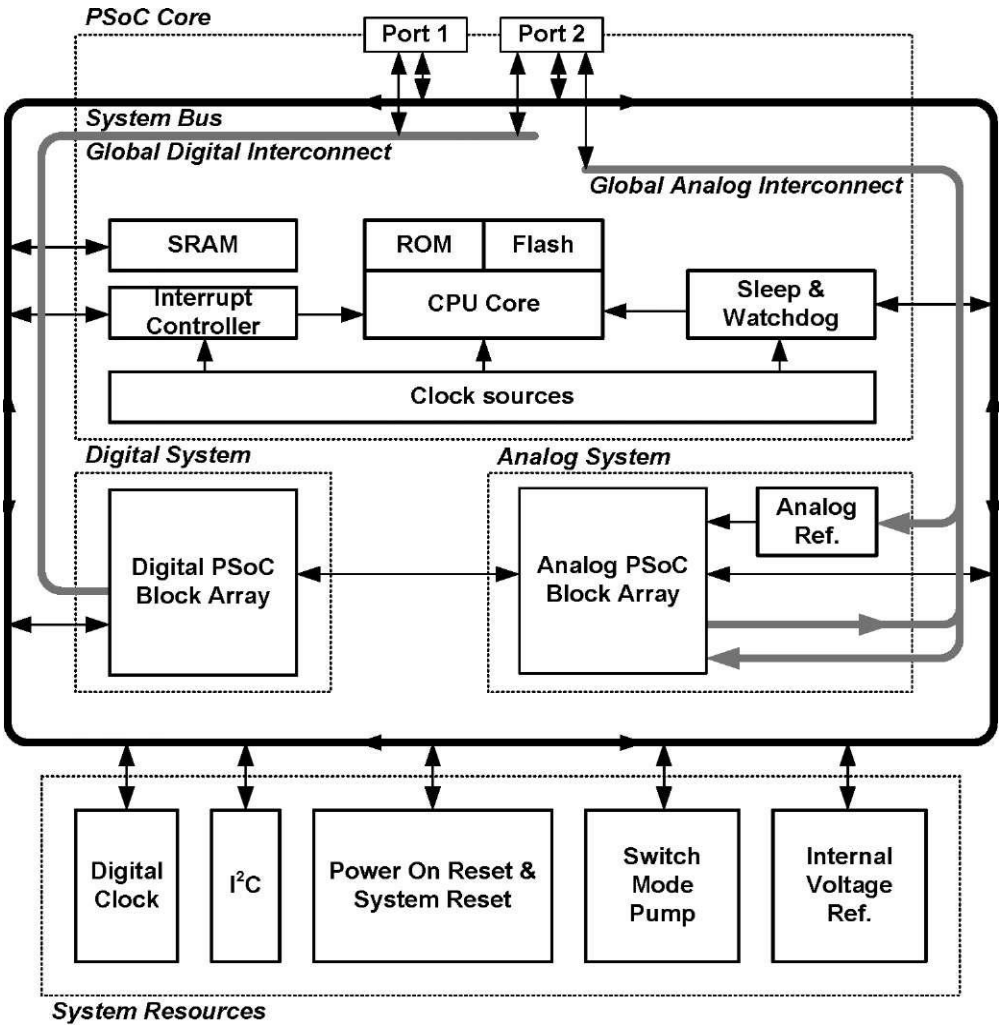



Рис. 1.20. Архітектура PSoC корпорації Cypress Semiconductor

### 1.2.7. Мережі на кристалі

Традиційно функціональні компоненти систем на кристалі об'єднуються за допомогою множини провідників, так званих шин. Наприклад, у мікропроцесорних системах на кристалі можна виділити шину управління (множина провідників, за допомогою яких передаються сигнали управління), шину даних і адреси.

Застосування багаторозрядних шин дозволяє збільшувати швидкість передачі даних між компонентами, проте разом з тим виникає ряд технічних проблем:

1) збільшення кількості компонент, що підключаються до шини, збільшує значення паразитної ємності, що може негативно позначитися на таких параметрах системи, як споживана потужність;

2) використання протяжних багаторозрядних шин ускладнює процес трасування і розміщення компонентів, що призводить до збільшення загальної площі напівпровідникового кристала;

3) збільшення протяжності внутрішніх шин змушує розробників систем на кристалі вирішувати завдання коректної синхронізації віддалених компонент;

4) велика кількість компонент, підключених до єдиної шині, вимагає наявності додаткової керуючої апаратури (шинного арбітра), що дозволяє вирішувати конфлікти при адресації різних компонент і передачі даних;

5) пропускна спроможність шин обмежена і визначається не тільки кількістю провідників, але і кількістю підключених компонент;

6) велика кількість шинних провідників ускладнює процес їх тестування при реалізації процедур самотестування систем на кристалі.

Для усунення перерахованих проблем було запропоновано реалізовувати міжз’єднання IP-компонент систем на кристалі не за допомогою шинної архітектури, а за допомогою єдиної універсальної мережі міжз’єднань, що налагоджуються [36]. Мережа на кристалі (NoC, Network on-a Chip) являє собою множину необхідних IP-компонент, що знаходяться в мережевому комунікаційному просторі, що містить блоки мережевих інтерфейсів (адаптерів), блоки маршрутизаторів і мережеві канали (міжз'єднувальні провідники) (рис. 1.21) [1].

Множина мережевих двонаправлених каналів разом з маршрутизаторами формують єдине універсальне середовище передачі пакетів даних різного призначення між IP-компонентами. Основними перевагами архітектури мереж на кристалі є:

— багатошаровість і масштабованість;

— гнучкість і зручні налагодження мережевої топології;

— можливість організації з'єднання типу «точка-точка» для довільної пари IP-компонент;

— застосовність мережевого принципу GALS (Globally Asynchronous Locally Synchronous) при реалізації функціональних ядер окремих IP-компонент і механізмів їх взаємодії.

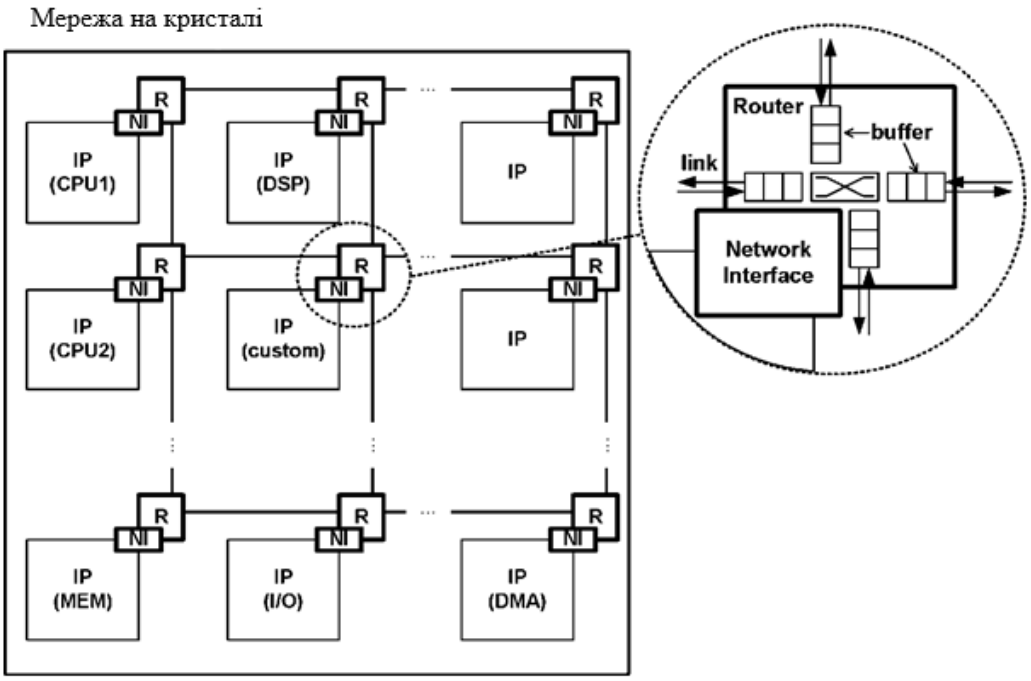



Рис. 1.21. Узагальнена архітектура мережі на кристалі

Передача даних між IP-компонентами здійснюється за допомогою трирівневої мережевої моделі, яка має фізичний рівень, транспортний рівень і рівень транзакцій [37]. На фізичному рівні забезпечується комунікація обраних IP-компонент за допомогою фізичних провідників. На транспортному рівні вирішуються завдання маршрутизації і передачі даних між IP-компонентами за допомогою пакетів, які можуть містити не тільки дані, що передаються, а й ідентифікатори IP-компонент і іншу системну інформацію. На цьому рівні також можуть вирішуватися завдання забезпечення надійності переданих даних за допомогою завадостійких кодів. Рівень транзакцій визначає конкретні типи операцій (читання або запис даних) для підключених IP-компонент. Таким чином, фізичний рівень забезпечується мережевими каналами, транспортний – маршрутизаторами, рівень транзакцій – мережевими адаптерами.

Важливу роль у проектуванні NoC відіграє обрана мережева топологія, що визначає не тільки уількість задіяних IP-компонент, але і кількість маршрутизаторів і їхню конфігурацію. Серед усього різноманіття топології виділяють вісім, що

найчастіше зустрічаються [38] при реалізації комерційних мереж на кристалі (рис. 1.22) [1].

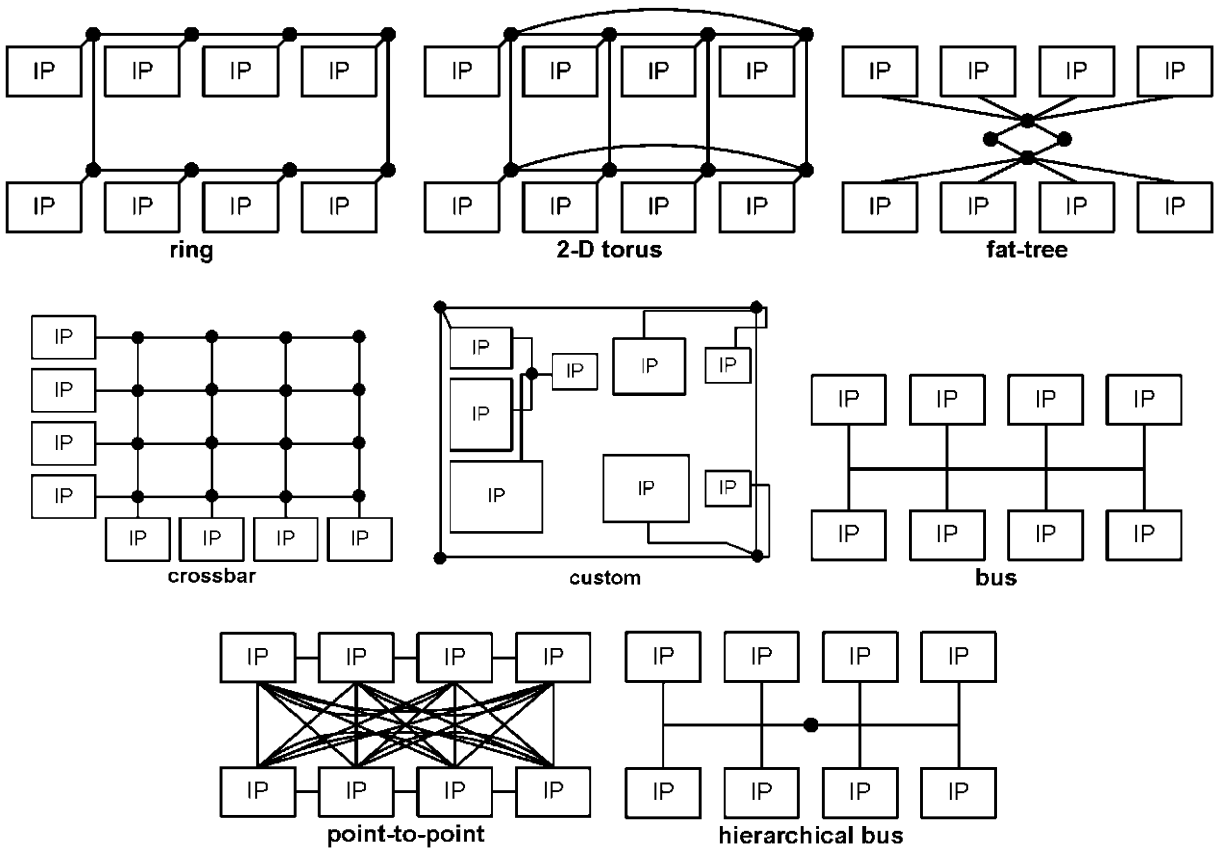



Рис. 1.22. Типи топологій мереж на кристалі

Крім топології, істотний вплив на продуктивність мережі на кристалі має реалізація алгоритмів маршрутизації пакетів даних. На відміну від систем, на кристалі з шинною архітектурою для мереж на кристалі можлива реалізація паралельної передачі даних по послідовних каналах між довільною множиною IP-компонент. Так, шинна архітектура має на увазі наявність синхронізації, а для мереж на кристалі всі транзакції є асинхронними, на шині має бути присутній ведучий пристрій (майстер), який забезпечує управління переданими пакетами і адресацією інших IP-компонент. За наявності кількох майстер-пристроїв на шині необхідна наявність арбітра, який не допускає і усуває конфлікти, що виникають, наприклад при одночасному захопленні шини декількома «майстрами». Подібні процедури не тільки складні в реалізації, але і вимагають додаткового часового

ресурсу, що негативно позначається на продуктивності системи. У роботі [39] подано результат порівняння основних характеристик цифрової системи, що містить 72 IP-компоненти, виконаної у двох варіантах: як система на кристалі з шинною архітектурою (SoC),і в якості мережі на кристалі (NoC). Обидва варіанти були виконані за однаковими технологічними нормами і мають такі значення показників: кількість логічних вентилів для SoC дорівнює 400000, для NoC – 210000. Максимальна частота функціонування SoC – 250 МГц, NoC – 750 МГц. Тривалість процедури шинного арбітражу для SoC – 42 шинних цикли, для NoC – 2 цикли. Пропускна спроможність SoC – 5 Гбіт/с, NoC – 100 Гбіт/с.

Симбіоз технології мереж на кристалі і технології програмованих логічних інтегральних схем є дуже перспективним через наявність можливостей реалізовувати різні типи мережевих топологій за допомогою реконфігурації маршрутизаторів і фізичних каналів [40] і можливості реконфігурації апаратури самих IP-компонент [41]. При цьому новими науковими та інженерними завданнями є планування обчислювальних процесів з урахуванням динамічної реконфігурації не тільки однієї процесорної IP-компоненти, а всієї системи в цілому; тестування і діагностика IP-компонент, мережевої апаратури і системи в цілому; забезпечення надійного зберігання та передачі даних між компонентами тощо.

## 1.3. Компоненти інтелектуальної власності

Історично поняття системи на кристалі з'явилося завдяки стрімкому розвитку ринку замовних НВІС. Цьому сприяло масове поширення програмних систем автоматизованого проектування НВІС і здешевлення процесу їх виготовлення. Замовні НВІС є вузькоспеціалізованими цифровими пристроями, виробленими невеликими партіями для застосування в особливих додатках вбудованих систем. Основне функціональне ядро замовних НВІС прийнято називати **IP-core** (Intellectual Property Core – компонента інтелектуальної власності, далі IP-компонента). Подібні IP-компоненти проектуються фірмами-розробниками і можуть бути повторно використані при

проектуванні інших замовних НВІС. IP-компонента може служити як спеціалізований цифровий пристрій, так і стандартний, але оптимізований фірмою під конкретну архітектуру або технологію. У наш час світовий ринок замовних НВІС виріс настільки, що проектування повнофункціональних обчислювальних систем може здійснюватися тільки за допомогою існуючих IP-компонент. Компанії-розробники пропонують на ринку цілі бібліотеки IP-компонент, вже готових до застосування при проектуванні вбудованих пристроїв і систем.

Виділяють три основні сценарії, за якими проводяться замовні НВІС [2]:

1. *Замовлене проектування:* є замкненим виробництвом, при якому всі компоненти замовної НВІС проектуються і виготовляються на одному заводі.

2. *Інтегроване проектування:* компоненти замовної НВІС проектуються різними компаніями (постачальниками IP-компонент). Інтеграцію різних компонент в одне проектне рішення з подальшим виготовленням кристалів бере на себе один виробник.

3. *Комп'ютерне проектування:* цей вид проектування НВІС стосується компаній, які не мають виробничих ресурсів. Основне завдання комп'ютерного проектування полягає в розробленні топології майбутнього цифрового пристрою за допомогою комп'ютерних систем автоматизованого проектування CAD (Computer-Aided Design). При цьому компанія-розробник може використовувати сторонні IP-рішення разом з проектуванням своїх власних спеціалізованих компонент.

При проектуванні замовних НВІС як SoC визначальним є вибір модулів IP-компонент. Сучасні SoC можуть включати до свого складу велику кількість стандартних IP-компонент, таких як мікропроцесорні модулі, масиви запам'ятовуючих пристроїв, аудіо- і відео-контролери, контролери різних інтерфейсів, графічні акселератори, модулі цифрової обробки сигналів тощо.

Як правило, компоненти функціональних модулів, що постачають сторонні розробники, можуть являти собою вихідні коди на високорівневих мовах опису апаратури HDL (Hardware Description Language), таких як VHD, Verilog, Abel, AHDL, або схемотехнічні описи на різних рівнях абстракції. Зручність

використання IP-компонент безпосередньо залежить від форми їх подання.

Розрізняють такі основні типи IP-компонент [42, 43]:

1. *Програмні IP-компоненти* – синтезовані описи на рівні реєстрових пересилань RTL (Register Trasfer Level) або описи у вигляді таблиць міжз'єднань (netlist) стандартних бібліотечних елементів. У разі використання програмних IP-компонент розробник сам вирішує завдання синтезу, розміщення і трасування при проектуванні цифрових пристроїв.

2. *Фірмові IP-компоненти* – це багаторазово використовувані компоненти, структурно і топологічно оптимізовані для поліпшення показників швидкодії і витрат апаратури за рахунок оптимального вибору і розміщення базових логічних елементів для конкретної технології. Фірмові IP-компоненти можуть являти собою синтезовані описи високорівневою мовою опису цифрової апаратури типу VHDL або таблицю міжз'єднань бібліотечних елементів з обов'язковим зазначенням технологічних норм виготовлення.

3. *Апаратні IP-компоненти* – повторно використовувані компоненти, оптимізовані за швидкодією, апаратурними витратами і споживаною енергією з урахуванням конкретної технології реалізації. Такі компоненти часто реалізуються у вигляді таблиць міжз'єднань, повністю розміщених і протрасованих елементів обраної технології, або як описи у форматі GDSII (Graphic Database System), який використовується для проектування фотошаблонів замовних НВІС.

При виборі типу IP-компоненти слід враховувати такі показники SoC, як можливість повторного використання, експлуатаційна гнучкість при функціонуванні, швидкодія, вартість, час проектування і виготовлення. Система на кристалі у вигляді замовної НВІС є одним з оптимальних рішень для досягнення найкращих показників зі швидкодії. Однак цей підхід не забезпечує повної або часткової експлуатаційної гнучкості реалізованої цифрової системи. Стрімке зростання сучасних додатків, поява нових протоколів і алгоритмів вимагають швидкої адаптації і модифікації цифрових пристроїв, готових у найкоротші терміни до появи на ринку. Так, алгоритмічна зміна у функціонуванні навіть однієї IP-компоненти веде до

перепроектування всієї SoC і виробництва нової серії замовних НВІС. Забезпечення проектної та експлуатаційної гнучкості SoC можна домогтися, використовуючи такі програмно-апаратні рішення на прикладі мікропроцесорних IP-компонент:

1. *Програмне рішення.* Цей підхід щодо забезпечення гнучкості SoC полягає у використанні IP-компонент, які являють собою функціональні аналоги серійно мікроконтролерів (МК) і мікропроцесорів (МП), що випускаються. Зміна функціонування всієї SoC може бути досягнута за рахунок модифікації виконуваного коду. Широке використання мов програмування і автоматизованих засобів розроблення програмного забезпечення (ПЗ) для стандартних МК і МП дозволяє ефективно в найкоротші терміни змінити функціонування SoC. Недоліком програмного рішення є можливість підвищення швидкодії системи тільки за рахунок нового ПЗ, а не за рахунок зміни її апаратної частини.

2. *Апаратно-програмне рішення.* Програмовані логічні інтегральні схеми дозволяють реалізовувати призначену для користувача логіку довільної складності аж до мікропроцесорних компонент (софт-процесори). За своєю суттю софт-процесори є програмними IP-компонентами серійних МП і разом з ПЛІС являють собою найбільш гнучке рішення при проектуванні і функціонуванні SoC. Як і програмні рішення, що налагоджуються, процесори можуть змінювати свою поведінку за допомогою запису модифікованого виконуваного коду в пам'ять команд. Якщо застосування програмного підходу не принесло бажаних результатів у підвищенні швидкодії системи, то розробник може змінювати і доповнювати апаратну частину процесора, що налагоджується, за рахунок резервних можливостей ПЛІС. Іншим апаратно-програмним рішенням після досягнення проектної та експлуатаційної гнучкості SoC є застосування вбудованих процесорів разом з ПЛІС. Фіксована апаратура вбудованих процесорів має вищі показники швидкодії, ніж процесори, що налагоджуються. Наявність стандартних засобів проектування і верифікації ПЗ дозволяє без особливих проблем переносити машинний код з настільних систем на вбудовані процесори. Додаткові ресурси ПЛІС дозволяють реалізовувати для вбудованих процесорів будь-які IP-компоненти периферійних пристроїв. Апаратно-програмні рішення є

найбільш ефективними для досягнення проектної та експлуатаційної гнучкості швидкодіючих SoC.

3. *Апаратне рішення* має мінімальну гнучкість порівняно з розглянутими вище програмними і апаратно-програмним рішеннями. Як правило, апаратне рішення являє собою SoC, виконану у вигляді замовної НВІС на основі спеціалізованих апаратних IP-компонент. На відміну від апаратно-програмних рішень на базі ПЛІС, замовні НВІС не є програмованими логічними пристроями, однак мають підвищені показники зі швидкодії.

При проектуванні SoC розробник має право знаходити компромісні рішення щодо забезпечення високих показників швидкодії, проектної та експлуатаційної гнучкості систем. Так, графік на рис. 1.23 ілюструє співвідношення цих показників при виборі різних шляхів реалізації SoC [1].

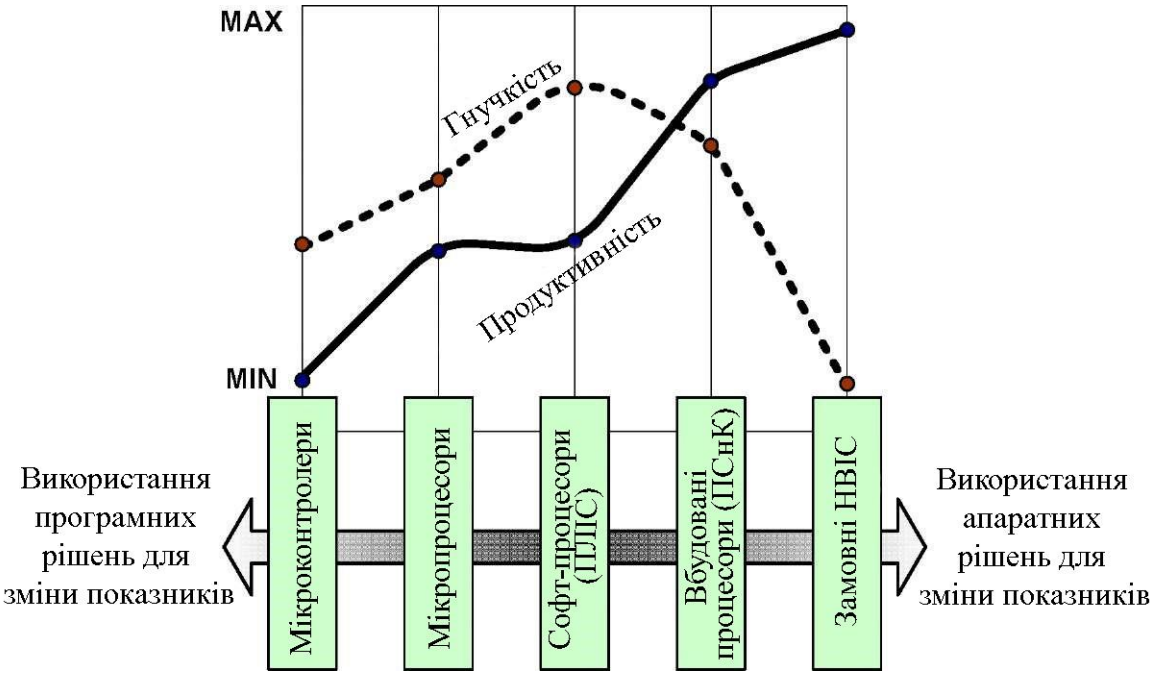



Рис. 1.23. Відносне співвідношення показників продуктивності і гнучкості

Як видно з графіка, найбільш оптимальним рішенням є використання програмованих логічних пристроїв і систем на кристалі, що доводиться стрімким розвитком даної технології в останнє десятиліття.

## 1.4. Вбудовані системи на базі ПЛІС

Історично поняття вбудованих систем завжди асоціювалося з використанням мікроконтролерів (МК). На відміну від мікропроцесорів, МК мають спрощену обчислювальну архітектуру, невеликі за обсягом запам'ятовуючі пристрої для зберігання даних і коду. Основними сферами застосування МК є функції управління периферійними пристроями вбудованої техніки:

1) *пристрої виведення інформації:* цифро-аналогові перетворювачі (ЦАП), світлодіоди, семисегментні світлодіоди, однорядкові і багаторядкові дисплейні панелі, акустичні системи тощо;

2) *пристрої введення інформації:* аналого-цифрові перетворювачі (АЦП), електричні ключі, перемикачі, тригерні кнопки, матричні клавіатури тощо;

3) *електромеханічні та сенсорні пристрої:* крокові двигуни, термодатчики, електромагнітні реле, фотодіоди тощо.

Побудова будь-якої вбудованої системи починається з визначення основних функцій і характеристик центральної обчислювальної компоненти – мікроконтролера. Крім необхідних і достатніх для вирішення поставлених завдань обчислювальних ресурсів, МК повинен мати такі властивості, як зручність програмування, наявність систем автоматизованого проектування ПЗ з підтримкою мов високого рівня, малими габаритами корпусу НВІС, прийнятно малою кількістю контактних майданчиків, низьким енергоспоживанням. Вибір МК є пошуком компромісних рішень, при яких вбудована система буде мати необхідну продуктивність при можливих малих фізичних габаритах, споживаній енергії і тепла, що виділяється.

Основним чинником, що впливає на функціонування вбудованої системи (ВС), є програмне забезпечення мікроконтролера. По суті ПЗ визначає узгоджену роботу всіх компонент, що входять до складу ВС. Спільне проектування ПЗ і апаратної частини вбудованих систем є ітераційним, багаторівневим процесом, при якому для обраної апаратної платформи створюється виконуваний код, що відповідає визначеним експлуатаційним характеристикам системи. Для первинної верифікації розроблюваного ПЗ проектувальники використовують програмні засоби емуляції апаратної складової

ВС. Прикладом подібних засобів може служити емулятор цифрових компонент і електричних кіл VSM Proteus Advantage, розроблений фірмою Labcenter Electronics [44]. Попередньо за допомогою вбудованого в емулятор схемотехнічного редактора проектується принципова схема майбутньої ВС. Велика кількість бібліотек різних компонент дозволяє проектувати схеми мікроконтролерних систем з використанням периферійних пристроїв різних типів. Вбудовані в емулятор VSM Proteus генератори і логічні аналізатори сигналів, віртуальні осцилографи, вольтметри і амперметри дозволяють здійснювати перевірку працездатності як цифрових, так і аналогових схем розроблюваного пристрою. При проектуванні мікроконтролерної системи VSM Proteus дозволяє здійснювати спільне налагодження коду і апаратної частини ВС.

Якщо розроблений виконуваний код не задовольняє вимоги щодо функціонування, то доцільно переглянути вибір апаратних компонент для досягнення бажаного результату. Після завершення процесу комп'ютерного проектування і спільної верифікації розробляється прототип ВС у вигляді друкованої плати з необхідними компонентами на ній. Прототипи ВС мають забезпечувати доступ для тестового та діагностичного обладнання до всіх полюсів системи з можливістю проводити верифікацію МК в покроковому режимі.

Наступним етапом після тестування прототипної плати є масове виробництво готових систем, які також мають забезпечувати доступ до своїх внутрішніх компонентів і полюсів. При функціонуванні готової ВС за призначенням можуть виникати такі випадки, при яких необхідний внутрішній доступ:

— заміна (модифікація) поточної версії виконуваних кодів (firmware), які можуть зберігатися у вбудованій пам'яті МК або в окремому модулі НВІС EEPROM;

— тестування і діагностика як окремих елементів ВС, так і з'єднувальних ліній на наявність несправностей.

Заміна firmware здійснюється, як правило, при модифікації алгоритму функціонування ВС (незначні зміни після виправлення поточної версії коду або повна заміна функцій на більш сучасні аналоги). Змінюючи таким чином поведінку ВС, не можна добитися значного збільшення продуктивності або кардинально

поміняти функціонування системи, що може бути досягнуто лише після її повного перепроектування. Недостатня експлуатаційна гнучкість мікроконтролерних ВС і часті випадки перегляду апаратної складової на початкових етапах проектування призвели до появи як прототипних плат, так і реалізованих вбудованих систем, центральною компонентою яких є ПЛІС.

Сучасні високопродуктивні ПЛІС типу CPLD і FPGA дозволяють проектувати і використовувати вбудовані системи, обчислювальним ядром яких є вбудовані процесори і/або софт-процесори. Так, прототипні плати з використанням ПЛІС і мови опису апаратури типу VHDL дозволяють у найкоротші терміни проектувати і верифікувати мікроконтролерні схеми вбудованих систем. Великий вибір мікроконтролерних і мікропроцесорних IP-компонент допомагає розробнику ефективно вирішувати як завдання спільного проектування апаратури і ПЗ, так і завдання збільшення продуктивності системи за рахунок гнучкої зміни її апаратної частини. Одна прототипна плата може бути багато разів використана для проектування і верифікації ВС різних типів. Зазвичай подібні плати будуються з використанням принципу розширюваної архітектури за підтримки стандартних фізичних інтерфейсів доступу і передачі даних, властивих ВС.

Як приклад розглянемо архітектуру плати швидкого прототипування (Fast Prototyping Board) NEXYS-2, розроблену фірмою Digilent Inc. [45]. Модуль NEXYS-2 являє собою системну плату, до якої можливе підключення дочірніх модулів за допомогою чотирьох 12-контактних і однієї 43-контактної монтажної панелі. Основними компонентами системної плати є НВІС FPGA SPARTAN-3E, 16 Мбайт flash-пам'яті даних, 16 Мбайт SDRAM і НВІС конфігурації FPGA XCF02. Програмована вентильна матриця SPARTAN-3E призначена для реалізації цифрової користувальницької логіки, у тому числі і процесорних IP- компонент. НВІС EEPROM XCF02 призначена для зберігання пам'яті конфігурації FPGA, яка також може програмуватися за допомогою послідовного інтерфейсу IEEE 1149.1 (JTAG). Реалізовані проекти, призначені для користувача, для FPGA можуть функціонувати на частоті 50 МГц, що генерується встановленим на платі осцилятором. Вільне посадкове місце на платі може бути використано для додаткового генератора системної частоти.

На рис. 1.24 зображена структурна схема прототипної плати NEXYS-2. Інтегральна схема FPGA SPARTAN-3E XCS500E містить 1164 CLB-блоки (4656 slice-блоків), 73 кбайт розподіленої пам'яті, 360 кбайт вбудованої пам'яті Block RAM, 20 вбудованих блоків конфігурованих помножувачів і 232 блоки IOB [1].

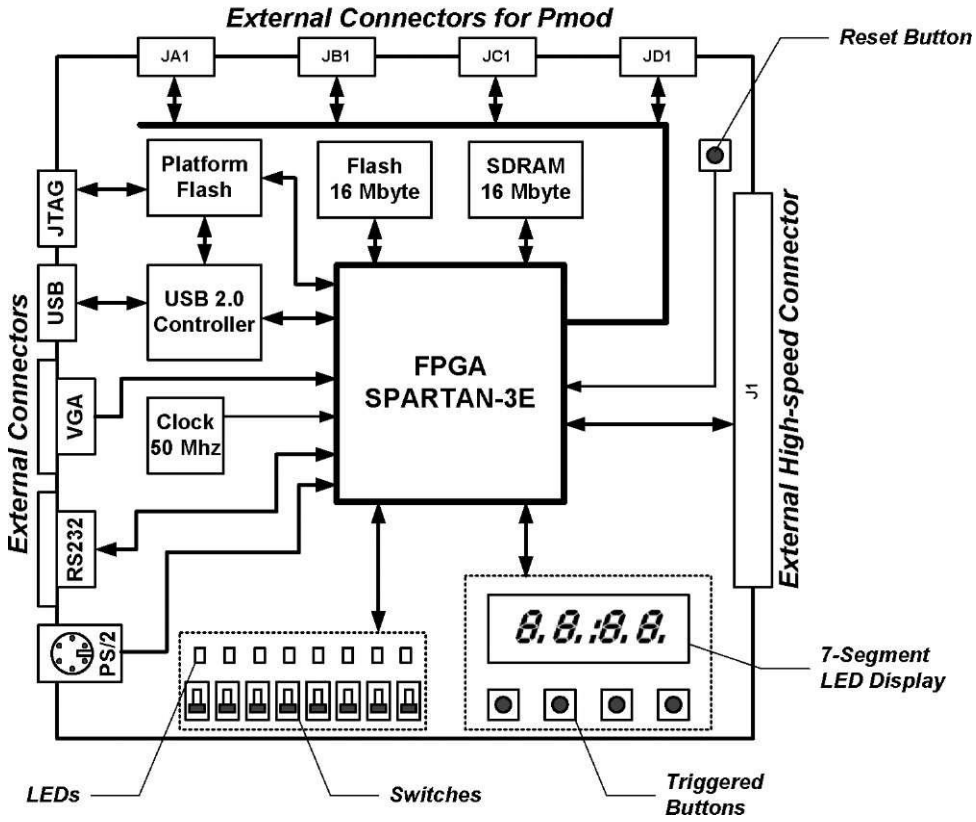



Рис. 1.24. Структура плати швидкого прототипування NEXYS-2

Вбудовані блоки двопортових ОЗУ BRAM можуть бути використані як запам'ятовуючі пристрої для зберігання даних і виконуваних кодів мікроконтролерних IP-компонент. Більшість з 232 програмованих виводів ПЛІС об'єднані в локальні шини, підключені до різних НВІС на платі NEXYS-2 і зовнішніх рознімачів. Два спеціалізованих виводи FPGA підключені до світлодіодного індикатора успішного завантаження пам'яті конфігурації та тригерної кнопки користувальницької функції скидання. Наявні схеми живлення і перетворювачі вхідної напруги 9 В дозволяють виробляти опорні напруги 1,8 і 3,3 В для НВІС прототипної плати і периферійних модулів. Призначені для

користувача електричні перемикачі на системній платі дозволяють вибирати метод завантаження пам'яті конфігурації FPGA: за допомогою EEPROM, через JTAG-порт або USB. Загальний ланцюг сканування (Boundary Scan) JTAG-інтерфейс об'єднує НВІС EEPROM і FPGA. Це дозволяє здійснювати конфігурацію ПЛІС різними способами: за допомогою персонального комп'ютера і спеціалізованого ПЗ або за допомогою JTAG-програматора. Протрасування до кожної монтажної панелі лінії опорних напруг і землі створюють можливість для живлення плат, що підключаються.

Плата NEXYS-2 містить мінімальний набір інтерфейсних компонент вбудованих систем: чотири тригерні кнопки, вісім перемикачів ковзних контактів, вісім світлодіодів і панель, що складається з чотирьох семисегментних індикаторів. Додаткові рознімачі стандартних інтерфейсів VGA і PS/2 дозволяють підключати до плати такі зовнішні пристрої, як монітор і клавіатура.

Обширна номенклатура різних модулів, пропонована фірмою Digilent Inc., робить систему NEXSYS-2 оптимальним і недорогим варіантом плат для швидкого розроблення та верифікації вбудованих систем. Спроектована таким чином ВС згодом може бути виготовлена в компактному варіанті за схемотехнікою прототипу з використанням мікроконтролера, замовної НВІС або ПЛІС.

Для прототипної плати вбудованих систем, побудованих на ПЛІС, можливі два сценарії реалізації обчислювального ядра:

1) використання IP-компоненти, яка є аналогом мікроконтролера, що серійно випускається;

2) розроблення власної спеціалізованої IP-компоненти.

Перший сценарій є прийнятним при кінцевій реалізації ВС на базі МК. Використовуючи різні моделі налагоджуваного МК, проектувальник за короткий термін може оптимально підібрати прийнятну апаратну платформу для розроблюваної системи. При цьому наявні САПР ПЗ серійних МК значно полегшують спільне проектування і верифікацію виконуваних кодів і апаратури. Розроблення спеціалізованої IP-компоненти може виявитися більш ефективним порівняно з першим варіантом. У цьому випадку проектувальник сам визначає оптимальну архітектуру

обчислювального модуля, домагаючись необхідних показників продуктивності системи. Але при цьому на розробника додатково покладається проектування системного ПЗ для спеціалізованої компоненти.

## 1.5. Проектування і реалізація цифрових пристроїв на ПЛІС

Послідовність основних етапів, яку проходить цифрова система від технічного завдання до реалізації, можна проілюструвати схемою, зображеною на рис. 1.25 [1].

Протягом усіх життєвих циклів система подається як об'єкти трьох типів:

— у вигляді *абстрактної моделі* під час комп'ютерного проектування;

— у вигляді *дослідного зразка (прототипу* під час проведення натурних випробувань перед виготовленням серійної партії цифрових систем);

— як *готова система,* що функціонує за призначенням.

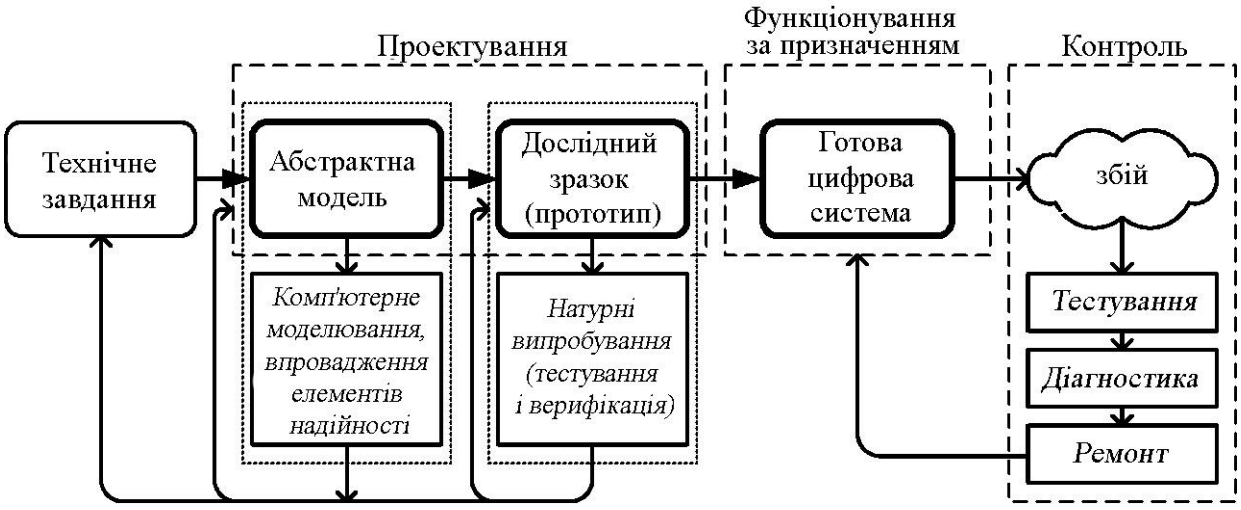



Рис. 1.25. Життєві етапи цифрової системи

Тривалість безвідмовного функціонування готової системи залежить від елементів надійності, застосовуваних на різних життєвих етапах. Так, надійність абстрактної моделі залежить від достовірності використовуваних алгоритмів, методів і засобів

проектування. Проведене комп'ютерне моделювання цифрової системи дозволяє виявляти невідповідності розробленої моделі поставленому технічному завданню.

Надійність дослідного зразка підтверджується натурними випробуваннями, які полягають у перевірці працездатності та відповідно до технічного завдання всіх найважливіших параметрів реалізованої системи. Якщо тестування прототипу не виявило збоїв, то система вважається успішно спроектованою, після чого переходять до масового виробництва виробу з подальшою реалізацією його на ринку.

У випадках виникнення збоїв під час функціонування системи за призначенням можуть застосовуватися елементи надійності, які були закладені на початкових етапах проектування. Цифрові системи та пристрої з вкладеними елементами надійності називаються *контролепридатними пристроями,* а самі елементи надійності дозволяють тестувати пристрої і всю систему на наявність збоїв, локалізувати збої діагностичними методами і по можливості усувати причини збоїв у вигляді ремонту.

Взагалі під проектуванням розуміють розроблення технічної документації, яка дозволяє виготовляти пристрій при заданих обмежувальних параметрах [46]. Проектування цифрових пристроїв часто називають *«логічним проектуванням»* через те, що опис поведінки практично всіх цифрових схем може бути подано за допомогою логічних (булевих) функцій, а виготовлення можливе за допомогою найпростіших логічних вентилів.

Проектування сучасних цифрових пристроїв високого ступеня інтеграції не може обмежуватися старими підходами, а вимагає застосування високорівневих мовних описів, які дозволяють визначати як внутрішню структуру, так і функціонування розроблюваних пристроїв з використанням різних абстрактних моделей. Сучасні методи проектування – це складний, багатоступінчастий процес, при якому проектувальник здатний задавати необхідні обмеження і переглядати проектні рішення на кожному з етапів.

Одним з основних понять проектування є *синтез* [47]. Так, під синтезом можна розуміти процес переходу від функціонального опису пристрою до його структури, яка представляє пристрій на одному з абстрактних рівнів.

Через те, що сучасні програмовані логічні пристрої є унікальною платформою як для реалізації, так і верифікації цифрових схем, особливу увагу приділимо методам, засобам і основним етапам проектування пристроїв, технологічною базою для яких є ПЛІС. При розгляді питань проектування цифрових пристроїв пріоритет віддамо високорівневим мовним методам опису разом з методами логічного і фізичного синтезу для ПЛІС типу FPGA.

Перед тим як безпосередньо перейти до опису питань, пов'язаних з проектуванням, визначимо види абстрактних моделей, які знаходять широке застосування при поданні цифрових пристроїв.

Необхідність подання розроблюваних цифрових пристроїв на різних рівнях абстракції обумовлена декількома причинами. По-перше, таке подання полегшує розробнику сам процес проектування, даючи йому розуміння як логічної, так і фізичної структури пристрою. По-друге, багаторівневе подання сприяє прискореній адаптації пристрою при зміні обмежувальних параметрів проектування. Наприклад, зміна елементної бази для реалізації пристрою не впливає на його поведінкову модель і логічну структуру, а отже, не вимагає перегляду всієї проектної документації. По-третє, використання того або іншого методу тестування вимагає наявності інформації про цифровий пристрій на різних рівнях абстракції.

Виділяють п'ять основних рівнів подання цифрових пристроїв, якими користуються проектувальники: поведінковий, функціональний, логічний, електричний, фізичний.

На кожному з названих рівнів абстракції цифровий пристрій розглядається як модель, що має певну структуру і функціонування.

Розглянемо докладніше типи моделей, що належать кожному з рівнів абстракції.

1. *Поведінкова модель*. має найвищий рівень абстракції. Вона не містить інформацію про внутрішню організацію пристрою. Відомими є лише функції пристрою. Дана модель має назву «чорного ящика» і використовується на початкових етапах проектування. Крім визначення функцій, поведінкова модель може включати таку інформацію про цифровий пристрій, як

кількість зовнішніх контактів (портів), їхні типи (вхідні, вихідні, двонаправлені) і їхня назва. Для групи портів загального призначення (шин) може вказуватися їхня розрядність. Поведінкові моделі розроблюваних пристроїв можуть бути визначені за допомогою високорівневих мов опису апаратури, таких як VHDL.

2. *Функціональна модель* заснована на деталізації внутрішньої структури проектованого цифрового пристрою до основних функціональних блоків, поведінка і структура яких добре відомі. Дана модель також містить інформацію про взаємозв'язки функціональних блоків між собою. При проектуванні цифрових пристроїв функціональний рівень часто подається як рівень реєстрових передач (RTL, Register Trasfer Level) [48]. Поведінкова модель цифрового пристрою може бути перетворена у функціональну модель. Такий процес перетворення носить назву *структурного синтезу*, або *RTL-синтезу*. Однак сама функціональна модель може бути відправною точкою в процесі проектування пристрою. Мова VHDL, крім поведінкового опису, має можливість структурно описувати цифровий пристрій за допомогою стандартних бібліотечних цифрових вузлів або за допомогою призначених для користувача компонент.

3. *Логічна модель* є описом пристрою у вигляді набору найпростіших базових цифрових модулів (вентилів) і зв'язків між ними. Ця модель може бути застосована при логічному синтезі, при якому кожна компонента функціональної моделі транслюється у відповідний опис за допомогою базису логічних вентилів. У рідкісних випадках логічна модель є відправною точкою проектування цифрових пристроїв. Виняток становлять прості логічні схеми, функціонування яких легко можна описати булеві виразами. Для сучасних автоматизованих засобів проектування логічний рівень абстракції є останнім технологічно незалежним поданням цифрових пристроїв.

4. *Електрична модель* ґрунтується на інформації про те, з яких електричних компонент складається цифровий пристрій і як ці компоненти взаємопов'язані один з одним. Зазвичай рівень абстракції, відповідний електричній моделі, використовує схемотехнічний опис пристрою з застосуванням транзисторів, резисторів, конденсаторів і провідних ліній. Велике поширення

серед електричних моделей цифрових пристроїв отримали так звані SPICE-моделі [49], за допомогою яких можна детально аналізувати такі параметри, як споживана потужність, струми витоку, перемикальна активність тощо. Електричний рівень абстракції є технологічно залежним і застосовується на останніх стадіях проектування цифрового пристрою для визначення його електричних параметрів і попередніх витрат апаратури на кінцеву реалізацію.

5. *Фізична модель* має повну інформацію про внутрішню будову напівпровідникового кристала цифрового пристрою. Дана інформація містить такі відомості: розміри базових елементів (транзисторів), параметри сигнальних ліній, фізичне розташування елементів і сигнальних ліній на кристалі. Часто цей вид моделі називають геометричним, оскільки основна інформація подана як геометричні параметри кристала цифрового пристрою. Через колосальний обсяг інформації про топології сучасних інтегральних схем фізична модель практично не використовується на початкових стадіях проектування.

Технологічно залежні моделі, крім інформації про структуру розроблюваого цифрового пристрою, містять додаткові відомості, що дозволяють визначати такі характеристики: час поширення сигналів на критичних ділянках, максимальна частота функціонування пристрою, споживана енергія, тепло, що розсіюється, тощо. Залежно від кінцевої технології виготовлення цифрових пристроїв деякі з наведених моделей можуть бути відсутніми або можуть бути об'єднані в єдину модель. Для сучасних систем автоматизованого проектування функціональні і логічні рівні об'єднуються в єдиний RTL-рівень, для якого базовими елементами можуть служити як логічні вентилі, так і стандартні функціональні компоненти (мультиплексори, дешифратори, суматори, регістри тощо). Якщо цифровий пристрій розробляється з урахуванням його реалізації на ПЛІС, то електричний і фізичний рівні абстракції при цьому об'єднуються в єдиний фізичний рівень, базові елементи якого подані як топологія програмованих ресурсів ПЛІС, а результатом фізичного синтезу є конкретне значення пам'яті конфігурації.

Процес проектування цифрових систем є комплексним і неоднозначним. Він залежить від багатьох чинників і параметрів,

які слід враховувати розробнику на кожному етапі. Проектування можна подати як сукупність послідовних переходів від одного рівня абстракції до наступного найближчого з метою виготовлення цифрового пристрою. Розрізняють дві основні методології проектування: *спадне* проектування (зверху вниз) і *висхідне* проектування (від низу до верху). Спадне проектування являє собою послідовну декомпозицію цифрового пристрою, поступово переходячи від системного (поведінкового) подання до фізичного (технологічного) рівня. Методологія висхідного проектування полягає в об'єднанні найпростіших модулів у більш складні цифрові компоненти і в кінцевому підсумку створення всього цифрового пристрою. Сучасні САПР цифрових систем, а особливо систем на кристалі, об'єднують у собі обидва підходи проектування. При цьому розробник має право сам вирішувати, яка методологія є найбільш прийнятною на поточній стадії проектування. Згідно з розглянутими рівнями абстракції цифрових пристроїв можна виділити такі базові елементи, за допомогою яких подається розроблюваний пристрій: система, функція/алгоритм, RTL-примітив, вентиль і транзистор (рис. 1.26) [1].

Перехід від одного базису проектування до іншого називається *синтезом.* Методи і алгоритми синтезу виконують трансляцію опису цифрового пристрою з одного рівня абстракції на інший рівень. При цьому вирішуються завдання оптимізації опису з урахуванням базових елементів нового рівня. Проектувальник, задавши опис пристрою на одному з рівнів абстракції, визначає також і набір обмежень, які будуть враховуватися алгоритмами синтезу. Крім користувальницьких обмежень, алгоритми синтезу оперують наближеними критеріями, зміни яких можуть призвести до отримання бажаного результату. Після синтезу ініціюється процедура верифікації, яка полягає в оцінюванні і порівнянні отриманих характеристик нового опису пристрою з затвердженими характеристиками на етапі розроблення технічного завдання. Якщо результати верифікації є незадовільними, здійснюється повернення назад на один або кілька етапів проектування, при цьому відбувається коригування опису пристрою і/або параметрів синтезу.


Рис. 1.26. Методології проектування цифрових пристроїв

Послідовність етапів проектування, як і самі етапи, цілком залежать від кінцевої технології виготовлення цифрових пристроїв. Розглянемо дев'ять основних етапів, які містить процес проектування цифрових пристроїв, з метою їх кінцевої реалізації на сучасних типах ПЛІС.

1. *Специфікація цифрового пристрою.* На цьому етапі розробляється загальна специфікація пристрою, визначається набір основних його функцій, сумісність з існуючими інтерфейсами, пристроями і технологіями. Визначаються фізичні обмеження пристрою, такі як напруга живлення, споживана потужність, габарити і тип корпусу ІС, кількість і призначення зовнішніх контактів. Результатом специфікації є системне подання цифрового пристрою.

2. *Функціональний і структурний опис пристрою.* На цьому етапі розробляється внутрішня архітектура пристрою. Проводиться декомпозиція системного опису на основні функціональні модулі, які можуть розроблятися незалежно один від одного. У свою чергу типи функціональних модулів визначають способи їх опису. Складні модулі задаються, як правило, поведінковим описом, у той час як стандартні і прості модулі – структурним описом. Сучасні програмні пакети САПР

пропонують різні методи початкового опису проектованих цифрових пристроїв:

- *графічне введення проекту*. До цього виду опису відносять ієрархічне впорядкування функціональної схеми пристрою з бібліотечних цифрових вузлів, якими можуть бути стандартні модулі, елементи, визначені розробником, або сторонні IP-компоненти. Складений таким чином графічний опис цифрового пристрою носить назву *блок-діаграми* (block diagram). Кожен елемент блок-діаграми у свою чергу може мати вкладений багаторівневий опис з елементами деталізації аж до логічних вентилів. Іншим поширеним графічним способом введення проекту цифрового пристрою є використання *діаграми станів* (state diagram). Цей спосіб найбільш прийнятний для опису поведінки послідовної логіки у вигляді цифрових кінцевих автоматів. Ще одним графічним способом опису є ГСА (граф-схема алгоритму), яка наочно відображує функціонування пристрою, але абсолютно не містить інформацію про його внутрішню структуру;

- *мовне введення проекту*. До мовних відносять методи опису розроблюваних цифрових пристроїв за допомогою HDL-мови. Наприклад, за допомогою мови VHDL проектувальник здатен описувати як функціонування пристрою, так і його структуру на основі бібліотечних стандартних модулів, своїх власних і сторонніх програмних IP-компонент. Мовний HDL-опис, на відміну від графічних способів, є легко змінним, технологічно незалежним, переносним способом введення проекту. Розроблені методи логічного синтезу з HDL-описів роблять мовний метод введення цифрових проектів одним з найпоширеніших на даний момент;

- *інші методи введення проекту*. Крім перерахованих способів введення проектів, різні фірми-розробники пакетів САПР цифрових пристроїв пропонують на вибір альтернативні варіанти, такі як складання таблиць істинності, графічний опис за тимчасовими діаграмами функціонування, використання мов програмування високого рівня (Pascal, C/C++). Останній варіант актуальний для проектування ЦОС-систем, поведінка яких легше описується за допомогою алгоритмічного подання. Потреба завдання як цифрової складової пристроїв, так і алгоритму їх

функціонування призвело до появи нових видів мовних описів, прикладом яких може служити мова SystemC.

Сучасні засоби проектування дозволяють розробнику використовувати різні варіанти введення проекту при проектуванні цифрових пристроїв. Так, при низхідному проектуванні на перших етапах можливе створення графічної блок-діаграми всього пристрою на рівні основних функціональних модулів. Самі модулі залежно від наявних можливостей можуть проектуватися з урахуванням їхньої специфіки функціонування. Пристрої управління легше описати за допомогою діаграм станів, стандартні інтерфейсні модулі – за допомогою IP-компонент, призначену для користувача комбінаційну та послідовну логіку – однією з HDL-мов, складні ЦОС-модулі – високорівневими мовами типу SystemC. Відсутність жорстких критеріїв на способи введення проектів залишають за розробником рішення про вибір прийнятного варіанта. Найчастіше невеликі проекти і проекти типу SoC можуть бути повністю реалізовані мовою HDL без використання інших способів введення проекту.

3. *RTL-синтез*. Після того як опис цифрового проекту складено, він піддається компіляції, при цьому всі модулі незалежно від способу їх початкового введення подаються у вигляді еквівалентних HDL-описів. Потім, провівши лексичний, синтаксичний і семантичний аналіз, повний HDL-опис пристрою транслюється в RTL-опис. RTL-синтез часто супроводжується побудовою керуючих графів потоків даних (CDFG, Control Data Flow Graph), які дозволяють оптимізувати елементи синхронної логіки, вирішуючи такі завдання, як планування обчислень, повторне використання ресурсів, структурування і розгалуження потоків даних. RTL-опис є деталізованим поданням потоків оброблюваних даних проектованого пристрою. Результат RTL-синтезу подається у вигляді схеми, що складається з RTL-примітивів, набір яких є унікальним для обраного засоба синтезу.

4. *Функціональне моделювання*. Моделювання застосовується для того, щоб завчасно визначити найбільш важливі експлуатаційні характеристики всього розроблюваного цифрового пристрою. У процесі моделювання на рівні регістрових передач досліджуються такі питання, як визначення правильності

функціонування основних вузлів пристрою, попереднє оцінювання витрат апаратури (обсяги синтезованих запам'ятовуючих пристроїв, кількість необхідних регістрів тощо). Метою моделювання є дослідження поведінки потоків передачі даних, які мають найбільший вплив на експлуатаційні параметри цифрового пристрою. Найчастіше як засоби моделювання використовують програмні пакети налагодження, що імітують поведінку цифрових пристроїв у часі. При цьому пристрій подається у вигляді набору RTL-примітивів, поведінка яких добре відома і для яких існує визначена програмна модель. За допомогою подачі деяких впливів на входи моделі можна спостерігати її реакції як на вихідних полюсах, так і на кожній з ліній міжз'єднань RTL-примітивів. Результати налагодження подаються на часових діаграмах модельованого пристрою. Сучасні програмні засоби моделювання дозволяють проектувальнику не тільки задавати конкретні значення на кожному полюсі RTL-моделі і аналізувати значення кожного полюса, але і автоматизувати цей процес за допомогою мовних алгоритмічних описів. Так, мова VHDL і засоби моделювання цифрових пристроїв широко застосовуються для автоматизації процесу функціонального моделювання. Для цього використовують поведінкову несинтезовану підмножину мови VHDL для задавання алгоритмів подачі вхідних наборів і аналізу реакцій модельованого пристрою. VHDL-опис подібного типу називають *testbench* (випробувальний стенд). Якщо в результаті проведеного функціонального моделювання виявилися невідповідності в поведінці цифрового пристрою, здійснюється повернення до попереднього етапу проектування, на якому проводиться корекція опису пристрою. У деяких випадках може знадобитися коригування всього технічного завдання на проектування, при якій кардинально переглядаються раніше поставлені завдання.

5. *Логічний синтез*. Після успішно проведеного функціонального моделювання переходять до наступного етапу проектування, на якому RTL-опис цифрового пристрою транслюється в опис на основі базових цифрових вузлів. Під час логічного синтезу RTL-опис поділяється на опис комбінаційних елементів і елементів пам'яті. Під логічним синтезом розуміють

процес трансляції елементів зберігання на базові синхронні (тригери) і асинхронні (засувки) елементи пам'яті, а комбінаційних елементів – на базові логічні вентилі. Комбінаційна логіка RTL-моделі подається як система булевих виразів, які оптимізуються відомими методами і алгоритмами [23].

6. *Фізичний синтез*. Наступним етапом проектування після проведеного логічного синтезу є фізичний синтез, який полягає в трансляції результатів логічного синтезу на технологічний базис ПЛІС. Результатом фізичного синтезу для програмованих ІС є контент запам'ятовуючого пристрою конфігурації. Основними вихідними даними для фізичного синтезу є конкретно обрана архітектура ПЛІС, яка визначає такі обмеження, як кількість логічних блоків і блоків введення/виведення, внутрішня організація кожного логічного блока, архітектура генераторів функцій та інших програмованих ресурсів. Серед апаратних обмежень проектувальник має право визначати також бажані часові характеристики розроблюваного пристрою, наприклад максимальне значення робочої частоти. Якщо результат фізичного синтезу не задовольняє одне з певних обмежень, проектувальник повинен переглянути зазначені вихідні дані для синтезу. Наведемо приклад фізичного синтезу для ПЛІС типу FPGA. Можна виділити чотири основні етапи синтезу: *розподіл* (partitioning); *розміщення* (placement); *трасування* (routing); генерування файла конфігурації.

Під *розподілом* розуміють процес декомпозиції логічного опису розроблюваного пристрою з метою його реалізації на програмованих ресурсах FPGA. На рис. 1.27 наведено приклад розподілу логічного опису 8-вхідного цифрового блока, що відтворює функцію виключного АБО над вісьмома бінарними значеннями, з урахуванням його реалізації на 4-вхідних LUT-блоків FPGA [1].

Результатом розподілу є набір конкретно обраних ресурсів ПЛІС, налагоджених на реалізацію функцій проектованого цифрового пристрою.

*Розміщення* – визначення конкретного фізичного стану використовуваних ресурсів FPGA для реалізації заданого цифрового пристрою. Вихідними даними для розміщення є результати розподілу і відповідність портів реалізованого

пристрою фізичним контактам ПЛІС (блокам IOB), яке описується за допомогою формату UCF (User Constraints File). Крім інформації про фізичні контакти, проектувальник має право визначати конкретне місцерозташування ресурсів FPGA, які будуть використані для розміщення, а також бажані часові обмеження. Наступним етапом після розміщення є *трасування,* яке полягає у визначенні налагоджень програмованих з'єднань між локальними ресурсами ПЛІС і блоками введення/виведення. Після успішного трасування формується файл, вміст якого контекст пам'яті конфігурації FPGA (рис. 1.28) [1].

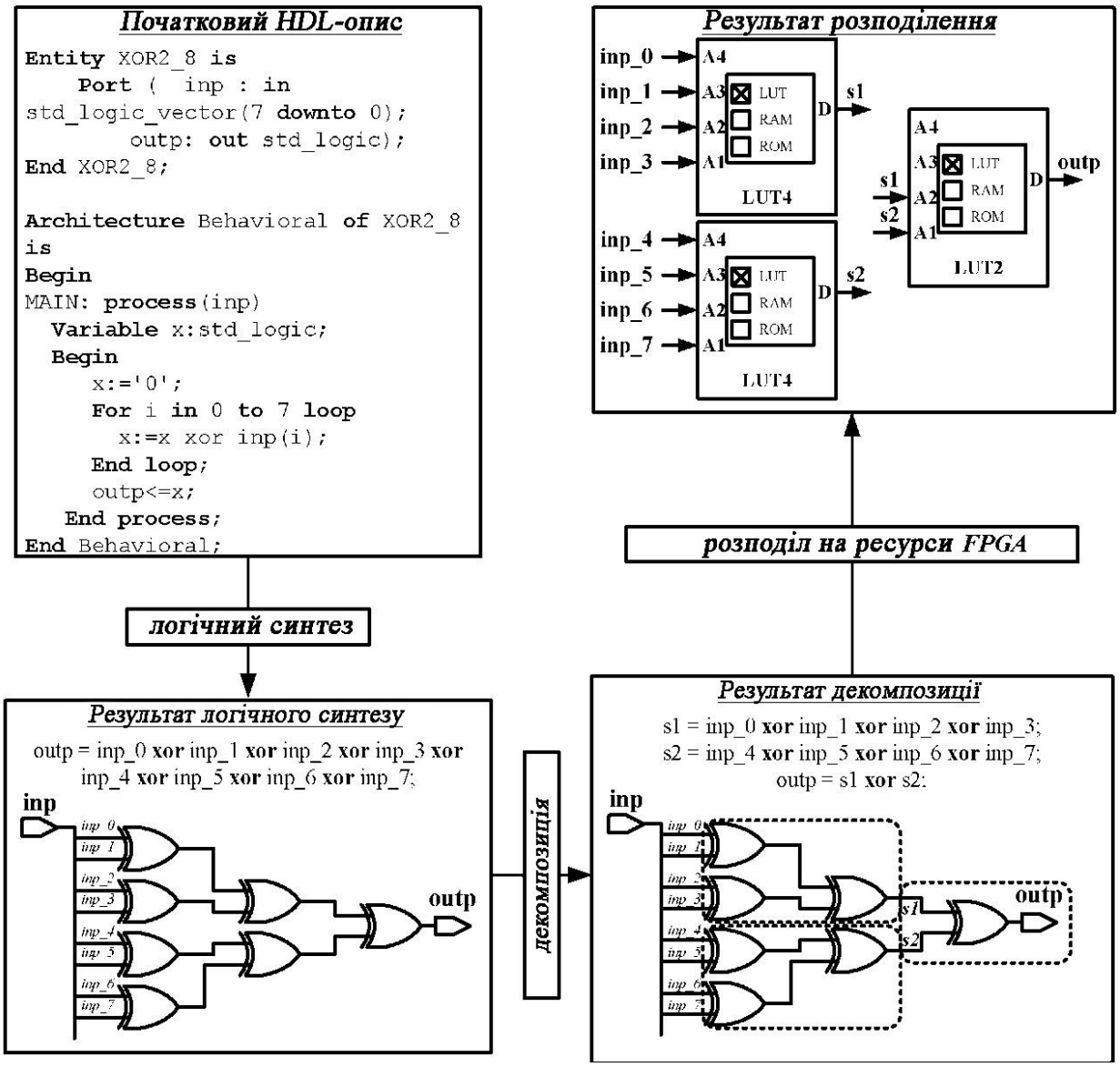



Рис. 1.27. Приклад процедури розподілу на ресурси FPGA

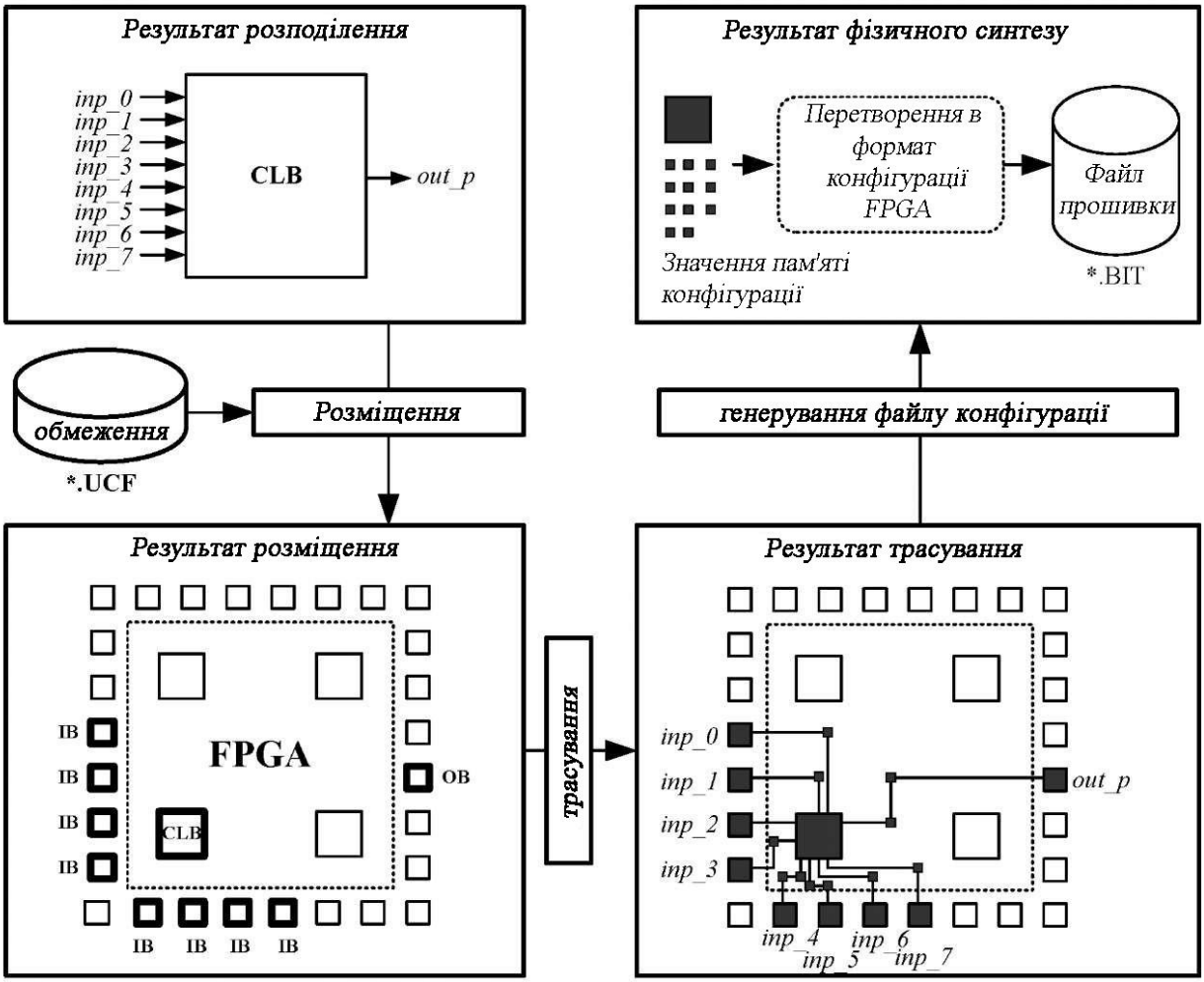



Рис. 1.28. Розміщення і трасування ресурсів FPGA

7. *Аналіз і часове моделювання*. Остаточні результати розміщення і трасування використовуються для аналізу підсумкових параметрів фізичного синтезу проектованого цифрового пристрою. Під аналізом розуміють оцінювання таких найважливіших характеристик, як апаратурні витрати на реалізацію цифрового проекту (наприклад кількість логічних блоків і блоків введення/виведення) і часові характеристики (затримки поширення сигналів на логічних блоках, транспортні затримки на з'єднувальних лініях) тощо. Обчислені параметри можуть бути використані для складання параметричного HDL-опису цифрового пристрою з метою детального аналізу його поведінки, максимально наближеного до реального. При цьому можуть бути використані TestBench-модулі, розроблені на етапі функціонального моделювання. Крім перерахованих параметрів, можливе оцінювання перемикальної активності цифрових елементів

реалізованого пристрою з подальшим обчисленням середнього значення споживаної енергії ПЛІС і тепла, що виділяється. Якщо результати проведеного аналізу повністю задовольняють поставленим критеріям на функціонування, то переходять до наступного етапу проектування – до реалізації пристрою.

8. *Реалізація*. Під реалізацією проекту цифрового пристрою на ПЛІС розуміють використання результатів фізичного синтезу для програмування ЗП конфігурації. По завершенню програмування і подачі сигналу скидання мікросхема FPGA починає відтворювати функціонування спроектованого цифрового пристрою. Як правило, FPGA, призначені для реалізації цифрових проектів, входять до складу прототипних плат, що полегшує процес верифікації пристрою і значно прискорює підготовку прототипу до виробничого випуску.

9. *Верифікація*. Після того як цифровий проект реалізований на ПЛІС, розробник приступає до процедури верифікації експериментального пристрою. З урахуванням того, що всі попередні етапи проектування були виконані успішно, завжди є ймовірність того, що в реалізованому проекті присутні помилки, які можуть проявитися під час експлуатації пристрою, а це у свою чергу може призвести до небажаних наслідків. Виконання процедури верифікації значно підвищує ймовірність випуску надійної продукції, однак повністю не гарантує її безвідмовне функціонування. Для проведення верифікації цифрового пристрою необхідна наявність спеціалізованого обладнання та програмного забезпечення, які дозволяють генерувати тестові впливи для пристрою і аналізувати його вихідні реакції. Наявність зовнішніх апаратних модулів тестування цифрових пристроїв робить процедуру верифікації ускладненою і дорогою. Зважаючи на це, найбільш прийнятним способом тестування цифрових пристроїв, реалізованих на ПЛІС, став метод вбудованого тестування, при якому разом з проектом програмується і тестове обладнання (рис. 1.29 [1]).

У цьому випадку проектувальник може використовувати як готові рішення, так і визначати свої власні тестові процедури. Використання такого підходу разом зі стандартним обладнанням типу цифрових осцилографів і логічних аналізаторів прискорює верифікацію.

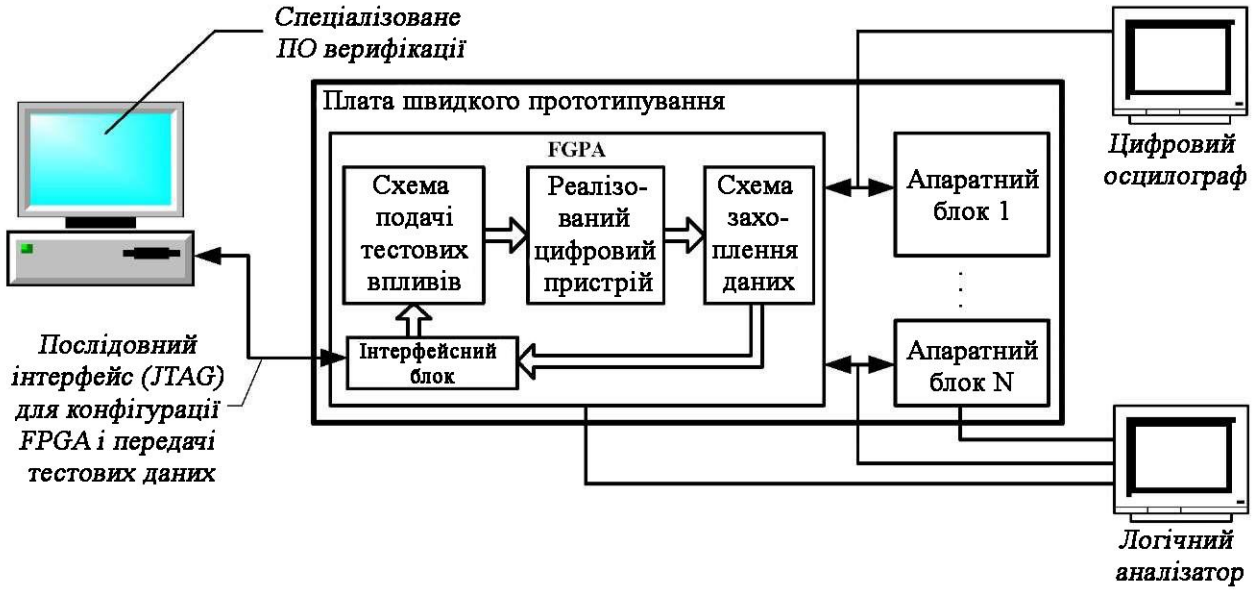



Рис. 1.29. Схема верифікації цифрових проектів для ПЛІС

Наведені етапи проектування цифрових пристроїв є якоюсь мірою узагальненими для різних технологій реалізації. Кінцевий результат проектування може бути використаний для ПЛІС або ПЛІС може виступити як верифікаційна платформа, а кінцевим результатом виявиться замовна НВІС. При цьому повторюються кінцеві етапи проектування починаючи з фізичного (транзисторного) синтезу і закінчуючи верифікацією проекту.

## 1.6. Реконфігуровані цифрові системи

Для реалізації обчислювальних алгоритмів на цифрових пристроях застосовуються два основні підходи:

1) використання спеціалізованої апаратури;
2) використання мікропроцесорного підходу.

Різниця цих двох підходів полягає у формуванні та управлінні потоками даних, що обробляються. Так, спеціалізована апаратура (часто у вигляді замовної НВІС) являє собою набір фіксованих апаратних блоків, які розробляються з метою обробки даних, що надходять за заздалегідь визначеним алгоритмом. Набори апаратних блоків, залучених до реалізації алгоритмів, формують потоки обробки даних. Для апаратних рішень ці потоки є фіксованими і можуть бути змінені тільки при

повному перепроектуванні пристрою. До переваг фіксованої апаратури можна віднести такі показники, як висока швидкодія, малі апаратурні витрати і низьке енергоспоживання.

Мікропроцесорний підхід, на відміну від розглянутого вище, має гнучку систему управління потоками обробки даних. МП має простий і зручний апаратний блок у вигляді ЗП інструкцій. Пристрій управління послідовно вибирає інструкції, виконує їх і тим самим впливає на вибір необхідних потоків обробки даних. Гнучкість такого підходу обмежена наявною фіксованою апаратурою. У разі реалізації алгоритмів, не передбачених фіксованою апаратурою, використовують послідовність наявних інструкцій для відтворення необхідного потоку обробки даних. Це спричиняє втрати продуктивності всієї системи. Час, який витрачає пристрій управління на вибірку, декодування і виконання кожної інструкції, значно зростає при програмній реалізації алгоритмів.

Розвитку альтернативних підходів для реалізації алгоритмів сприяла поява ПЛІС. Надалі такі підходи отримали назву *реконфігурованої апаратури.* Основна ідея, що стала архітектурною основою цифрових реконфігурованих систем, полягає в повторному (багаторазовому) використанні апаратних ресурсів для реалізації різних потоків обробки даних. Реконфігуровані системи у вигляді ПЛІС мають позитивні якості двох стандартних підходів: високу продуктивність фіксованої апаратури і гнучкість мікропроцесорних систем.

Наприклад, проектувальник може реалізувати IP-компоненту софт-процесора на FPGA, що дозволить при експлуатації змінювати його функціонування за допомогою зміни виконуваного коду. У разі, якщо проектувальник не досяг бажаних результатів у швидкодії або обсязі виконуваного коду, він може видозмінити безпосередньо саму IP-компоненту, що призведе до змін в апаратній складовій софт-процесора. Внісши зміни до HDL-опису софт-процесора, повторно синтезуючи і програмуючи FPGA, проектувальник здійснює так звану процедуру *статичної реконфігурації.*

*Динамічна ре конфігурація,* на відміну від статичної, має на увазі зміну апаратної складової безпосередньо при функціонуванні цифрової системи. Так, маючи кілька образів

пам'яті конфігурації ПЛІС, які реалізують різні цифрові пристрої, можна домогтися повторного використання апаратних ресурсів або динамічної реконфігурації.

Стандартні типи ПЛІС, такі як FPGA, мають послідовну статичну SRAM-пам'ять конфігурації. На час, відведений завантаженню контексту з EEPROM, FPGA припиняють своє функціонування. Після завантаження пам'яті конфігурації на НВІС FPGA подається сигнал ініціалізації, який активізує процедуру налагодження всіх необхідних конфігураційних блоків. Подібна процедура налагодження ПЛІС, що займає досить великий часовий відрізок (до десятків мілісекунд), негативно позначається як на функціонуванні, так і продуктивності цифрових систем. Методи, покладені в основу рішень щодо усунення таких небажаних ефектів, отримали назву *реконфігурації на льоту* (run-time reconfiguration).

Традиційні типи ПЛІС мають одноконтекстний ЗП конфігурації, який може бути тільки цілком завантажений ззовні по послідовному інтерфейсу. Для зміни хоча б одного біта конфігурації подібний ЗП має бути повністю перезаписаний. Для реалізації реконфігурації на льоту одноконтекстних ПЛІС використовують ЗП з довільним доступом. Такий підхід називається *частковою реконфігурацією* (partial reconfiguration) і не вимагає зупинки функціонування всієї ПЛІС. Прикладами подібних ПЛІС з можливістю часткової реконфігурації можуть служити FPGA Atmel 40K і Xilinx 62xx. Однак ПЛІС з одноконтекстним ЗП не дозволяють швидко і ефективно змінювати повністю свою конфігурацію.

Іншими прикладами динамічно реконфігурованих систем є багатоконтекстні *FPGA* (multicontext FPGA), для яких кожен програмований елемент конфігурації представлений декількома бітами. При цьому кожен біт належить одному контекстному рівню, що дозволяє швидко, без завантаження ззовні і великих часових витрат змінювати апаратне налагодження FPGA. Дана технологія знайшла своє застосування в реконфігурованій системі Chameleon [50], НВІС CS2000 RCD, яка має чотири контекстних рівні. Використовуючи подібні системи, з'являється можливість додатково завантажити нові контексти ззовні, не перериваючи функціонування ПЛІС, а час перемикання контекстів оцінюється наносекундами.

Ще одним прикладом динамічних методів реконфігурування апаратури може служити так звана конвеєрна реконфігурація [51], яка використовує принципи часткової реконфігурації.

Для реалізації обчислювальних систем на основі динамічно реконфігурованої ПЛІС використовують такі основні принципи, які були визначені в роботі [52]:

1. Елементом реконфігурованої системи є переміщуваний *апаратний об'єкт,* який являє собою сконфігуровану частину ПЛІС, яка б відтворювала задану функцію. Апаратний об'єкт може бути розміщений у вільних ресурсах ПЛІС, при цьому його положення не впливає на його функціонування.

2. Кожен апаратний об'єкт має бути наділений універсальним інтерфейсом для забезпечення зв'язку з іншими об'єктами. Це дозволить будувати великі обчислювальні пристрої за модульним принципом, при якому кожен об'єкт є заміщуваним і повторно використовуваним.

3. Набір апаратних об'єктів оточений інтерфейсною структурою, що складається зі спеціальних об'єктів, що дозволяють підключатися до стандартних периферійних пристроїв, таких як ОЗП, МП тощо.

Реалізація повноцінної реконфігурованої системи неможлива без участі програмного оточення, яке вирішує завдання планування обчислювального процесу та управління реконфігурованою апаратурою. Так, ПЗ вирішує, які необхідні апаратні об'єкти і коли слід використовувати. Завантаження і вивантаження об'єктів з поточного апаратного контексту, динамічне трасування з'єднань між об'єктами також визначається програмним забезпеченням.

Важливим параметром реконфігурованих систем є «зернистість», яка визначається як мінімальний розмір одного апаратного об'єкта. «Дрібнозернисті» структури, в основі яких можуть лежати генератори функцій від декількох змінних, використовуються для біт-орієнтованих обчислень. Елементами «великозернистих» структур можуть бути такі апаратні блоки, як налагоджені арифметико-логічні пристрої, які оперують багаторозрядними аргументами. Такі елементи є більш придатними для реалізації мікропроцесорних потоків обробки даних.

Використання реконфігурованих систем дозволяє позбавлятися від недоліків мікропроцесорного підходу, при якому виконання певного алгоритму бере на себе не фіксована апаратура і програмний код, а тільки конфігурована апаратура. Прикладом збільшення продуктивності обчислювальних алгоритмів може служити реконфігурована система, що побудована на FPGA Xilinx Virtex XCV1000 і реалізує алгоритм шифрування даних Serpent Block Cipher [53]. Швидкість шифрування даних на FPGA у 18 разів перевершує швидкість цього ж алгоритму, але реалізованого для мікропроцесора Pentium Pro. При цьому системна частота синхронізації для двох варіантів була обрана 200 МГц.

Реконфігуровані обчислення мають велике значення в таких дослідницьких галузях, як комп'ютерні архітектури та програмне забезпечення обчислювальних систем. Реконфігуровані системи мають гнучкість, властиву програмним рішенням, і продуктивність спеціалізованої апаратури. Залежно від вирішуваного завдання подібні системи динамічно адаптують свою апаратну складову для ефективного виконання заданих алгоритмів. Повторне використання апаратури також дозволяє швидко впроваджувати на ринок нові алгоритми і протоколи без перепроектування і перевиробництва цифрових пристроїв.

Крім перерахованих якостей реконфігурованих систем слід відзначити можливість побудови саморемонтованих цифрових пристроїв, які, не перериваючи свого функціонування, можуть динамічно заміщувати апаратні блоки, які відмовили, на справні резервні.

**Контрольні питання**

1. Що таке вбудована система?
2. У чому полягає специфіка вбудованих систем у загальному класі комп'ютерних систем?
3. Як можна класифікувати вбудовані теми?
4. Які класи завдань вирішують вбудовані комп'ютерні системи?
5. Що таке ПЛІС? У чому її відмінності від мікроконтролерів і звичайних інтегральних схем?

6. Яка архітектура типової багаторівневої системи управління?

7. У чому відмінність програмувального мікроконтролера від комп'ютера загального призначення та промислового комп'ютера?

8. Які принципові розходження в структурі універсальних і вбудованих мікропроцесорних систем?

9. Що входить до складу типової вбудованої системи управління?

10. Чим відрізняється процесор від мікропроцесора?

11. Що таке мікропроцесорний комплект?

12. Наведіть основні характеристики мікропроцесорів.

13. Наведіть класифікаційні ознаки мікропроцесорів.

14. Наведіть основні принципи архітектури фон Неймана.

15. У чому полягає принципова відмінність гарвардської архітектури від архітектури фон Неймана?

16. Наведіть переваги та недоліки архітектур фон Неймана та гарвардської.

17. Що являє собою принцип ЗМ при побудові мікропроцесорних систем?

18. Принцип модульної організації МПС.

19. Магістральний спосіб обміну інформацією в МПС.

20. Мікропрограмна реалізація управління в МПС.

21. Що являє собою принцип агрегатування?

22. Організація одно- і багатомагістральних мікропроцесорних систем.

23. Типова структура тришинної МПС.

24. Призначення схеми синхронізації та початкового скидання.

25. Організація доступу до пам'яті МПС.

26. Організація доступу до пристроїв введеня/виведення.

27. Призначення таймера в МПС.

28. Організація прямого доступу до пам'яті в МПС.

29. Призначення контролера переривань у МПС.

# 2. ПРОЕКТУВАННЯ БАЗОВИХ КОМПОНЕНТ ВБУДОВАНИХ ПРИСТРОЇВ

Лінгвістичне забезпечення САПР є найбільш затребуваним інструментарієм для сучасних проектувальників вбудованих цифрових пристроїв. Проектування цифрового пристрою за допомогою схемотехнічного редактора вже є практично неактуальним і може застосовуватися тільки на кінцевих стадіях компіляції готового проекту. На перший план серед інструментів проектувальників виходять спеціалізовані мови, що дозволяють описувати не тільки структуру, але і функціонування цифрових компонент на різних рівнях абстракції.

Серед усього різноманіття мов проектування цифрової апаратури можна чітко виділити мову VHDL (Very high speed integrated circuits Hardware Description Language), яка завдяки своїй універсальності дозволяє описувати як структуру і функціонування найпростішого логічного елемента, так і багатопроцесорної обчислювальної системи спеціального призначення. Поширення VHDL і підтримка провідними виробниками систем автоматизованого проектування робить його де-факто стандартною мовою проектування цифрових пристроїв.

У цьому розділі розглядаються питання, пов'язані з проектуванням базових компонент вбудованих цифрових пристроїв: комбінаційних схем, послідовних схем, синхронних схем, реєстрових схем, схем синхронних кінцевих автоматів, схем запам'ятовуючих пристроїв різних організацій і призначень. Крім того, наводяться приклади типових помилок проектних описів і можливі методи їх усунення.

Описи наведених компонент представлені мовою VHDL, а схемна реалізація компонент – як результати їх RTL- і технологічного синтезу для ПЛІС.

З урахуванням наведених обмежень розглянемо спочатку стилі проектних описів, доступні проектувальнику при використанні мови VHDL в процесі реалізації базових компонент і закінчених цифрових пристроїв для ПЛІС типу FGPA.

## 2.1. Стилі проектних описів

Як було зазначено вище, лінгвістичні можливості мови VHDL дозволяють описувати цифрові пристрої різного ступеня складності на різних рівнях абстракції. У свою чергу всі VHDL-описи класифікуються на дві групи: *синтезовані* і *несинтезовані*.

Під синтезованим описом цифрового пристрою будемо розуміти такий вихідний текст мовою VHDL, який може бути сприйнятий програмною системою логічного синтезу з подальшим отриманням RTL-моделі цього пристрою.

Несинтезований опис цифрового пристрою являє собою вихідний текст мовою VHDL, що містить несинтезовані оператори і лінгвістичні конструкції. При цьому сам вихідний текст мовою VHDL є коректним з точки зору граматики та синтаксису. Як правило, несинтезовані VHDL-описи використовуються як параметричні або тестові моделі (testbench).

У свою чергу синтезовані описи можуть проектуватися чотирма основними стилями: *структурним, поведінковим, псевдоструктурним* і *змішаним.* Розглянемо ці стилі докладніше.

*Структурний* стиль проектного опису полягає в декларації стандартних бібліотечних і/або призначених для користувача компонент **(component)** і описи їх міжз'єднань за допомогою сигналів **(signal)** (рис. 2.1) [1].

Структурний стиль часто застосовується на практиці для опису (збірки) верхньої ієрархічної компоненти всього проекту (top level design), що дає наочне уявлення про те, з яких компонент складається розроблюваний пристрій і як вони з'єднані між собою. Однак такий стиль опису не надає інформацію про функціонування кожної компоненти і всього проекту в цілому. Незважаючи на це, структурний стиль опису часто застосовується системами САПР для автоматичного створення RTL- і технологічних описів після успішно проведеного синтезу для можливості здійснення функціонального і параметричного моделювання.

Також структурний стиль дозволяє описувати ітераційні цифрові схеми, що складаються з однакових компонент і повторюваних комбінацій їх міжз'єднань. Прикладами ітераційних схем можуть служити суматори, помножувачі,

реєстрові схеми, багатовходові логічні елементи тощо. Для опису подібних цифрових схем в мові VHDL можна використовувати оператор **for generate** і **generic**-параметр для визначення кількості ітерацій (рис. 2.2) [1].

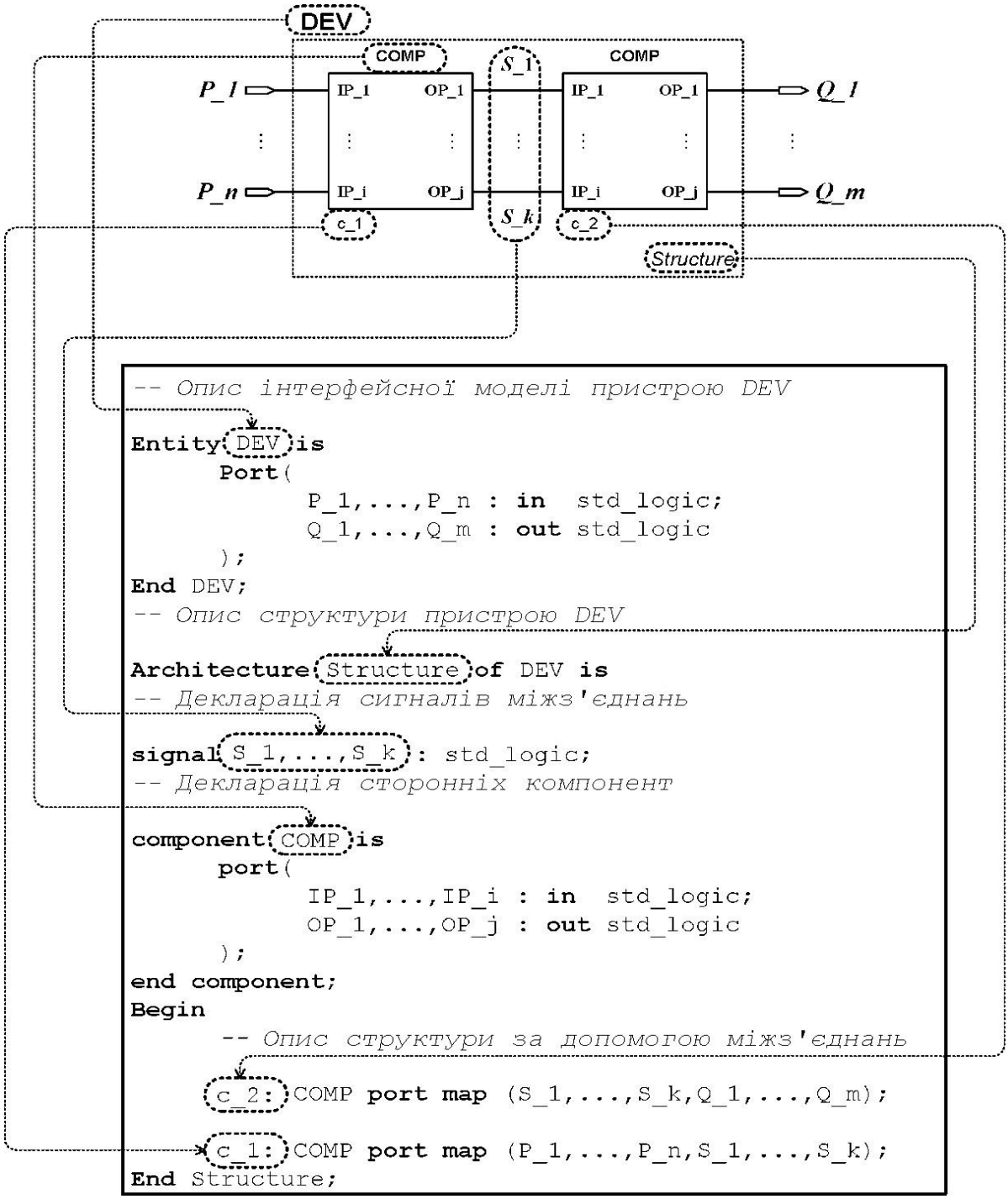



Рис. 2.1. Структурний стиль опису цифрового пристрою DEV

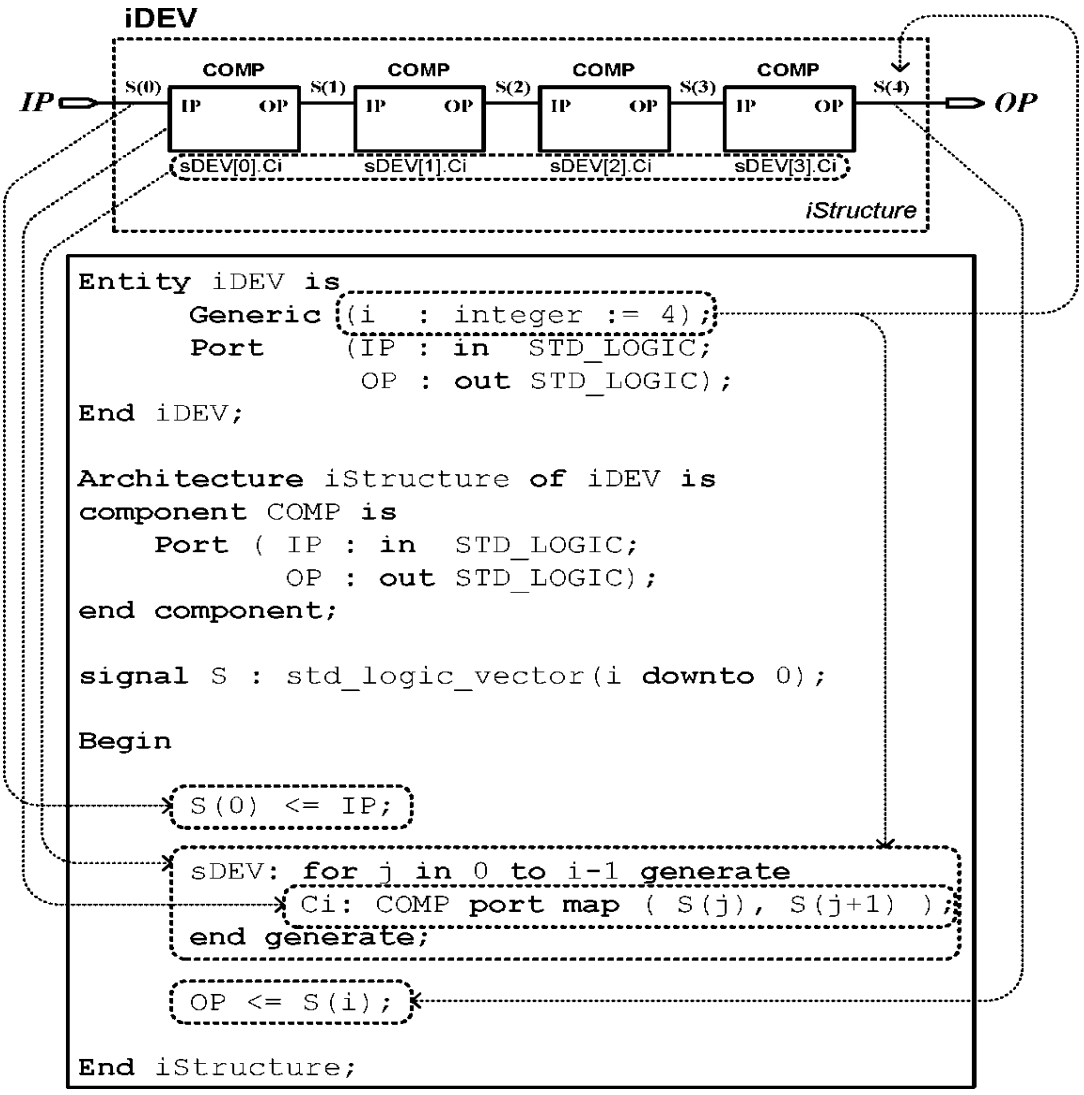



Рис. 2.2. Структурний стиль опису ітераційної схеми пристрою iDEV

*Поведінковий* стиль проектного опису заснований на алгоритмічному поданні функціонування проектованого пристрою. Для складання подібного опису частіше застосовуються так звані послідовні оператори, застосування яких можливе виключно в блоках типу **process.** Алгоритмічний опис всередині процесів можливий при використанні як сигналів, так і змінних **(variable),** які є локальними послідовними об'єктами процесу, у якому вони оголошені.

Деякі паралельні оператори також можуть бути застосовані для складання поведінкового опису проектованого цифрового пристрою. Найчастіше використовуваними паралельними операторами є оператори **with-select** і **when-else** – аналоги послідовних операторів **case** і **if-else** відповідно.

На відміну від структурного стилю поведінковий стиль опису практично не несе інформації про структуру розроблюваної схеми, але дає можливість оцінити її функціонування (рис. 2.3) [1].

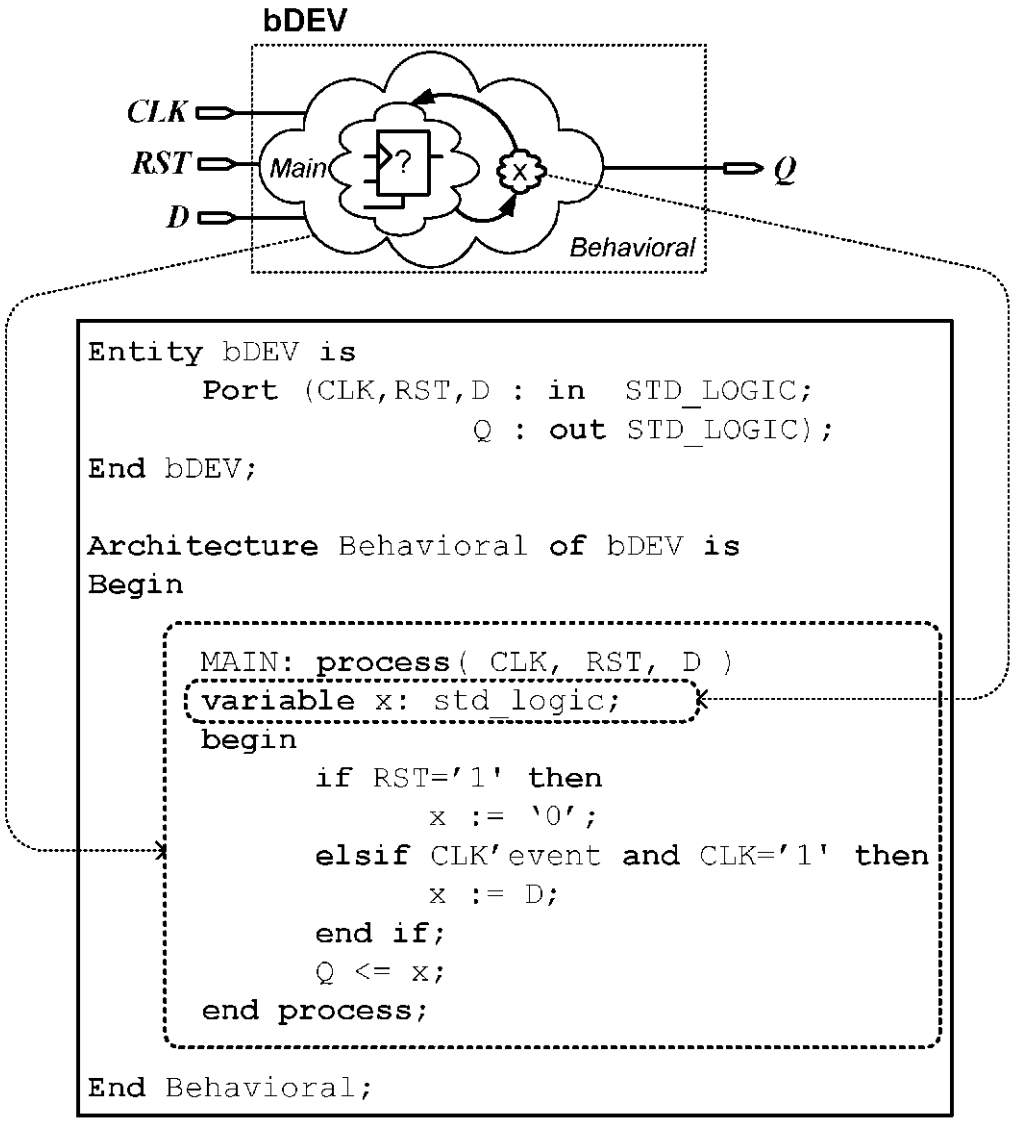



Рис. 2.3. Поведінковий стиль опису пристрою bDEV

*Псевдоструктурний* стиль проектного опису є описом окремих компонент схеми розроблюваного пристрою за допомогою процесів, взаємодія яких забезпечується сигналами. На відміну від структурного опису, псевдоструктурний опис несе інформацію як про складові частини розроблюваного пристрою, так і функціонування кожної частини окремо. Таким чином, опис структури та функціонування пристрою розміщується в одному

архітектурному блоці VHDL-проекту. Подібний стиль опису добре застосовний для проектування цифрових пристроїв, що містять невелику кількість складових частин (рис. 2.4) [1].

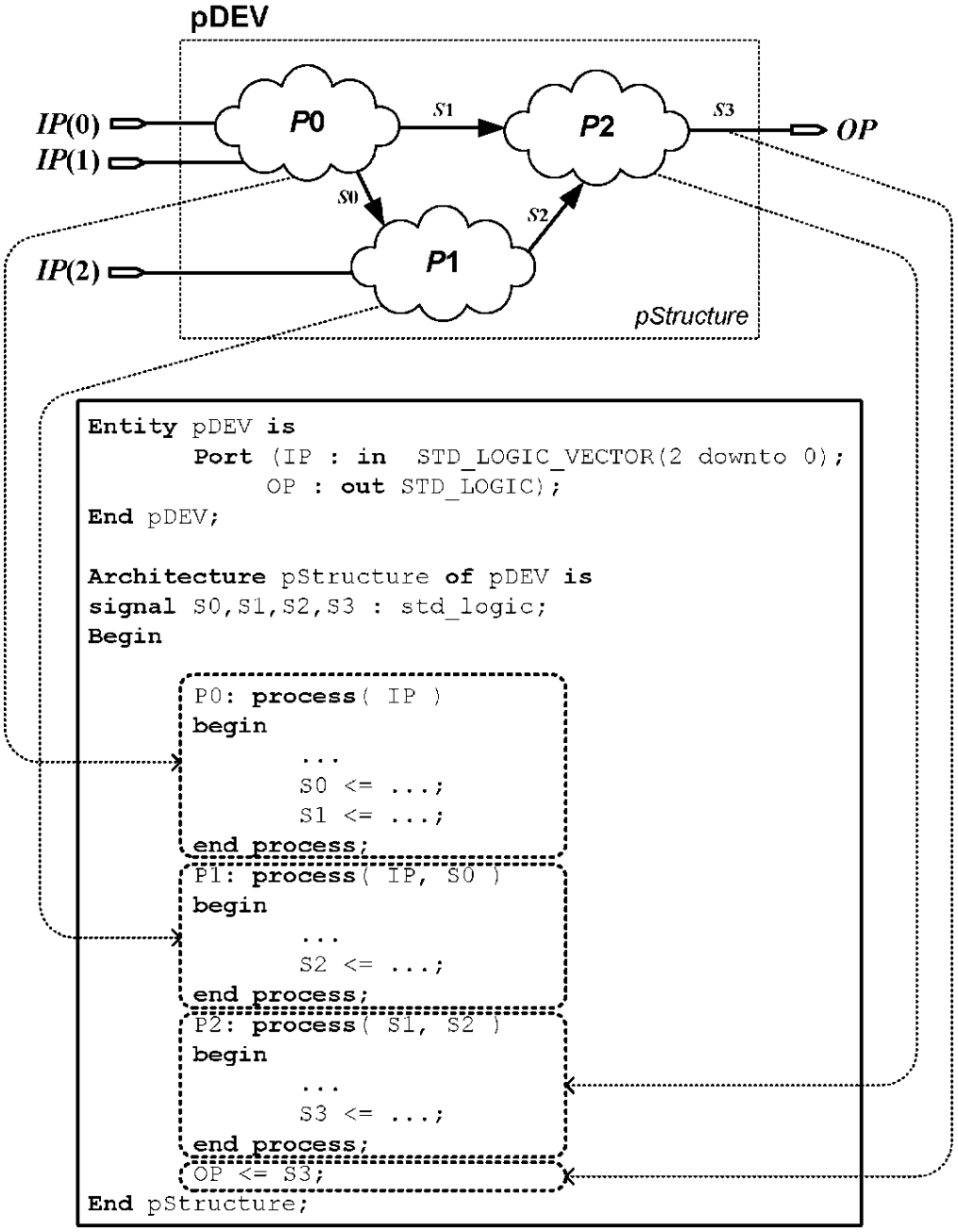



Рис. 2.4. Псевдоструктурний стиль опису пристрою pDEV

Псевдоструктурні описи можуть застосовуватися для проектування нескладних цифрових пристроїв, для яких окремими процесами описується їхня послідовна складова (наприклад реєстрова пам'ять), а іншими процесами – комбінаційна логіка. При використанні псевдоструктурного

стилю опису слід пам'ятати, що взаємодія процесів здійснюється тільки за допомогою сигналів і/або портів. Для одного сигналу може бути тільки один процес, який змінює його значення. Такий процес називається **драйвером** відповідного сигналу. При складанні синтезованих описів слід дотримуватися правила: *в одного сигналу може бути тільки один драйвер.* В іншому випадку буде мати місце спроба опису монтажного з'єднання, яка не підтримується багатьма засобами синтезу.

*Змішаний* стиль проектного опису є комбінацією перерахованих вище стилів і часто застосовується на практиці. Так, один проект може містити структурний опис стандартних і користувальницьких компонент разом з описом функціонування складових частин проектованого пристрою за допомогою процесів (рис. 2.5) [1].

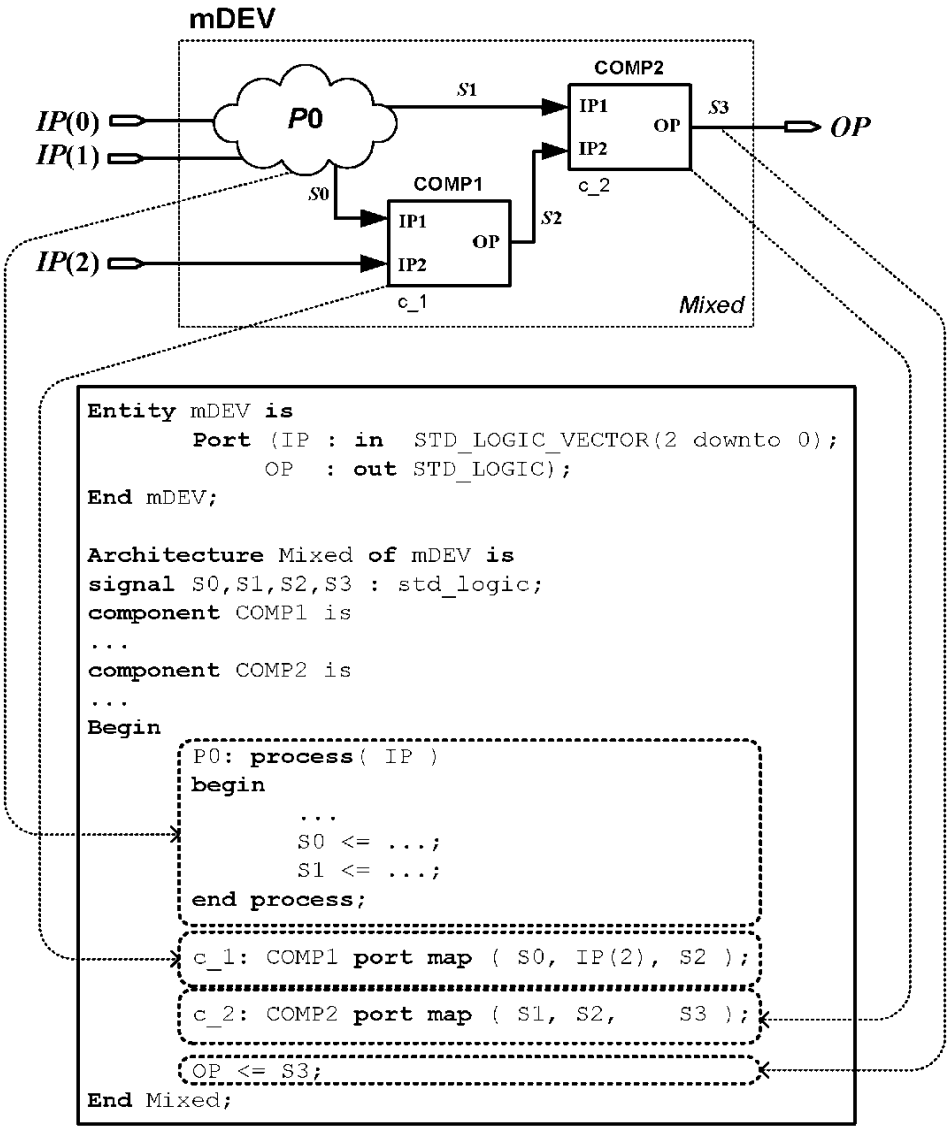



Рис. 2.5. Змішаний стиль опису пристрою mDEV

Розглянуті стилі проектних описів не є обов'язковими у використанні, але дають уявлення розробникам про можливості мови VHDL, які можуть застосовуватися в проектуванні реальних цифрових пристроїв. Конкретний стиль проектних описів формується розробниками виходячи з їхнього особистого досвіду, обмежень використовуваних засобів синтезу, особливостей цільової технології реалізації пристрою, регламенту і традицій попередніх розробок тощо. Однак під час проектування цифрових пристроїв виникають ситуації, при яких є необхідність змінювати (адаптувати) використаний стиль опису для вирішення завдань оптимізації витрат апаратури або проблем синхронізації окремих модулів пристрою.

Наприклад, синтезований проектний опис складної цифрової схеми може бути поданий як поведінковий VHDL-опис за допомогою одного процесу, який наочно розкриває суть функціонування схеми. Але результат RTL-або технологічного синтезу такого опису може бути не очевидним навіть для досвідчених розробників через багато причин: високий рівень абстракції вихідного опису, множинність конфігураційних налагоджувань засобів синтезу, фізичні обмеження цільової технології тощо.

Інформація, отримана після аналізу результатів проведеного синтезу, може підштовхнути розробника до зміни вихідного проектного опису, наприклад до необхідності провести функціональну декомпозицію схеми пристрою з застосуванням бібліотечних компонент, результати синтезу яких очевидні не тільки розробнику, а й автоматизованим системам проектування.

Для спроби вироблення універсального стилю проектних описів цифрових пристроїв будемо виходити з таких обмежень. Проектовані пристрої будуть реалізовані на ПЛІС типу FGPA, що мають такі ресурси, як CLB-блоки, які дозволяють відтворювати довільну комбінаційну і послідовну логіку.

Проектований цифровий пристрій подається як абстрактна модель [54], яка дозволяє розглядати його з позиції тракту обробки даних **(datapath),** пристрій управління, який контролює тракт, і схеми синхронізації, що забезпечують вироблення тактових сигналів, що погоджують роботу всіх компонент пристрою (рис. 2.6) [1].

У поданій моделі чітко виділяються такі базові компоненти, що входять до складу тракту обробки даних, пристрою

управління та схем синхронізації: вхідні та вихідні буфери підсилювачів; тристабільні буфери; асинхронні елементи пам'яті (засувки, від англ. latches); синхронні елементи пам'яті (тригери); комбінаційні пристрої.

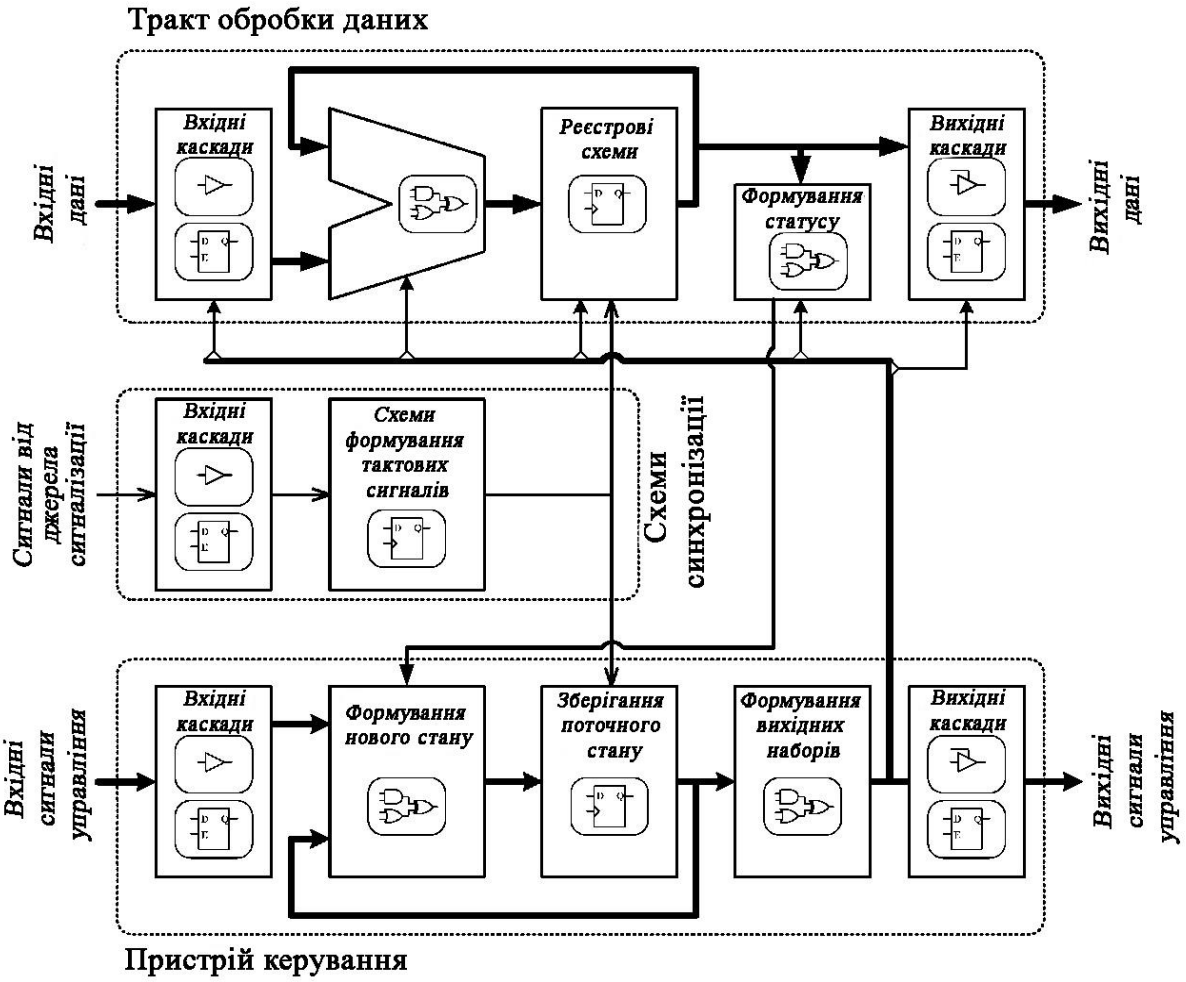



Рис. 2.6. Абстрактна модель цифрового пристрою

Розглянемо, як можна описувати мовою VHDL перераховані вище базові компоненти, які в подальшому будуть використані для проектування більш складних компонент і готових цифрових пристроїв.

## 2.2. Проектування буферних елементів

Буферні елементи забезпечують реалізацію інтерфейсних ліній цифрового пристрою, за якими можливі приймання та

передача сигналів від підключених пристроїв. Розрізняють такі типи буферних елементів: вхідні буфери-підсилювачі, вихідні буфери-підсилювачі, вихідні буфери-підсилювачі з трьома станами, двонаправлені буфери-підсилювачі.

Для можливості опису і проектування відповідних буферних елементів в мові VHDL передбачені такі типи портів **(port),** що підтримують синтезовану підмножину **std_logic**:

— **in std_logic:** вхідний буфер-підсилювач;

— **out std_logic:** вихідний буфер-підсилювач, у тому числі і тристабільний буфер;

— **buffer std_logic:** вихідний буфер-підсилювач з можливістю читання переданого значення (на відміну від нього, значення порту типу **out** прочитати неможливо);

— **inout std_logic:** двонаправлений буфер-підсилювач.

Розглянемо приклад проектного опису та реалізації цифрового пристрою PORTS, що має однорозрядні порти таких призначень: IP – вхідний порт даних; OP, NOP і BOP – вихідні порти даних; ZOP – вихідний порт даних з можливістю перемикання в третій стан; *Z* – вхідний порт управління станом порту ZOP. Нехай сигнал зі входу IP безпосередньо транслюється на вихідні порти OP, NOP і BOP, а на порт ZOP – з урахуванням значення вхідного сигналу Z.

VHDL-опис подібного цифрового пристрою з використанням перших трьох типів можна подивитися в дод. 1 (лістинг 2.1).

Як видно з поданого прикладу, значення вхідного сигналу з порту IP транслюється без змін на вихідні порти OP і BOP. З урахуванням того, що порт BOP оголошений як порт типу **buffer**, його значення можна прочитати і транслювати на вихідний порт NOP (NOP <= BOP), що неможливо здійснити зі значенням сигналу порту OP (NOP <= OP).

Поведінку вихідного тристабільного порту ZOP опишемо за допомогою оператора **when-else.** За наявності низького рівня сигналу на вхідному порту Z (Z = '0') сигнал з вхідного порту *IP* транслюється без змін на вихідний порт ZOP. В іншому випадку (Z = '1') порт ZOP перемикається в третій стан (ZOP <= ' Z').

На рис. 2.7 подано результат технологічного синтезу цифрового пристрою PORTS для FGPA класу Spartan-3E [1].

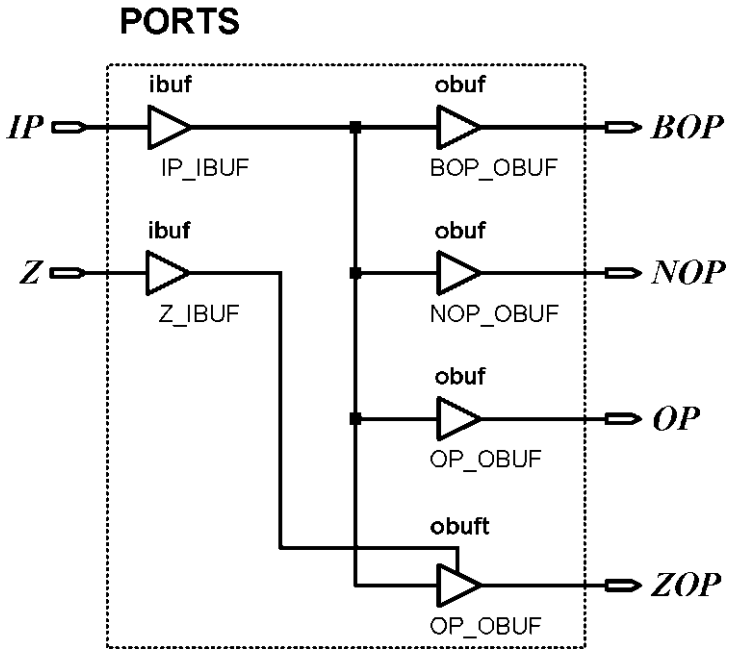



Рис. 2.7. Технологічна модель пристрою PORTS

Слід зазначити, що оператор присвоювання значення сигналам (<=) призводить до синтезу вхідних і вихідних буферних схем тільки при реалізації компоненти найвищого рівня ієрархії. В іншому випадку буферні схеми не реалізуються, а оператор присвоювання при цьому описує з'єднання провідних ліній.

Для реалізації наведеного прикладу компоненти PORTS для FPGA класу SPARTAN-3E буде використано шість блоків введення/виведення IOB, два з яких будуть сконфігуровані як вхідні буферні елементи ibuf, три - як вихідні буферні елементи obuf, і тільки один як буферний елемент з третім станом obuft.

Припустимо, що у використаному операторі **when-else** з наведеного прикладу відсутній операнд альтернативної умови **else**, і запис набуває вигляду ZOP <= IP **when** Z = '0';. Якщо значення сигналу на керуючому порту Z дорівнює "0", то відбувається трансляція сигналу з порту IP на вихідний порт ZOP, а результатом синтезу будуть два буферних елементи. При Z = '1' оператор присвоювання «не спрацьовує» або значення сигналу на вихідному порту ZOP не змінюється, а продовжує утримувати значення сигналу вхідного порту IP при останньому виконанні умови Z = '0'. Таким чином, оператор **when** описує поведінку елемента пам'яті разом з вихідним буферним елементом. Результатом технологічного синтезу такого оператора

є асинхронний D-тригер, керований низьким рівнем сигналу дозволу (рис. 2.8) [1].

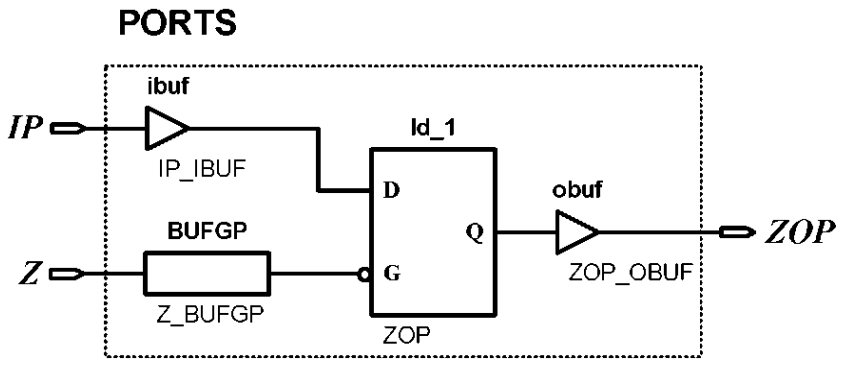



Рис. 2.8. Результат технологічного синтезу портів IP, Z та ZOP

Як видно з рис. 2.8, результатом синтезу вихідного порту ZOP є D-засувка ld 1 і вихідний буфер obuf , що входять до складу одного блока IOB. Оскільки керування засувкою вимагає підключення до глобальної мережі ліній синхронізації FPGA, то вхідним буфером для сигналу порту Z є технологічний примітив BUFGP, по суті також є вхідним буфером-підсилювачем [55].

Розглянемо приклад опису та реалізації двонаправленого порту, схема якого базується на використанні тристабільного буферного елемента. Розглянемо приклад реалізації цифрового пристрою IOPORT, який має такі порти: Di – вхідний порт даних, Do – вихідний порт даних, Dio – двонаправлений порт даних, MODE – вхідний порт керуючого сигналу. За умови MODE = "0" порт Dio є вихідним портом, що транслює сигнал зі входу Di, в іншому випадку за умови MODE = '1' порт Dio є вхідним портом, значення сигналу з якого транслюється на вихідний порт Do.

VHDL-опис такого цифрового пристрою можна подивитися в дод. 1 (лістинг 2.2).

Оскільки один фізичний порт Dio може виконувати роль портів різних типів, то необхідно мультиплексування режимами функціонування цього порту, яке забезпечується двома парафазними значеннями керуючого сигналу MODE. На рис. 2.9 подано результат технологічного синтезу пристрою IOPORT.

Як видно з рис. 2.9, для реалізації двонаправленого порту використовуються всі три секції, що налагоджуються, I, O і T

блока IOB, кожна з яких може бути доповнена елементами пам'яті залежно від функціонального призначення [1].

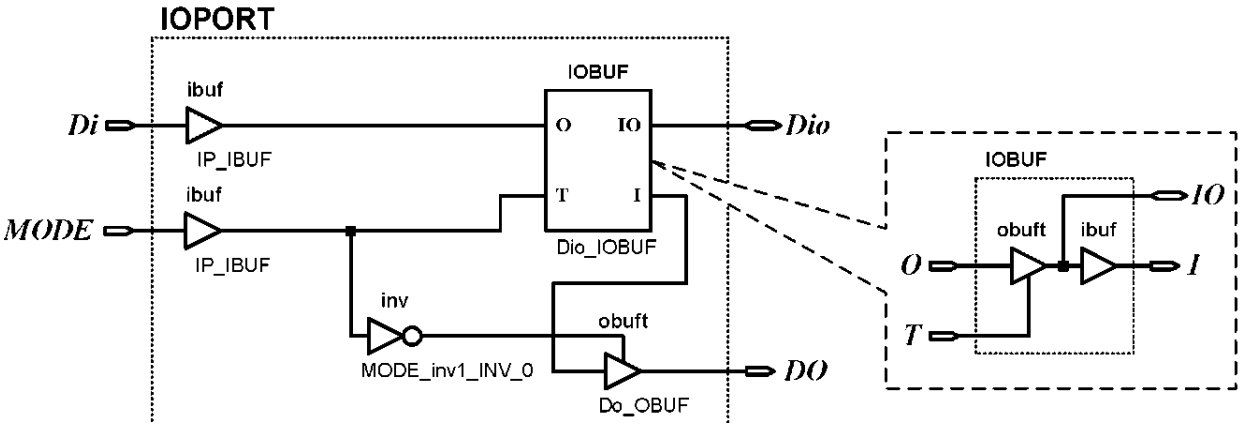



Рис. 2.9. Результат технологічного синтезу двонаправленого порту

На закінчення слід зазначити, що для опису буферних елементів також можна використовувати послідовні оператори мови VHDL і процеси, а для опису вхідних і вихідних багаторозрядних шин – об'єкти типу **std_logic_vector**. При складанні структурних описів складних цифрових пристроїв краще описувати буферні елементи як окремі компоненти або оператори, що підвищує не тільки читабельність вихідного VHDL-коду, але і дозволяє легко вносити зміни до структури і функціонування описуваних портів.

## 2.3. Проектування комбінаційних пристроїв

Комбінаційна логіка, що лежить в основі побудови комбінаційних схем цифрових пристроїв, часто подається як мережева статична модель, елементами якої є базові логічні вентилі [56]. Довільна комбінаційна схема може бути подана як два види абстракції: перемикачів (булевими функціями) і/або мережею логічних вентилів. Обидва види абстракції підтримуються мовою VHDL і можуть бути реалізовані одним з розглянутих раніше стилів опису цифрових пристроїв.

Під *комбінаційною схемою* будемо розуміти цифрову схему, що має n вхідних двійкових портів і m вихідних портів, значення

сигналів на яких знаходяться в прямій і однозначній залежності від значень сигналів на вхідних портах.

Нехай B є множиною, що містить два елементи 0 і 1 (B = {0,1}). Нехай запис виду $B^n$ визначає множину всіх двійкових слів довжиною n, наприклад $B^2$ = {00,01,10,11}. Нехай запис виду $f: X \wedge Y$ визначає функцію f , яка відображує множину X на множину Y. Це означає, що для довільного елемента xGX значення функції $f(x)$ є унікальним елементом з множини $Y$.

Таким чином, поведінку комбінаційної схеми можна подати як функції $f: B^n \wedge B^m$, яка є *перемикальною функцією.*

Для малих значень n і m = 1 перемикальну функцію можна подати як таблицю, яка містить всі можливі $2^n$ значення елементів з множини $B^n$ і значення відповідних елементів з множини {0,1}. Подібні таблиці називаються *таблицями істинності*, при цьому елементи множини $B^n$ є аргументами, а елементи множини {0,1} є значеннями перемикальної функції. На рис. 2.10 подано перемикальні функції базових логічних операцій і відповідні їм базові логічні вентилі [1].

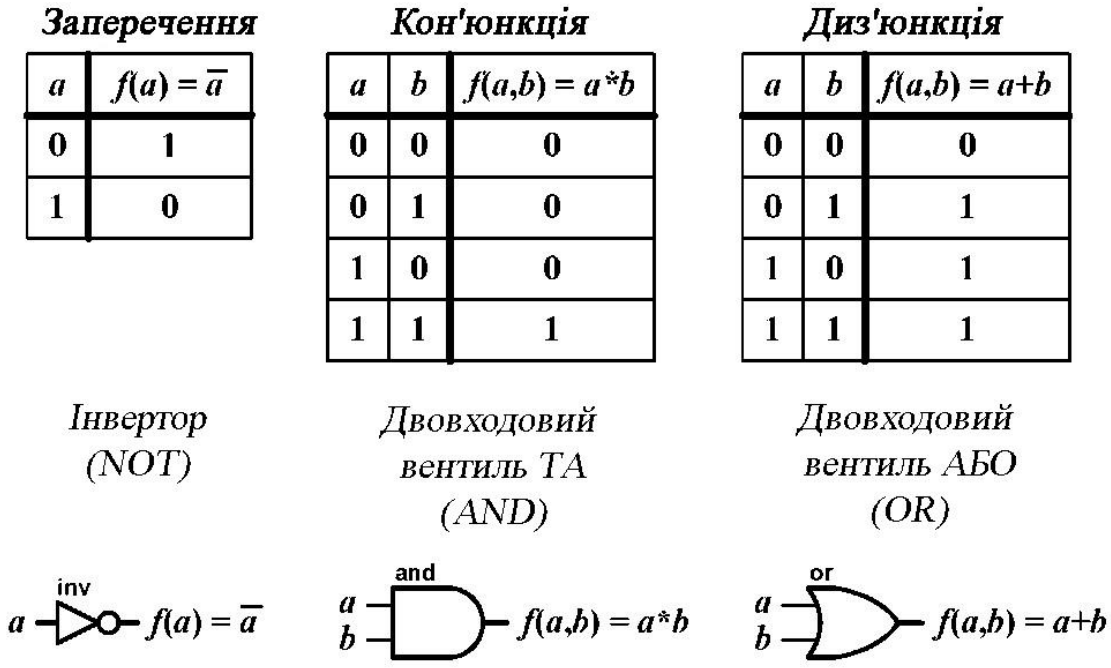



Рис. 2.10. Таблиці істинності базових логічних операцій

У загальному випадку таблиця істинності наочно демонструє, для яких значень аргументів відповідна

перемикальна функція набуває значення 1 (істина) і 0 (похибка). Таблиця істинності є розгорнутим поданням перемикальної функції. Для формування більш компактного подання перемикальної функції можна використовувати скорочену таблицю істинності, яка випливає з повної таблиці шляхом виключення рядків, у яких значення функції дорівнює 0. Однак для великої кількості n і такий спосіб подання перемикальної функції є надмірним і важким для сприйняття.

Альтернативним способом подання перемикальної функції є використання логічного (булевого) виразу, що являє собою математичний запис перемикальної функції за допомогою базових логічних операцій. Нехай дана таблиця істинності перемикальної функції f від n аргументів. При цьому таблиця істинності складається з $2^n$ рядків і двох стовпців. Виберемо один *i*-й рядок з таблиці, у якій функція набуває значення 1 (f = 1). Одиничного значення функція набуває при конкретному значенні відповідного *i*-го набору аргументів $\{a_i^0, a_i^1, \dots, a_i^{n-1}\}$. Подамо *i*-й набір аргументів як такий запис: $m_i = m_i^0 \times m_i^0 \times \dots \times m_i^{n-1}$, де оператор – це логічна функція I (кон'юнкція), а елемент $m_i^j$ подається як $a_i^j$ при $a_i^j = 1$ або як $\overline{a_i^j}$ (заперечення $a_i^j$) при $a_i^j = 0$. При цьому сформувався логічний вираз m, який називається *мінтермом,* що визначає одну з умов, при яких перемикальна функція набуває значення 1. Наприклад, мінтерм $m_2$ перемикальної базової функції АБО (див. рис. 2.10) подається як логічний вираз $m_2 = a \times \bar{b}$, який слід трактувати так: функція $f(a,b) = a + b$ на вхідному наборі *(a, b)* = (1, 0) набуває значення 1. Таким чином, одиничне значення функції можна подати як логічний вираз

$$f = f_0 \times m_0 + f_1 \times m_1 + \dots + f_{n-1} \times m_{n-1},$$

де оператор – це базова логічна операція АБО (диз'юнкція). З урахуванням того, що з наведеного виразу можна виключити доданки, для яких $f = 0$, значення перемикальної функції можна подати як

$$f = \sum_{i-0}^{n-1} f_i \times m_i, \forall f_i = 1. \qquad (2.1)$$

Розглянемо приклад. Нехай дана перемикальна функція трьох аргументів, таблиця істинності якої представлена в табл. 2.1.

Таблиця 2.1

Таблиця істинності і мінтерми функції *f*(a,b,c)

| *a* | *b* | *c* | Мінтерми | *f(a,b,c)* |
|---|---|---|---|---|
| *0* | *0* | *0* | $m_0 = \bar{a} \times \bar{b} \times \bar{c}$ | $f_0 = 0$ |
| *0* | *0* | *1* | $m_1 = \bar{a} \times \bar{b} \times c$ | $f_1 = 1$ |
| *0* | *1* | *0* | $m_2 = \bar{a} \times b \times \bar{c}$ | $f_2 = 1$ |
| *0* | *1* | *1* | $m_3 = \bar{a} \times b \times c$ | $f_3 = 0$ |
| *1* | *0* | *0* | $m_4 = a \times \bar{b} \times \bar{c}$ | $f_4 = 1$ |
| *1* | *0* | *1* | $m_5 = a \times \bar{b} \times c$ | $f_5 = 0$ |
| *1* | *1* | *0* | $m_6 = a \times b \times \bar{c}$ | $f_6 = 0$ |
| *1* | *1* | *1* | $m_7 = a \times b \times c$ | $f_7 = 1$ |

Функція набуває значення 1 на чотирьох наборах (1, 2, 4 і 7), і подається як такий логічний вираз:

$$f(a,b,c) = f_1 \times m_1 + f_2 \times m_2 + f_4 \times m_4 + f_7 \times m_7. \qquad (2.2)$$

З урахуванням, що $f_1$, $f_2$, $f_4$ та $f_7 = 1$ і розкриваючи значення всіх мінтермів, подамо вираз (2.2) як

$$f(a,b,c) = \bar{a} \times \bar{b} \times c + \bar{a} \times b \times \bar{c} + a \times \bar{b} \times \bar{c} + a \times b \times c. \qquad (2.3)$$

Вираз (2.3) являє собою досконалу диз'юнктивну нормальну форму (ДДНФ) функції *f(a,b,c)*. У загальному випадку ДДНФ довільної перемикальної функції можна подати за допомогою трьох базових операцій (І, АБО, НІ), що у свою чергу дозволяє реалізувати функцію у вигляді комбінаційної схеми, елементами якої є базові логічні вентилі (рис. 2.11) [1].

Таким чином, можна стверджувати, що реалізація довільної перемикальної функції являє собою комбінаційну схему, що складається з базових логічних вентилів, з'єднаних між собою без використання зворотних зв'язків. В іншому випадку за наявності зворотних зв'язків можуть виникати ситуації, при яких значення реалізованої комутаційної функції будуть залежати не тільки від

значення вхідних наборів, а й від значень сигналів на внутрішніх полюсах схеми.

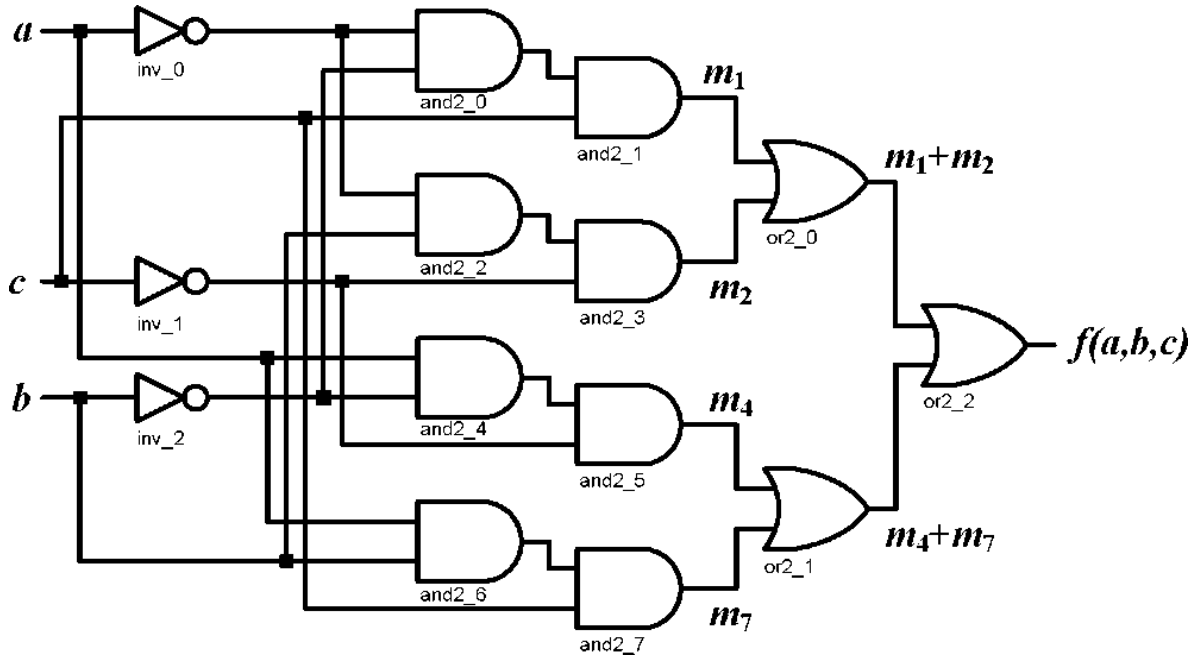



Рис. 2.11. Схемна реалізація перемикальної функції *f(a,b,c)* у базисі логічних елементів

Схемне подання перемикальної функції безпосередньо залежить від базових елементів того рівня абстракції, на якому воно відображується. Так, на рис. 2.11 подано результат реалізації функції на логічному рівні абстракції, а процес переходу від логічного виразу (2.3) до поданої комбінаційної схеми є *логічним синтезом.* [1]

На RTL-рівні подання логічний вираз (2.3) може бути реалізовано в базисі багатовхідних логічних елементів з інверсними входами, що заміщують використання інверторів. Наприклад, у бібліотеці RTL-примітивів фірми Xilinx Inc. є логічні вентилі **and3** (тривходовий елемент І), **and3b2** (тривходовий елемент І з двома інверсними входами), **or4** (чотиривходовий елемент АБО) тощо. У цьому випадку результат RTL-синтезу перемикальної функції f*(a,b,c)* може виглядати так, як це показано на рис. 2.12 [1].

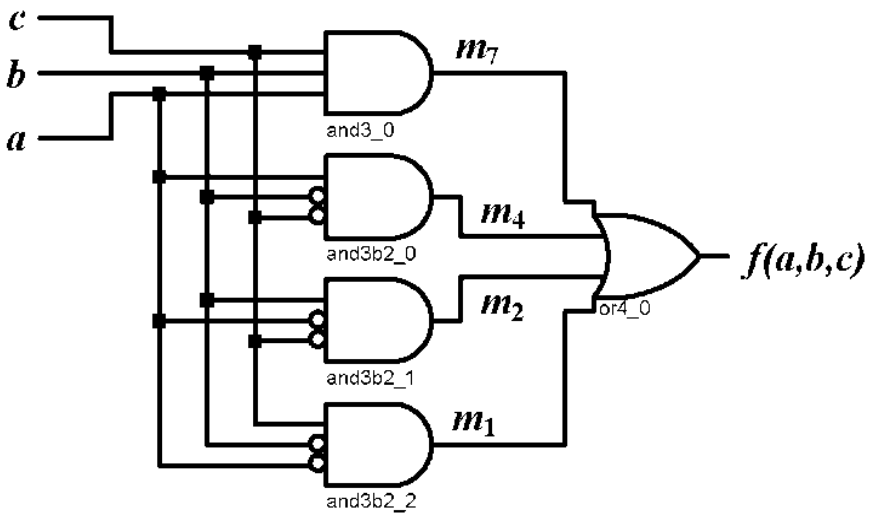



Рис. 2.12. Схемна реалізація функції *f(a,b,c)*
у базисі RTL-примітивів

У разі технологічного синтезу комбінаційної схеми здійснюється трансляція її RTL-подання у схему, яка містить базові технологічні елементи. Так, для FPGA-технології практично всі комбінаційні схеми подаються в базисі LUT-блоків. За своєю суттю LUT-блок являє собою біт-орієнтований елемент пам'яті, який може розглядатися як апаратна реалізація таблиці істинності. Наприклад, в інтегральній схемі Spartan-3E xc3s500e кожен CLB-блок містить вісім 4-входових LUT-блоки, кожен з яких дозволяє реалізовувати одну перемикальну функцію від одного до чотирьох аргументів [55]. На рис. 2.13 подано результат технологічного синтезу функції *f(a,b,c)* для FPGA xc3s500e з урахуванням того, що значення аргументів і безпосередньо значення самої функції подаються рівнями сигналів на трьох вхідних і одному вихідному портах [1].

Для можливості опису комбінаційних схем перемикальних функцій у мові VHDL передбачено безліч можливостей. Зупинимося на основних з них. Як було показано вище, комбінаційна логіка може бути подана у вигляді логічних виразів і таблиць істинності. Мова VHDL має набір стандартних логічних операторів, за допомогою яких можна описати довільну перемикальну функцію (not, and, or, xor, nand, nor, nxor). Перераховані оператори можуть бути застосовані до об'єктів типу std_logic, std_logic_vector, bit та bit_vector, при цьому об'єктами є

порти, сигнали і змінні. Наприклад, вираз (2.2) мовою VHDL можна записати як

```
– – . . .
signal m1, m2, m4, m7 : std_logic;
– – . . .
m1 <= (not a) and (not b) and c;
m2 <= (not a) and b and (not c);
m4 <= a and (not b) and (not c);
m7 <= a and b and c ;
f <= m1 or m2 or m4 or m7;
– – . . .
```

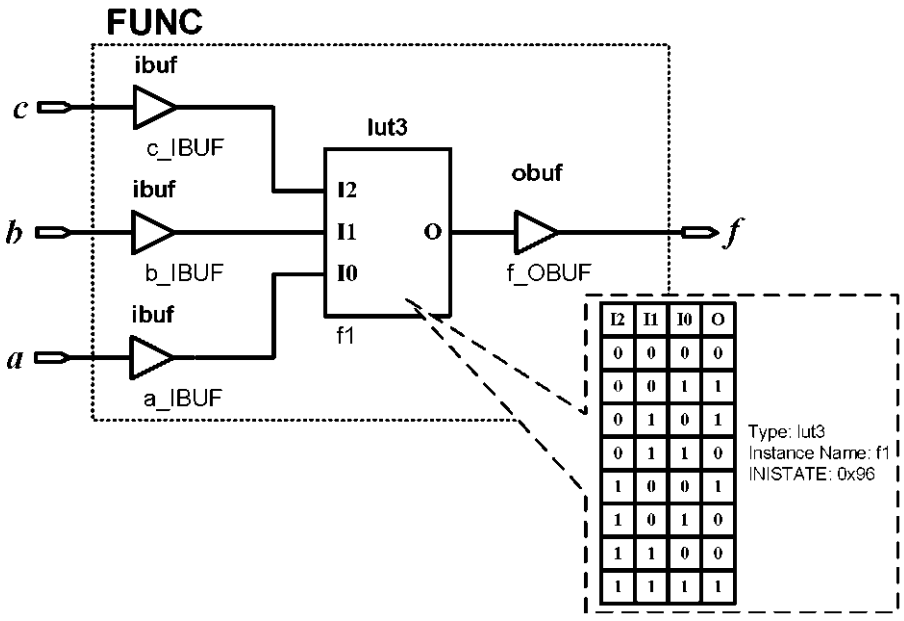



Рис. 2.13. Результат реалізації функції *f(a,b,c)* у базисі технологічних примітивів FPGA

Аналогічно можна описати функцію за допомогою змінних з урахуванням того, що результуюче значення буде присвоєно сигналу:

```
– – . . .
P0: process( a, b, c )
variable m1, m2, m4, m7 : std_logic;
– – . . .
begin
m1 := (not a) and (not b) and c;
```

```
m2 := (not a) and b and (not c);
m4 := a and (not b) and (not c );
m7 := a and b and c ;
f <= m1 or m2 or m4 or m7 ;
end process;
-- . . .
```

З урахуванням, що в мові VHDL є синтезований логічний оператор **xor,** функцію f можна описати як

```
-- . . .
f <= a xor b xor c;
-- . . .
```

Мова VHDL також дозволяє складати синтезовані описи перемикальних функцій на основі їхніх таблиць істинності. Наприклад, шляхом використання оператора конкатенації **&** можна сформувати двійковий вектор аргументів функції, а потім за допомогою паралельного оператора **with-select** перерахувати мінтерми, при яких функція набуває одиничного значення.

```
- - . . .
signal abc: std_logic_vector (2 downto 0);
- - . . .
abc <= a & b & c ;
with abc select
f <= '1' when "001" | "010" | "100" | "111",
'0 ' when others ;
- - . . .
```

Ключове слово **others,** використане разом з іншим оператором **when,** визначає всі інші умови, які не були перераховані в першому операторі **when** і при яких функція набуває нульових значень. Аналогом оператора **with-select** можна вважати послідовний оператор **case,** що застосовується в процессах.

```
- - . . .
P0: process ( a, b, c )
variable abc: std_logic_vector (2 downto 0);
- - . . .
begin
```

```
abc := a & b & c;
case abc is
when "001" | "010" | "100" | "111" =>
f <= '1 ';
when others => f <= '0 ';
end case;
end process;
- - . . .
```

Відмінною особливістю опису перемикальних функцій і синтезу комбінаційних схем є визначення за допомогою мови VHDL всіх умов, при яких функція набуває своїх значень. Розглянемо приклад опису комбінаційної схеми двовходового мультиплексора, перемикальна функція якого описується таким логічним виразом:

$$f(a,b,s) = a \times \bar{s} + b \times s, \qquad (2.4)$$

де аргументи *a* і *b* є значенням обраних сигналів, а *s* – значенням селективного сигналу. Якщо s = 0, то значення функції *f* дорівнює значенню аргументу *a*, в іншому випадку – значенню аргументу *b*.

Перемикальну функцію (2.4) можна описати мовою VHDL так, як це було показано в прикладах вище. З іншого боку, комбінаційні схеми вибірки сигналів краще описувати за допомогою послідовного оператора **if**.

```
- - . . .
P0: process ( a, b, s )
begin
if s = '0' then
f <= a ;
else
f <= b ;
end if ;
end process;
- - . . .
```

Оператор **else** в цьому випадку відіграє таку саму роль, що і ключове слово **others** в попередніх прикладах, а саме визначає значення функції *f* за умов, відмінних від s = 0. Оператор **if** також придатний для опису буферних тристабільних елементів:

```
– – . . .
P0: process ( a, b, s )
begin
if s = '0' then
f <= a ;
else
f <= 'Z';
end if ;
end process;
– – . . .
```

Однак якщо в операторі **if** не вказати значення, що привласнюються сигналу за інших умов, то результатом синтезу такого оператора буде не комбінаційна схема, а схема з пам'яттю. Наприклад, приберемо з двох попередніх описів гілку альтернативних умов з оператора **if**.

```
– – . . .
P0: process ( a, b, s )
begin
if s = '0' then
f <= a ;
end if ;
end process;
– – . . .
```

За умови s = 0 значення сигналу *f* буде дорівнювати значенню сигналу *a*. Однак якщо умова зміниться на протилежне (s = 1), то сигнал *f* не змінить свого значення, а воно залишиться рівним тому значенню *a*, яке було визначено за умови s = 0. Результатом синтезу подібного оператора **if** буде асинхронний елемент пам'яті ld_1, приклад якого наведено на рис. 2.8.

На основі розглянутих прикладів можна виробити правило опису перемикальних функцій мовою VHDL, який має на меті визначення значень функцій, що формуються за допомогою високорівневих операторів, у яких визначено всі умови, за яких функції набувають одиничних і нульових значень.

Наведене правило для оператора **if** означає обов'язкову наявність гілки альтернативних умов **else.** Розглянемо ще один приклад використання оператора **if** для опису тривходового

мультиплексора, який має три вхідних порти даних *a*, *b* і *c*, дворозрядну шину селективного сигналу *s* і один вихідний порт *f*. Нехай перемикальна функція вихідного сигналу порту *f* описується таким логічним виразом:

$$F(a,b,c,s(1),s(0)) = a \times s(1) \times s(0) + b \times s(1) \times s(0) + c \times s(1) \times s(0). \quad (2.5)$$

Складемо VHDL-опис виразу (2.5) за допомогою оператора **if**.

```
– – . . .
P0: process ( a, b, c, s )
 begin
if s="00" then
f <= a;
elsif s="01" then
f <= b ;
 elsif s="10" then
f <= c ;
end if ;
end process;
– – . . .
```

Результатом RTL-синтезу такого опису буде схема, подана на рис. 2.14, яка складається з тривходового мультиплексора mux_0, двовходового вентиля I and2_0 і D-засувки latch_0 [1].

Використаний оператор **if** дійсно описує поведінку схеми вибору значення сигналу *f* залежно від значень сигналів на селективній шині s, що призводить до синтезу RTL-компоненти mux_0. Однак у поданому VHDL-описі відсутня інформація про формування значення *f* за умови s = «11», що призводить до синтезу елемента пам'яті latch_0 і схемою його управління у вигляді логічного елемента and2_0. При значеннях s, відмінних від «11», на виході елемента and2_0 формується сигнал рівня логічного нуля, що подається на вхід G, що забезпечує трансляцію сигналу зі входу D на вихідний порт Q. При s = «11» засувка latch_0 переходить у режим зберігання, при якому неможлива зміна сигналу на її виході.

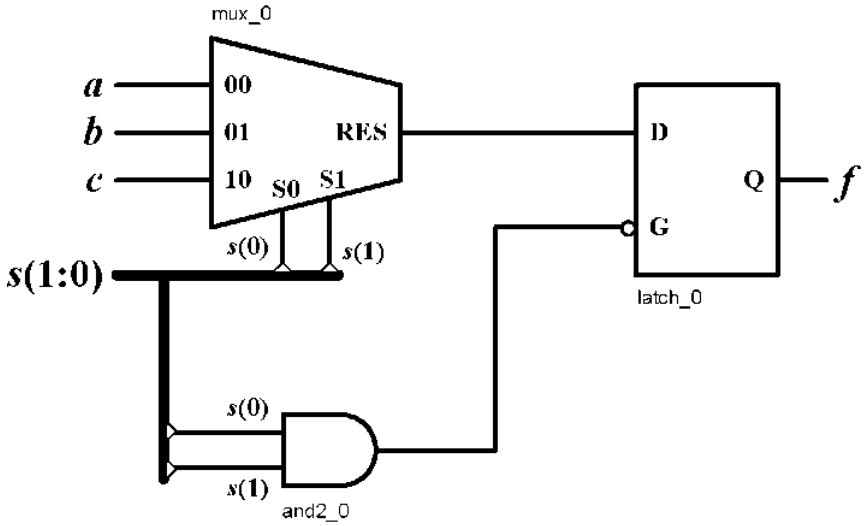



Рис. 2.14. Результат RTL–синтезу оператора **if**

Проблема синтезу «зайвих» компонент обумовлена частковим визначенням значень перемикальної функції *f* за допомогою оператора **if.** Виконаємо довизначення значень *f*.

```
- - . . .
P0: process ( a, b, c, s )
begin
if s="00" then
f <= a ;
elsif s="01" then
f <= b ;
elsif s="10" then
f <= c ;
else — довизначення функції f
f <= '0'; — для випадку, коли s = "11"
end if ;
end process;
- - . . .
```

У цьому випадку перемикальна функція повністю визначена, а результатом технологічного синтезу буде комбінаційний пристрій (рис. 2.15) [1].

Автоматизований засіб технологічного синтезу реалізував комбінаційну схему перемикальної функції *f* засобами одного slice-блока, використавши при цьому два генератори функцій lut3 і lut2 і один мультиплексор muxf5.

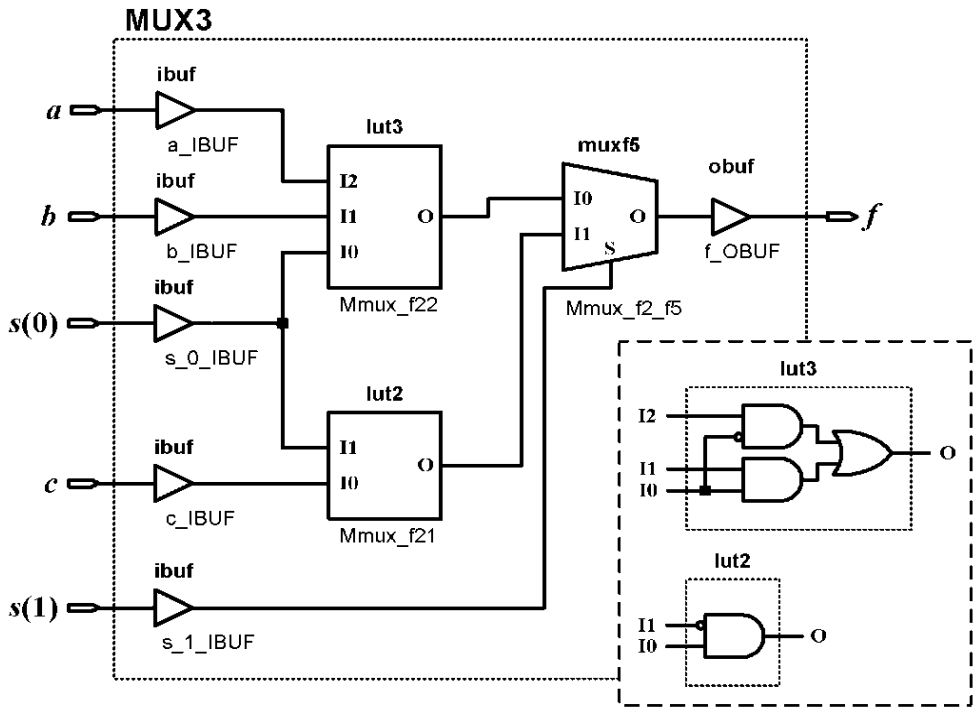



Рис. 2.15. Результат технологічного синтезу тривходового мультиплексора

Розглянемо ще один приклад VHDL-опису комбінаційних схем за допомогою таблиць істинності. Спроектуємо компоненту, яка буде виконувати роль таблиці істинності для реалізації довільної перемикальної функції від чотирьох аргументів. Наведемо таку компоненту за допомогою generic-параметрів, які дозволять визначати кількість аргументів (вхідних портів) і безпосередньо значення функції згідно з її таблицею істинності (дод. 1, лістинг 2.3 [1]).

Для поданих у лістингу початкових значень generic-параметрів (Ni: = 1, INIT: = x"0001") компонента описує перемикальну функцію заперечення значення одного аргументу, результатом синтезу якої є інвертор.

Приклади використання компоненти CLOGIC_4x1 і їхні еквівалентні логічні вирази подано нижче.

```
_ _ . . .
_ F <= not A;
U0: CLOGIC_4 generic map ( 1, x"0001 " )
             port map (A, F );
_ F <= A and B;
U1: CLOGIC_4 generic map ( 2, x"0004" )
```

```
                p ort map ( A & B , F ) ;
    — F <= A xor B;
    U1: CLOGIC_4 generic map ( 2, x"0006" )
                p ort map ( A & B , F ) ;
    — F <= A xor B xor C;
    U1: CLOGIC_4 generic map ( 3, x"0096" ) —
 см. табл. 2.1
                port map ( A & B & C, F );
- - . . .
```

Результатом технологічного синтезу представлених компонент для FPGA будуть чотиривходові LUT-блоки, які використовуються в якості генераторів перемикальних функцій. Для САПР Xilinx ISE існує можливість безпосереднього використання технологічних примітивів, представлених синтезуємими VHDL-описами. Наприклад, в бібліотеці UNISIM є пакет vcomponents, що містить опис технологічної компоненти lut4, який можна використовувати безпосередньо в призначеному для користувача проекті:

```
    Library UNISIM;
    use UNISIM . vcomponents . all ;
    - - . . .
    _ _ LUT4: 4— input Look—Up Table with general
output
    _ _          Spartan —3E
    _ _ Xilinx HDL Language Template, version 12.4

    LUT4_inst : LUT4
    generic map (
    INIT => X"0000")
    port map (
    O => O, — LUT general output
    I0 => I0 , -- LUT input
    I1 => I1 , — LUT input
    I2 => I2 , — LUT input
    I3 => I3 -- LUT input
    );
    — End of LUT4 inst instantiation
```

На закінчення слід зазначити, що реалізація перемикальних функцій і комбінаційних схем може бути здійснена за допомогою структурних і змішаних VHDL-описів, елементами яких можуть бути як стандартні бібліотечні компоненти і технологічні примітиви, так і призначені для користувача компоненти, спроектовані з урахуванням вищевикладених рекомендацій.

## 2.4. Проектування послідовних пристроїв

Під *послідовними пристроями* розуміють цифрові пристрої, які формують значення вихідних сигналів не тільки від поточних значень вхідних сигналів, але і від значень вхідних сигналів, що надійшли в попередні моменти часу [57, 58]. Таким чином, послідовні пристрої аналізують послідовність надходження сигналів на своїх входах для формування значень сигналів на своїх виходах. Для можливості аналізу послідовності вхідних сигналів необхідна наявність елементів пам'яті, що зберігають попередні значення вхідних сигналів. Таким чином, послідовні пристрої є цифровими пристроями з пам'яттю, а загальну схему таких пристроїв можна подати як структурну схему на рис. 2.16 [1].

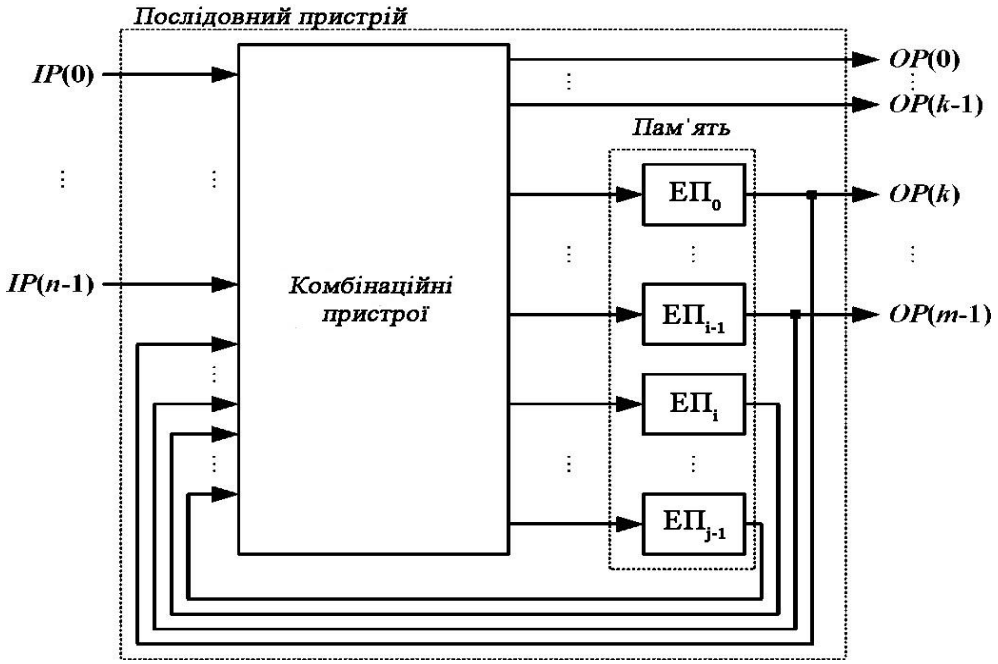



Рис. 2.16. Узагальнена структура послідовних пристроїв

У загальному випадку послідовний пристрій містить n входів, j елементів пам'яті ЕП і m виходів. Елементи пам'яті служать для зберігання значень вхідних і внутрішніх сигналів. Значення, які зберігаються на елементах пам'яті, визначають поточний стан послідовного пристрою. Поточний стан може безпосередньо використовуватися для формування значень вихідних сигналів *(OP(k),...,OP(m*—1)) або брати участь у формуванні значень перемикальних функцій *(OP*(0),...,OP(k—1)) чи нового стану пристрою.

Цифрові послідовні пристрої класифікуються так:

– найпростіші елементи пам'яті, які у свою чергу можуть бути засувками (асинхронними елементами) і тригерами (синхронними елементами);

– регістрові пристрої, які виконують не тільки функції зберігання даних, але і лежать в основі побудови лічильних схем, подільників частоти, генераторів числових послідовностей тощо;

– синхронні кінцеві автомати, що являють собою, як правило, цифрові пристрої управління;

– запам'ятовуючі пристрої великої ємності.

Послідовно розглянемо перераховані види цифрових пристроїв з пам'яттю і особливості їх проектування за допомогою мови VHDL.

Найпростішим цифровим запам'ятовуючим елементом можна вважати бістабільну схему, яка складається з двох послідовно з'єднаних інверторів з позитивним зворотним зв'язком, який з'єднує вихід останнього інвертора зі входом першого. Визначимо, що вихід першого інвертора підключений до виходу *Q* бістабільної схеми. Зрозуміло, що на виході другого інвертора *nQ* буде формуватися інверсне значення сигналу *Q*. Значення на виході визначаються в момент вмикання бістабільної схеми і залежать від багатьох чинників, у тому числі і фізичної особливості структурних елементів даної схеми.

Для можливості задавання певного значення на вихідних портах бістабільна схема може бути модифікована таким чином. Замінимо інвертори двовходовими логічними вентилями АБО-НІ, вихідні порти яких разомо з першими вхідними портами утворюють ланцюг зворотного зв'язку. Другі вхідні порти обох

вентилів підключимо до вхідних портів керуючих сигналів *S* і *R* (рис. 2.17) [1].

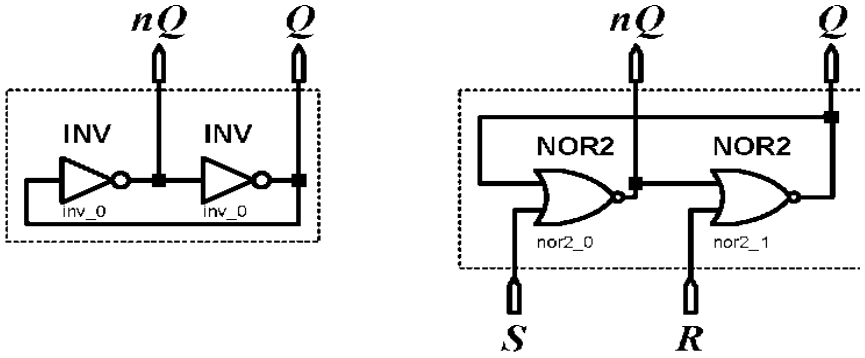



Рис. 2.17. Схеми бістабільних елементів

Зрозуміло, що при R = 0 і S = 0 поведінка схеми еквівалентна найпростішій бістабільній схемі з інверторами. Припустимо, що за цієї умови сигнал на виході Q набуває значення qG {0,1}. Тоді значення сигналу на виході nQ дорівнюватиме qG {0,1}. Потім встановимо на керуючому вході S сигнал з рівнем "1" (S = 1), при цьому продовжуючи утримувати сигнал на другому вході R = 0. У цьому випадку значення сигналу на виході nQ дорівнюватиме 0 (1+q=0), а на виході Q — 1 (0 + 0 = 1).

При поданні на керуючі входи схеми протилежних значень (R=1, S=0) на виходах також встановлюються протилежні значення сигналів (Q=0, nQ=1).

Таким чином, можна констатувати, що вхідний вектор керуючих сигналів RS = «01» встановлює схему в стійкий стан 1 (Q = 1), а вектор RS = «10» — у стійкий стан 0 (Q = 0). При подачі вхідного вектора RS = «00» стійкий стан схеми не зміниться, що можна інтерпретувати як зберігання попереднього стійкого стану.

Якщо при зберіганні стійкого стану q на керуючі входи подати значення RS = «11», то на вихідних портах сформуються сигнали Q = q + 1 = 0 та nQ = q + 1 = 0, що можна розглядати як «неприпустимий стан» схеми через вироблення сигналів з протилежними значеннями. Крім цього, при послідовній подачі керуючих сигналів RS = «11» і RS = «00» схема перемикається в

нестабільний стан, при якому значення на її виходах буду непередбачуваними.

За умови, що комбінація сигналів RS = «11» ніколи не буде подана на відповідні керуючі входи, розглянута схема є найпростішою схемою з пам'яттю, що називається *асинхронним тригером*, або *засувкою*.

Розглянутий найпростіший елемент пам'яті є асинхронним через управління рівнями вхідних сигналів R і S. Активний рівень сигналу R = 1 встановлює елемент пам'яті в нульовий стан Q = 0 і називається сигналом скидання **(R**eset), а активний рівень сигналу S = 1 встановлює елемент пам'яті в одиничний стан Q = 1 і називається сигналом установки **(S**et). Зважаючи на це повна назва подібної схеми елемента пам'яті звучить як асинхронний RS-тригер, або асинхронна RS-засувка (RS–latch).

Розглянуту схему можна описати на структурній VHDL, проте результатом RTL- і технологічного синтезу для FPGA буде більш складна схема через відсутність відповідної компоненти RS-засувки як серед RTL-примітивів, так і серед технологічних примітивів.

Через наявність «забороненої» комбінації керуючих сигналів ця схема елемента пам'яті не використовується на практиці. Замість цього поширення набули схеми асинхронних елементів пам'яті у вигляді D-засувок (D-latch або gated D-latch).

Логічна схема D-засувки може бути отримана зі схеми RS-засувки шляхом парафазного об'єднання вхідних портів R і S в єдиний вхідний порт D, тим самим перешкоджаючи появі «забороненої» комбінації керуючих сигналів RS = «11». Для можливості подачі комбінації керуючих сигналів для перемикання схеми в режим зберігання необхідно ще ввести додатковий керуючий вхід дозволяючого сигналу E (рис. 2.18) [1].

При одиничному значенні рівня сигналу на дозволяючому вході E схема функціонує як RS-засувка в режимах встановлення/скидання. При E = 0 RS-засувка перемикається в режим зберігання. Функціонування логічних елементів з пам'яттю прийнято формально описувати за допомогою таблиць переходів або граф-схем станів (рис. 2.19) [1].

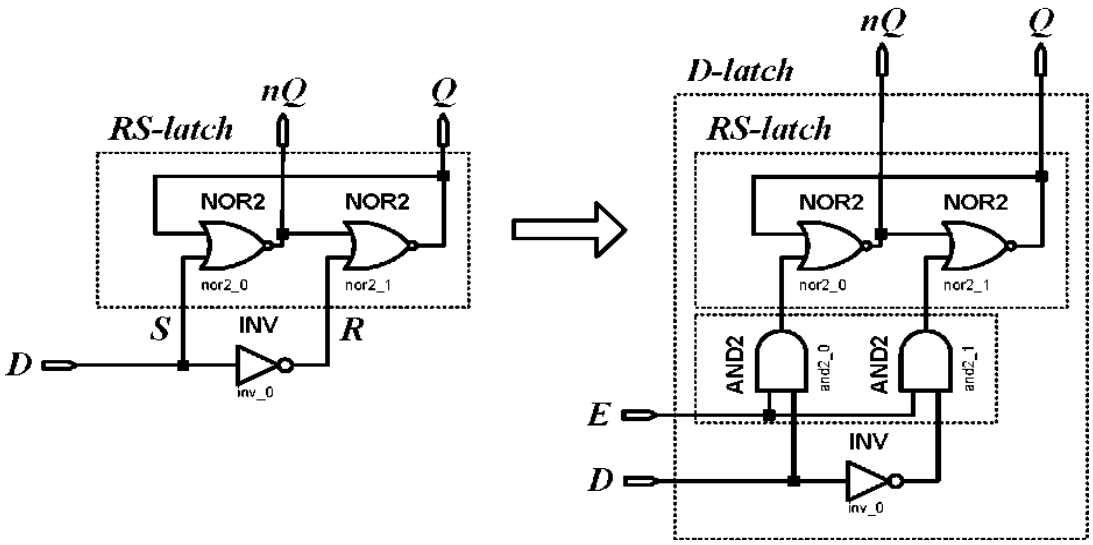
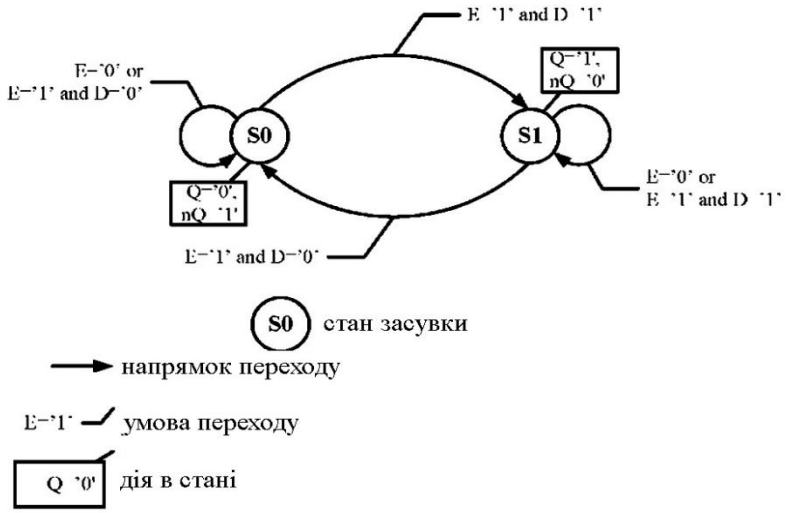



Рис. 2.18. Процес перетворення схеми RS-засувки в D-засувку

| E | D | Q(t) | Q(t+1) | nQ(t+1) |
|---|---|---|---|---|
| 0 | X | q | q | $\bar{q}$ |
| 1 | 0 | q | 0 | 1 |
| 1 | 1 | q | 1 | 0 |




**X** – байдуже значення
**q**={0,1} – значення стану
***Q(t)*** – поточне значення на виході
***Q(t+1)*** – наступне значення на вихід

Рис. 2.19. Таблиця переходів і граф-схема станів D-засувки

Опишемо функціонування D-засувки поведінковим стилем за допомогою оператора **if** мови VHDL (дод. 1, лістинг 2.4). Як видно з прикладу, при значенні сигналу дозволу E=1 вихідний порт Q набуває значення сигналу з входу D, а інверсне значення цього сигналу надходить на вихід nQ. При E=0 оператор **if** не виконується, а відсутність у процесі Main будь-яких інших операторів дозволяє зберегти попередні значення сигналів, що транслюються на вихідні порти. На рис. 2.20 показано результат синтезу наведеного VHDL-опису [1].

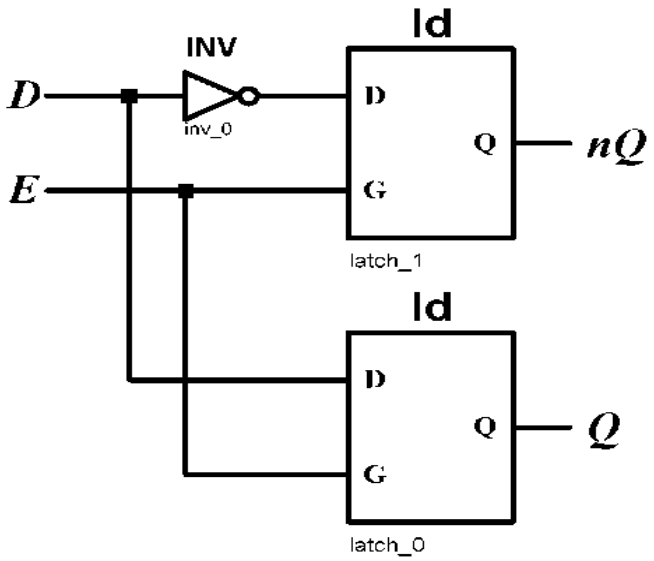



Рис. 2.20. Результат RTL-синтезу D-засувки

З огляду на те, що в бібліотеці RTL-примітивів САПР Xilinx Inc. не існує простих елементів пам'яті з парафазними виходами, результатом синтезу є два елементи пам'яті. Придатним за функціональними ознаками описуваної RS-засувки є RTL-примітив ld (latch d), що має вхідний порт сигналу даних D (Data), вхідний порт сигналу дозволу установлення стану G (Gate) і єдиний вихідний порт сигналу стану *Q*.

У наведеному VHDL-прикладі є три істотні помилки, які призвели до вищеописаного результату синтезу. По-перше, незважаючи на присутність у послідовному операторі if двох паралельних операторів присвоювання, синтезуються вони також у паралельні об'єкти, якими і є елементи пам'яті. По-друге, у VHDL-описі відсутній у явному вигляді об'єкт синтезу (D-засувка), адже об'єкти *Q* і *nQ* оголошені як порти, результатом синтезу яких будуть відповідні буферні елементи. По-третє, при формуванні парафазного значення сигналу стану на виході *nQ* за основу було взято значення вхідного сигналу на порту D, а не значення на виході елемента пам'яті, тобто значення сигналу *Q*. З огляду на розглянуті помилки, внесемо зміни в VHDL-опис D-засувки.

```
- -  …….
Architecture Beh of DLATCH is
- - Об'єкт синтезу елемента пам'яті
```

```
    signal q_t : std_logic;
    Begin
    - - Поведінковий опис елемента пам'яті
    Main: process ( D, E )
    begin
    if E = ' 1 ' then
    q_t <= D;
    end if;
    end process;
    - - Передача значення, що зберігається, на
вихідний порт
    Q <= q_t;
    - - Формування парафазного значення
    nQ <= not q_t;

    End Beh ;
    - -  …….
```

Результатом синтезу модифікованого опису буде схема, що складається з двох RTL-примітивів: D-засувка та інвертор. Розглянемо проектування базових елементів пам'яті з більш складним управлінням. Так, існують RTL- і технологічні примітиви D-засувок з можливістю асинхронного скидання їх стану в нуль (Clear) і асинхронним попереднім встановленням стану в значення одиниці (Preset). Прикладом подібних елементів пам'яті може служити RTL-примітив ldcp, який є D-засувкою з додатковим портом скидання CLR і встановлення PRE. Опис функціонування цього примітиву наведено в роботі [59], а таблицю переходів подано в табл. 2.2. Як видно з табл. 2.2, сигнал на керуючому вході CLR є домінуючим по відношенню до всіх інших вхідних сигналів.

Таблиця 2.2

Таблиця переходів RTL-примітиву ldcp

| *CLR* | *PRE* | G | D | Q(t) | *Q*(*t* + 1) |
|---|---|---|---|---|---|
| 1 | X | X | X | X | 0 |
| 0 | 1 | X | X | X | 1 |
| 0 | 0 | 1 | d | X | d |
| 0 | 0 | 0 | X | q | q |

Спробуємо цю особливість функціонування відобразити в поведінковій VHDL-моделі засувки ldcp.

```
- - . .
Architecture Beh of LDCP is
- - Об'єкт синтезу елемента пам'яті
signal q_t : std_logic;
Begin
 -  - Поведінковий опис елемента пам'яті
 Main: process ( CLR, PRE, D, G )
begin
if CLR = '1' then — сигнал з найвищим пріоритетом
q_t <= '0'; elsif PRE = '1' then — если CLR='0' q_t <= '1 ';
elsif G = '1' then — если CLR='0' и PRE='0' q_t <= D;
end if ;
end process;
-  - Передача значення, що зберігається, на вихідний порт
Q <= q_t;
End Beh ;
- - . . .
```

При зміні пріоритетності керуючих сигналів

```
- - . .
if PRE = '1' then - - сигнал з найвищим пріоритетом
q_t <= '1 ';
elsif CLR = '1' then - - якщо PRE='0'
q_t <= '0 ';
elsif G = '1' then - - якщо PRE='0' и CLR='0'
q_t <= D;
end if ;
- - . .
```

засіб синтезу модифікує схему управління наявним примітивом (рис. 2.21), що може негативно позначитися на схемній складності розроблювального пристрою [1].

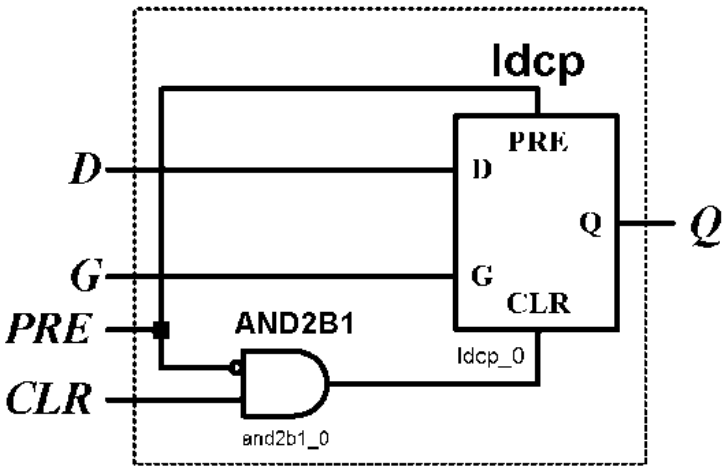



Рис. 2.21. Результат RTL-синтезу засувки ldcp

Іншим типом найпростіших запам'ятовуючих елементів є *тригери*, управління якими здійснюється не рівнем сигналу, що управляє, а перехідним процесом, обумовленим моментом зміни значення керуючого сигналу. Якщо значення логічного рівня сигналу змінюється від значення нуля до одиничного значення ($0 \to 1$), то таку подію прийнято називати *фронтом*, або *переднім фронтом* сигналу. Подію в момент зворотного переходу логічного рівня сигналу ($1 \to 0$) прийнято називати *спадом*, або *заднім фронтом* сигналу. Поведінку тригера можна розглянути на його моделі, що будується на основі двох D-засувок, одна з яких є ведучою (master D-latch), а друга – веденою (slave D-latch) [56]. На рис. 2.22 подана функціональна master-slave модель D-тригера, керованого переднім фронтом синхросигналу [1].

Управління ведучою і веденою засувками здійснюється парафазними значеннями керуючого сигналу CLK. Вихідний порт ведучої засувки з'єднаний з вхідним портом D веденої засувки в точці S. При нульовому значенні сигналу *CLK* = 0 ведуча засувка відкрита і значення сигналу в точці S дорівнює значенню сигналу на вхідному порті D. При настанні переднього фронту сигналу *CLK* ведуча засувка перемикається в режим зберігання, а зафіксоване значення сигналу в точці S транслюється на вихідний порт Q веденої засувки. Таким чином, тригер фіксує і запам'ятовує «миттєве» значення сигналу на вхідному порті D при настанні переднього фронту сигналу *CLK*, який

називається синхронізуючим сигналом, або *синхросигналом*. При цьому тригери є найпростішими синхронними елементами пам'яті, спрацьовування яких відбувається за синхронним фронтом або спадом сигналу. Тригер, керований спадом сигналу синхронізації, можна отримати шляхом інверсії сигналів на дозволяючих входах G засувок latch_0 і latch_1, поданих на рис. 2.22.

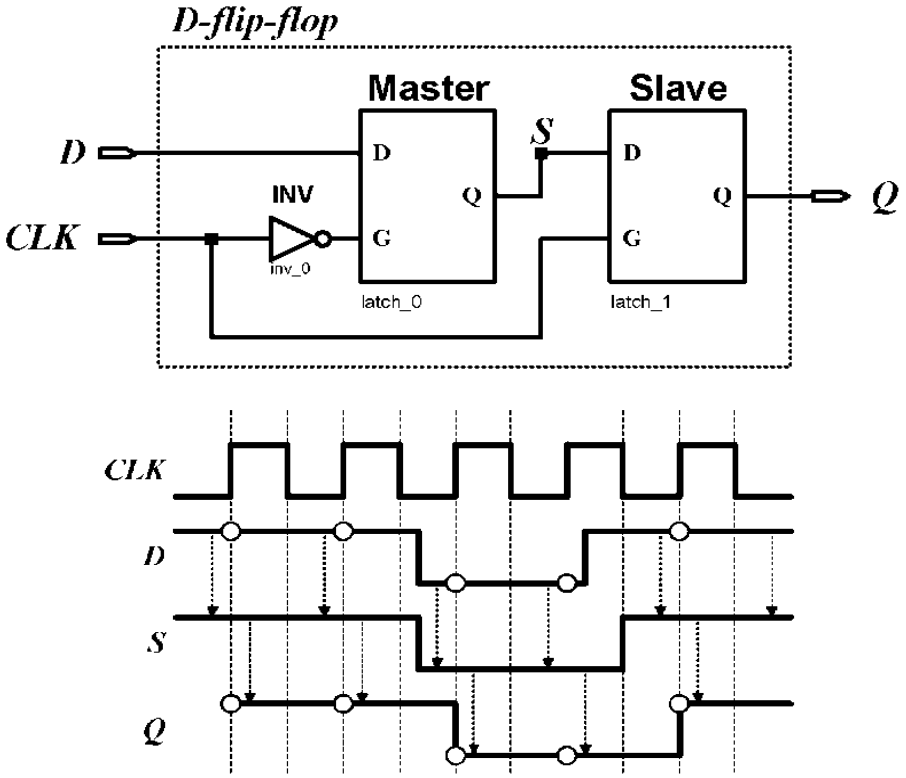



Рис. 2.22. Функціональна модель D-тригера та часова діаграма його сигналів

Для можливості опису поведінкових VHDL-моделей синхронних тригерних схем можна використовувати оператор **event** або функції **rising_edge()** та **falling_edge()** (дод. 1, лістинг 2.5). Почнемо з першого оператора і будемо намагатися використовувати його для створення VHDL-моделі D-тригера, поданого раніше.

Поданий опис схожий з описом D-засувки за винятком використаного оператора event, який бере участь у формуванні комплексної умови настання переднього фронту сигналу синхронізації. Дійсно, VHDL-оператор event повертає значення «істина» у випадках, коли значення відповідного об'єкта змінилося. У наведеному прикладі таким об'єктом є порт CLK, а

при передньому фронті синхросигналу значення порту змінюється з "0" на "1". Таким чином, комплексну умову настання переднього фронту слід інтерпретувати так: значення сигналу змінилося і стало рівним 1. Міркуючи аналогічним чином, можна записати умову настання заднього фронту сигналу синхронізації.

```
- - . . .
- - умова настання заднього фронту сигналу CLK
if ( CLK'event and CLK = '0' ) then
- -  захоплення значення сигналу порту D
s <= D; - -  при всіх інших умовах s здійснює
зберігання
- -  захопленого значення сигналу порту D
end if ;
- - . . .
```

Наведені приклади з використанням оператора event є синтезованими, проте є одна особливість, яка може ввести в оману розробника при функціональному моделюванні подібних описів. Через те, що описуваний порт синхросигналу CLK є об'єктом типу std_logic, він може набувати довільних значень з дев'яти можливих [60]. Припустимо, що при функціональному моделюванні значення сигналу на вхідному порту CLK набуває значення 'Z' з подальшою зміною на значення '1'. У цьому випадку виконається умова настання переднього фронту, що є функціональною помилкою. Для її усунення можна скористатися оператором **last_value,** ускладнивши умову настання фронту сигналу синхронізації.

```
- - …
- - умова настання переднього фронту сигналу
CLK
 if ( CLK'event and CLK = '1' and
CLK'last_value = '0' ) then
- - …
```

Наведена умова настання переднього фронту повністю подана в стандартній функції **rising_edge(),** аргументом якої є сигнал (порт) типу std_logic або std_ulogic.

```
    – – …
    – – умова настання переднього фронту сигналу
CLK
     if rising_edge( CLK ) then
    – – …
```

Результатом синтезу опису, наведеного в лістингу 2.5 для FPGA SPARTAN-3E, буде технологічний примітив **fd**, що являє собою D-тригер, керований переднім фронтом сигналу синхронізації. Як і в разі засувок, існує велика різноманітність найпростіших тригерних елементів, які можна реалізовувати на FGPA. Розглянемо приклад опису та синтезу D-тригера, керованого переднім фронтом синхросигналу, з можливістю асинхронного скидання в нульовий стан, асинхронного установлення в одиничний стан і з керуючим сигналом дозволу синхронізації. Наведеному опису відповідає примітив **fdcpe** (D Flip–Flop with Clock Enable and Asynchronous Preset and Clear) FGPA SPARTAN–3E [59]. Функціонування такого тригера відображено в таблиці переходів (табл. 2.3).

Таблиця 2.3

Таблиця переходів тригера fdcpe

| CLR | *PRE* | CE | D | C | Q(t) | *Q(t* + 1) |
|---|---|---|---|---|---|---|
| 1 | X | X | X | X | X | 0 |
| 0 | 1 | X | X | X | X | 1 |
| 0 | 0 | 0 | X | X | q | q |
| 0 | 0 | 1 | d | t | X | d |

Як і в розглянутому випадку засувки ldcp, керуючі сигнали D-тригера мають явні пріоритети. Так, керуючий сигнал *CLR* асинхронного скидання в нульовий стан має найвищий пріоритет, а сигнал синхронізації C – найменший пріоритет. Тригер захопить миттєве значення сигналу на вході D тільки в разі, якщо сигнали CLR і PRE неактивні і дорівнюють нулю, а сигнал дозволу синхронізації CE активний і дорівнює одиниці. З урахуванням наведеної таблиці спроектуємо поведінковий опис тригера fdcpe (дод. 1, лістинг 2.6).

Результатом синтезу наведеного опису буде технологічний примітив fdcpe (рис.) 2.23 [1].

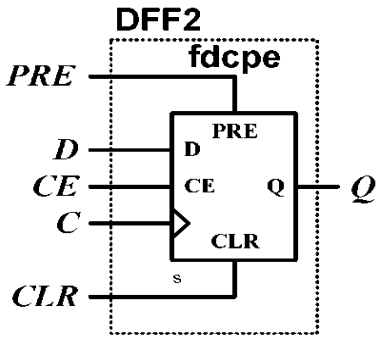



Рис. 2.23. Результат RTL-синтезу компоненти DFF2 з лістингу 2.6

При спробі змінити пріоритет керуючих сигналів або спроби логічного об'єднання умов функціонування результат синтезу буде відмінний від очікуваного. Наприклад, результатом синтезу модифікованого процесу, що описує поведінку тригера fdcpe, буде схема, що має у своєму складі не тільки синхронний елемент пам'яті, але і додаткову комбінаційну логіку, відповідну наведеним умовам функціонування (рис. 2.24).

```
- - …
Main: process ( CLR, PRE, CE, C, D ) begin
if ( CLR =' 0' and PRE = '0' and CE = '1' )
then
if rising_edge( C ) then
s <= D;
end if ;
elsif CLR = '1' then
s <= ' 0 ' ;
elsif PRE = '1' then
s <= ' 1 ' ;
end if ;
end process ;
Q <= s ;
- - …
```

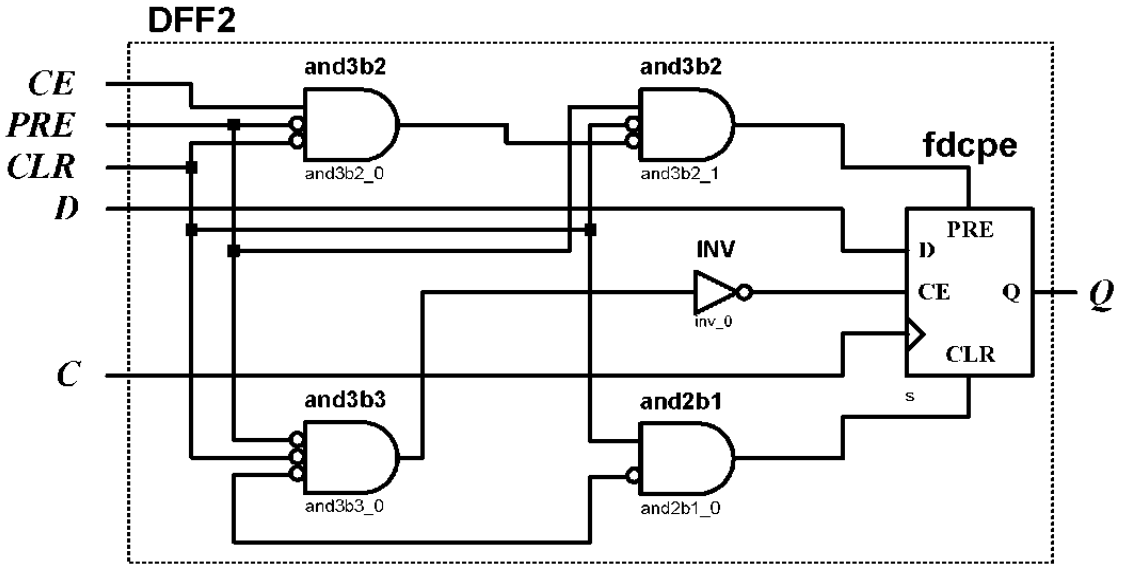



Рис. 2.24. Результат RTL-синтезу компоненти DFF2
з модифікованими умовами функціонування

Проектувальник має право використовувати готові бібліотечні примітиви для реалізації своїх проектів, але це не завжди є виправданим з точки зору сприйняття і читабельності поведінкових описів. В остаточному результаті проектувальник повинен чітко уявляти собі структурні особливості цільової технології реалізації свого проекту і співвідносити проектні описи з результатами їх синтезу. В іншому випадку можуть виникати проблеми не тільки з функціонуванням реалізованого пристрою, але і з його апаратними і часовими характеристиками, споживаною енергією тощо.

Виробивши конкретний стиль поведінкового опису базових схемотехнічних компонент, таких як тригери, проектувальник здатний використовувати його при описі більш складних пристроїв з очікуваними результатами синтезу для обраної технології.

Наведемо ще один приклад складання проектного опису Т-тригера, керованого переднім фронтом синхросигналу та з можливістю асинхронного скидання в нульовий стан. У проектувальника є кілька варіантів складання опису даного пристрою: використовувати бібліотечну компоненту **ftc** (Toggle Flip–Flop with Asynchronous Clear) або спробувати самостійно скласти поведінковий опис, наприклад, таким чином, як показано нижче.

```
- - …
- CLR — вхідний порт сигналу асинхронного скидання
- C — вхідний порт сигналу синхронізації
- T — вхідний порт керуючого сигналу
- Q — вихідний порт даних
- - …
signal s : std_logic;
Begin

Main: process ( CLR, C, T, s )
begin
if CLR = '1' then
s <= ' 0 ';
elsif T = '1' then
if rising_edge (C) then s <= not s;
end if ;
end if ;
end process;
Q <= s;
- - …
```

З огляду на те, що для FPGA SPARTAN-3E відсутній технологічний примітив Т-тригера, автоматизований засіб синтезу побудує послідовну схему на основі наявного примітиву D-тригера. При цьому, за наведеним описом, сигнал T буде використаний як сигнал дозволу синхронізації і буде сформовано негативний зворотній зв'язок між виходом D-тригера і його вихідним портом даних. Еквівалентна цифрова схема буде отримана при синтезі технологічного примітиву ftc, в основі якої також буде лежати D-тригер, але без управління синхросигналом (рис. 2.25).

Відновимо поведінковий опис примітиву ftc, маючи інформацію про структуру його схемної реалізації.

```
- - …
signal s : std_logic;
Begin
Main: process ( CLR, C, T, s )
begin
if CLR = '1 ' then
```

```
s <= ' 0 ' ;
elsif rising_edge (C) then
s <= s xor T;
end if ;
end process ;

Q <= s ;
- - …
```

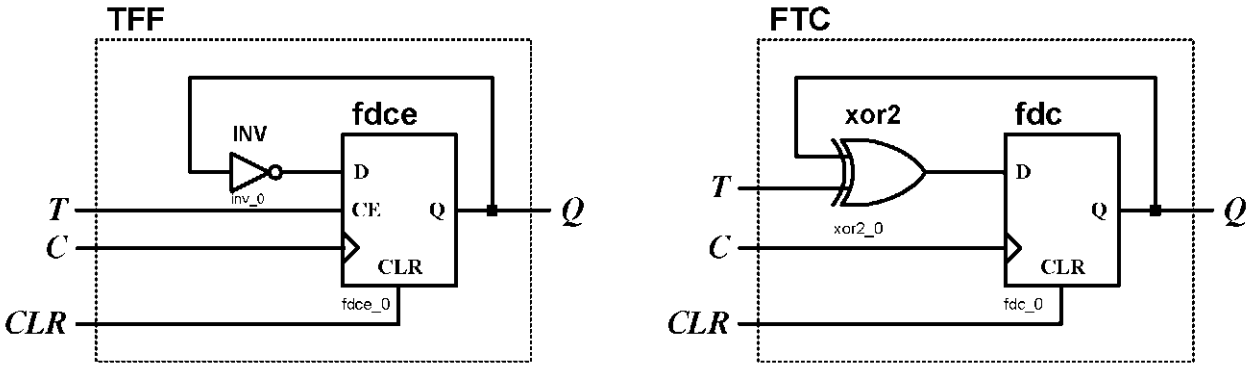



Рис. 2.25. Результати синтезу Т-тригера для FPGA SPARTAN-3E

Подані схемні результати синтезу можна інтерпретувати як симбіоз комбінаційних і послідовних схем, що підлягають роздільному опису. Якщо функціонування комбінаційної схеми описується разом з тригерною схемою, то існує ймовірність синтезу зайвих елементів пам'яті або додаткової комбінаційної логіки з відповідними негативними ефектами. Для коректного складання поведінкового опису можна скористатися наступною методикою, яка припускає роздільний опис комбінаційної частини пристрою і його елементів пам'яті. Розглянемо один з можливих прикладів такого роздільного опису.

```
- - …
- сигнал, що описує значення вихідного порту тригера
signal s : std_logic;
- сигнал, що описує значення виходу комбінаційної схеми
signal tx : std_logic;
Begin
```

```
- - опис комбінаційної схеми
tx <= s xor T;
- - опис елемента пам'яті (D-тригер)
DFF: process( CLR, C, tx )
begin
if CLR = '1' then
s <= '0';
elsif rising_edge( C ) then s <= t x ;
end if ;
end process;
-- вихідний комбінаційний каскад
Q <= s ;
- - …
```

У наведеному прикладі опис схеми поділено на три фрагменти: опис комбінаційної схеми вхідного каскаду, опис елемента пам'яті і опис вихідного комбінаційного каскаду. Результатом синтезу такого опису буде схема, що складається з трьох вхідних буферів-підсилювачів, двовхідного логічного елемента XOR2, D-тригера (процес DFF) і одного вихідного буфера підсилювача. Подібний роздільний опис дозволяє проектувальнику легко контролювати окремі складові цифрового пристрою. Наприклад, для додавання до вихідного каскаду тристабільного буфера-підсилювача необхідно внести зміну тільки в останній оператор.

```
- - . . . .
- - вихідний тристабільний буфер-підсилювач
Q <= s when OE= '0' else 'Z';
- - . . .
```

Якщо є необхідність видозмінити функціонування елемента пам'яті, то модифікувати слід опис процесу DFF, не зачіпаючи інші оператори.

Надалі намагатимемося слідувати наведеній методиці роздільного опису, яка може бути ефективно застосована для проектування регістрових схем і цифрових кінцевих автоматів.

## 2.5. Проектування регістрових схем

*Регістрові схеми* належать до класу послідовних схем і застосовуються в цифровій схемотехніці для різних цілей:

– зберігання даних (регістри зберігання, регістрові файли);

– перетворення кодів (наприклад перетворення послідовного коду в паралельний);

– генерування різних послідовностей (наприклад псевдовипадкової М-послідовності);

– стиснення даних (наприклад реалізації сигнатурних аналізаторів).

Регістрові схеми, як правило, будуються на основі базових запам'ятовуючих елементів: на основі засувок і тригерів. Через те, що схеми перетворювачів кодів, генераторів різних послідовностей і схеми стиснення даних вимагають багататактного функціонування, для їх проектування використовуються, як правило, тільки синхронні елементи пам'яті. Виняток становлять схеми зберігання даних, для проектування яких можливе застосування засувок.

### 2.5.1. Регістри зберігання

Регістри зберігання являють собою множину з n найпростіших елементів пам'яті, об'єднаних єдиними лініями управління і призначених для зберігання інформаційних слів розрядністю n. Регістри зберігання мають вхідну n-розрядну шину даних, на яку надходить значення інформаційного слова, і вихідну n-розрядну шину даних, на якій фіксуються значення розрядів збереженого слова. Структура найпростішого регістра зберігання, побудованого на основі D-засувок, подана на рис. 2.26.

Регістр має n-розрядні вхідні і вихідні шини даних *Din* і *Dout*, вхідний порт сигналу завантаження даних у регістр EN і побудований на основі елементів пам'яті типу ld.

Для складання VHDL-опису наведеної структури регістра можна скористатися структурним стилем і, наприклад, оператором for generate через те що структура регістра є ітераційною. Однак набільш переважним виглядає поведінковий опис, який можна скласти на основі існуючого опису D-засувки з дозволом (дод. 1, лістинг 2.8). Об'єднаємо вхідні порти регістра

*Din(i)* в єдиний порт *Din* типу std_logic_vector, аналогічно зробимо з описом вихідних портів, а сигнал дозволу G зробимо загальним для всіх елементів пам'яті (дод. 1, лістинг 2.7).

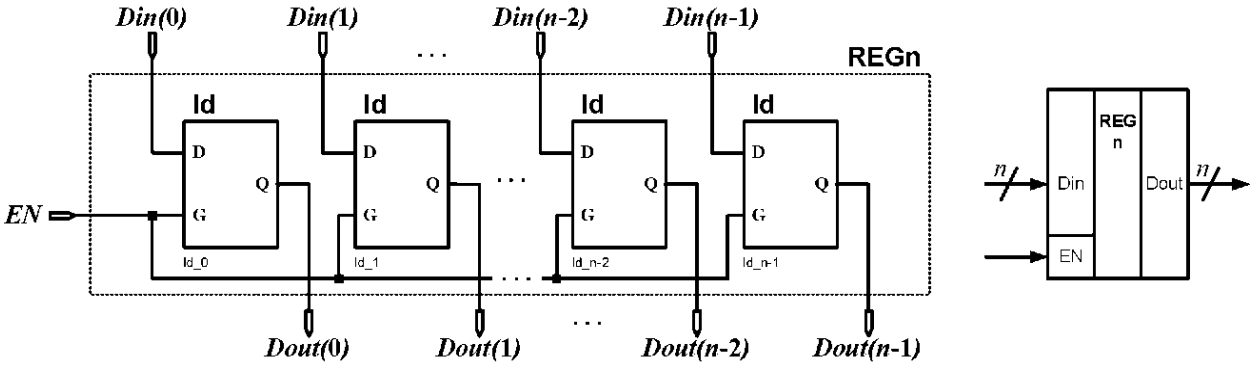



Рис. 2.26. Структура найпростішого регістра зберігання

Результатом синтезу наведеного опису буде структура, схожа з наведеною на рис. 2.26, що складається з чотирьох елементів ld.

Регістри зберігання також можуть мати синхронне управління, при цьому вони проектуються на основі відповідних D-тригерів. Наприклад, при синтезі синхронного 8-розрядного регістра зберігання з можливістю асинхронного скидання в нульове значення результуюча схема буде складатися з восьми D-тригерів, об'єднаних єдиною лінією сигналу асинхронного скидання.

Регістри зберігання можна форматувати не тільки нульовим словом, а й будь-яким іншим двійковим n-розрядним словом залежно від їх призначення. Крім того, регістри зберігання можуть мати у своїй структурі набір вихідних тристабільних буферів-підсилювачів для можливості їх підключення до єдиної шини передачі даних.

Доповнимо попередній приклад вхідним портом сигналу синхронізації CLK, вхідним портом сигналу управління OE, вихідними трихстабільними буферами і вхідним портом сигналу ініціалізації регістра INIT, двійковою константою *IN IT REG* (дод. 1, лістинг 2.8). Для складання налагоджуваного опису синхронного регістра був використаний generic-об'єкт INITREG безрозмірного типу std_logic_vector, ініціалізований константним чотирирозрядним вектором «1001». Розмірність константного

вектора і визначає кількість розрядів регістра за допомогою оператора **range.**

У декларативній частини архітектури введена додаткова константа ALLZ, яка визначає вектор типу std_logic_vector, у кожному розряді якого записано значення 'Z'. Цей вектор використовується для опису функціонування тристабільних буферів вихідної шини даних регістра. Функціонування самого регістра описано процесом Main, який схожий з описом D-тригера, керованого переднім фронтом сигналу синхронізації, з можливістю асинхронного скидання/установлення і з сигналом, що дозволяє синхронізацію (примітив fdce). На рис. 2.27 наведено результат RTL-синтезу синхронного регістра з умовою, що *INITRER* :=«1001».

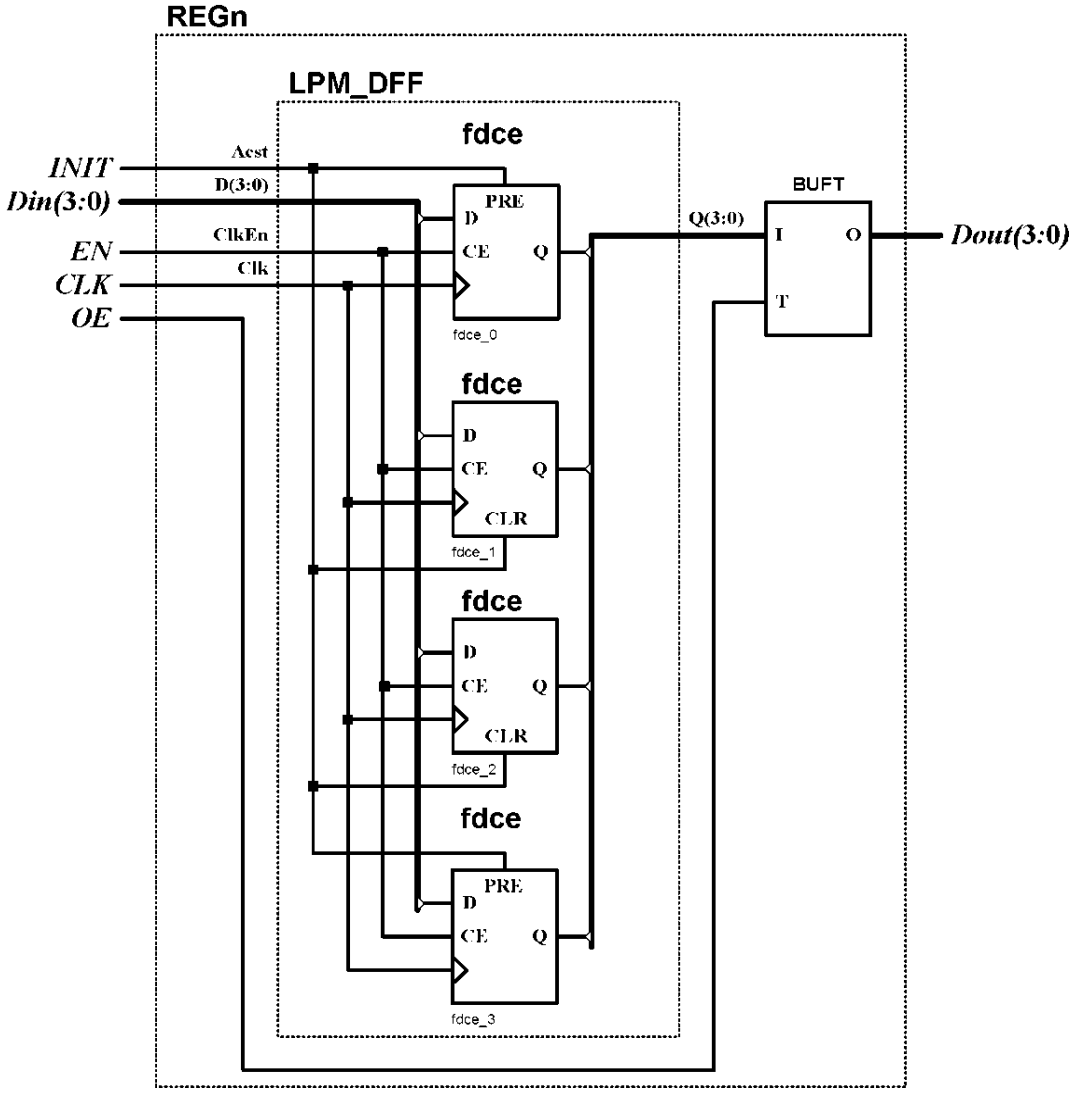



Рис. 2.27. Результат RTL-синтезу синхронного регістра зберігання

Схемна реалізація регістра включає чотири тригери (примітиви fdce), ініціалізація яких здійснюється згідно з використаним значенням *INITREG:* тригери fdce_0 і fdce_3 встановлюються в середнє арифметичне значення шляхом подачі керуючого сигналу з порту *INIT* на входи *PRE*, а тригери fdce_1 і fdce_2 скидаються в нульовий стан шляхом подачі керуючого сигналу на входи *CLR.*

Виходи всіх тригерів об'єднані в єдину шину даних *Q* (3:0), яка надходить на вхід *I* блока BUFT, що є макроблоком, який складається з чотирьох тристабільних буферів із загальним керуючим входом T, на який надходить сигнал з порту *OE* всієї регістрової схеми REGn. Вихідна чотирирозрядна шина *O* безпосередньо підключена до відповідних розрядів порту *Dout(3*:0).

У процесі RTL-синтезу наведений опис було розпізнано як синхронний регістр зберігання, реалізація якого здійснена у вигляді бібліотечного параметризованого модуля LPM_DFF (Library of Parameterized Modules), у цьому прикладі, що складається з чотирьох D-тригерів (DFF).

### 2.5.2. Регістрові файли

На основі схем регістрів зберігання можлива побудова запам'ятовуючих пристроїв типу *регістровий файл.* У **мікропроцесорній** техніці під регістровим файлом розуміють модуль, який реалізує регістри **мікропроцесора** і являє собою багатопортовий масив регістрів зберігання. На відміну від традиційних запам'ятовуючих пристроїв, регістровий файл має явний поділ портів запису і читання даних. Наприклад, для виконання трьохоперандних інструкцій (часто зустрічаються в RISC-**мікропроцесорах**) необхідна наявність регістрового файла з трьома портами: два порти читання і один порт запису, які дозволяють виконувати одночасне читання значень аргументів з двох регістрів і здійснювати запис результату в третій регістр.

Розглянемо приклад проектування двопортового регістрового файла (один порт запису і один порт читання) на основі компоненти REGn (дод. 1, лістинг 2.8). Для зручності адресації і виключення некоректних операцій запису і читання даних регістрові файли проектуються на основі $2^a$ (a = 0,1,2,...)

регістрів, при цьому їх адресація здійснюється за допомогою a-розрядних адресних шин. З урахуванням того, що операції запису і читання для регістрового файла можна здійснювати одночасно, необхідна наявність таких портів: вхідний порт шини даних для запису (WDP, Write Data Port), вихідний порт шини прочитаних даних (RDP, Read Data Port), вхідний порт шини адреси регістра для запису (WA, Write Address) і вхідний порт шини адреси регістра для читання (RA, Read Address). При використанні компоненти REGn умовимось, що операція запису даних до регістрового файла буде синхронною, а операція читання даних – асинхронною. Для можливості незалежної адресації двох портів у структурі регістрового файла необхідна наявність двох адресних дешифраторів – дешифратор адреси порту запису (WAD, Write Address Decoder) і дешифратор адреси порту читання (RAD, Read Address Decoder). З урахуванням вищевикладеного загальну структуру регістрового файлу можна подати так, як показано на рис. 2.28.

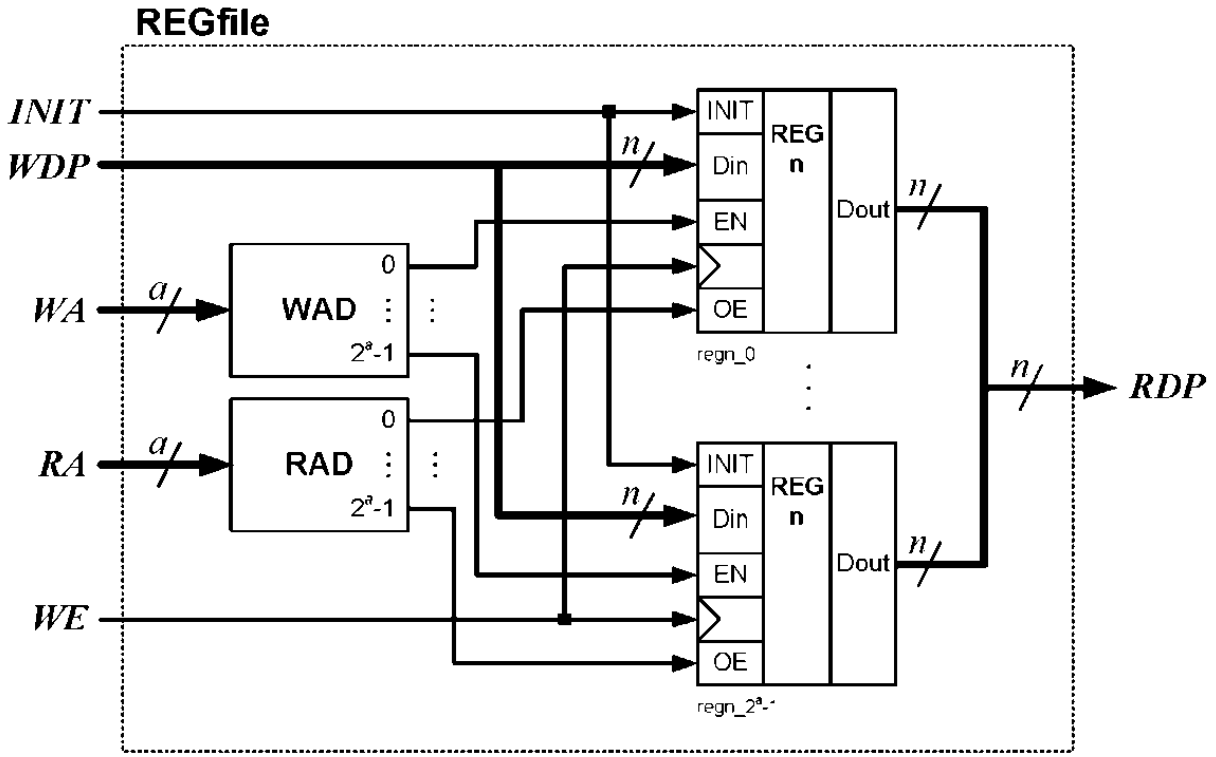



Рис. 2.28. Загальна структура двопортового регістрового файла

Через те, що компоненти REGn мають вихідні тристабільні буфери, їхні порти *Dout* можна об'єднати в єдину вихідну шину реєстрового файла RDP. Для опису наведеної структури мовою VHDL скористаємося змішаним стилем, при цьому множину регістрів опишемо структурним стилем, а інші компоненти – поведінковим (дод. 1, лістинг 2.9).

Як і в попередньому прикладі, визначимо розмірності кожного регістра за допомогою безрозмірного generic-об'єкта *INITREG.* За замовчуванням описаний регістровий файл складається з чотирьох регістрів (a = 2), при цьому кожен регістр складається з чотирьох тригерів *(IN IT REG'range),* кожен з яких ініціюється нульовим значенням («0000»).

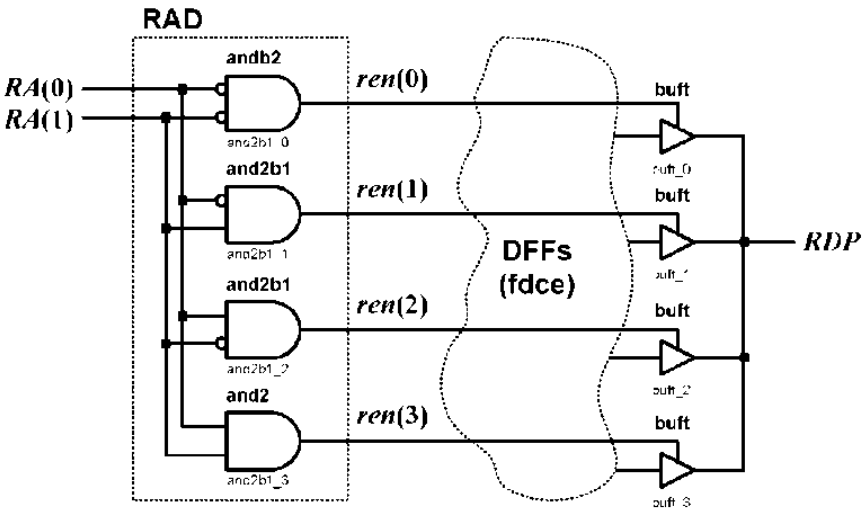
Результатом RTL-синтезу наведеного опису будуть чотири компоненти REGn (у сукупності 16 примітивів fdce і 16 тристабільних буферів) і комбінаційні схеми двох адресних дешифраторів.

При спробі здійснення технологічного синтезу для FPGA SPARTAN-3E може бути видано попередження «*WARNING:Xst:2040 – Unit REGFile: 4 multi–source signals are replaced by logic»,* яке означає, що через відсутність у внутрішніх ресурсах FPGA тристабільних буферів, засіб синтезу замінив їх у результуючій схеми на еквівалентні логічні схеми. Зазвичай такими схемами є схеми мультиплексорів.

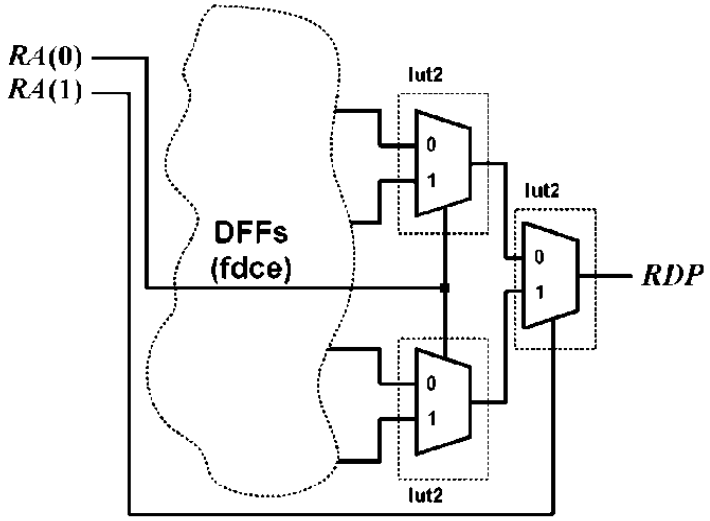
В обох схемних реалізаціях будуть використані чотири тригери fdce. Результатом RTL-синтезу двох адресних дешифраторів будуть комбінаційні схеми, що складаються з чотирьох логічних елементів AND2 і чотирьох інверторів (рис. 2.29).

При технологічному синтезі схема дешифратора адреси читання трансформується в комбінаційну схему вибору значення сигналу від чотирьох джерел, якими є вихідні порти D-тригерів.

Регістрові файли можуть бути реалізовані не тільки з різною кількістю портів, але і з різним типом функціонування. Наприклад, на відміну від розглянутої реалізації, обидві операції запису і читання можуть бути синхронними з додатковими сигналами дозволу тощо.




а




б

Рис. 2.29. Еквівалентні схеми адресації чотирьох тригерів: результат RTL-синтезу (а), результат технологічного синтезу (б)

### 2.5.3. Зсувні регістри

До окремого класу регістрових схем належать *зсувні регістри,* або регістри зсуву, які часто застосовуються на практиці для здійснення перетворення послідовних кодів у паралельні і навпаки. *Під послідовним кодом* розуміють спосіб передачі даних, при якому окремі біти інформаційного слова передаються з часовим поділом. Послідовний код застосовується для реалізації послідовних інтерфейсів між цифровими пристроями з метою скорочення кількості приймально-

передавальних ліній. У той час як цифрові пристрої зазвичай оперують даними, поданими в паралельному коді (байти, слова, подвійні слова тощо), то для передачі і приймання інформації за допомогою послідовних інтерфейсних ліній необхідно здійснювати вищеописані перетворення кодів.

В основі схемних реалізацій подібних перетворювачів лежать регістрові синхронні схеми, що відрізняються від регістрів зберігання тим, що утворюють послідовні з'єднання тригерів для можливості здійснення операції зсуву.

Структуру найпростішого зсувного регістра можна отримати, взявши за основу структуру n-розрядного регістра зберігання, подану на рис. 2.26. Змінимо регістр зберігання таким чином: замість D-засувок ld будемо використовувати синхронні D-тригери fdce, вихідний порт Q і-го тригера, за винятком останнього, з'єднаємо з вхідним портом D-тригера з індексом *i + 1* і з відповідним вихідним портом *Pout(i),* замість вхідної шини даних *Din* сформуємо один вхідний порт *Sin,* підключений до входу першого тригера з індексом 0, вхідні порти сигналу синхронізації і сигналу CE всіх тригерів об'єднаємо в єдині лінії управління *CLK* і *SE* відповідно (рис. 2.30).

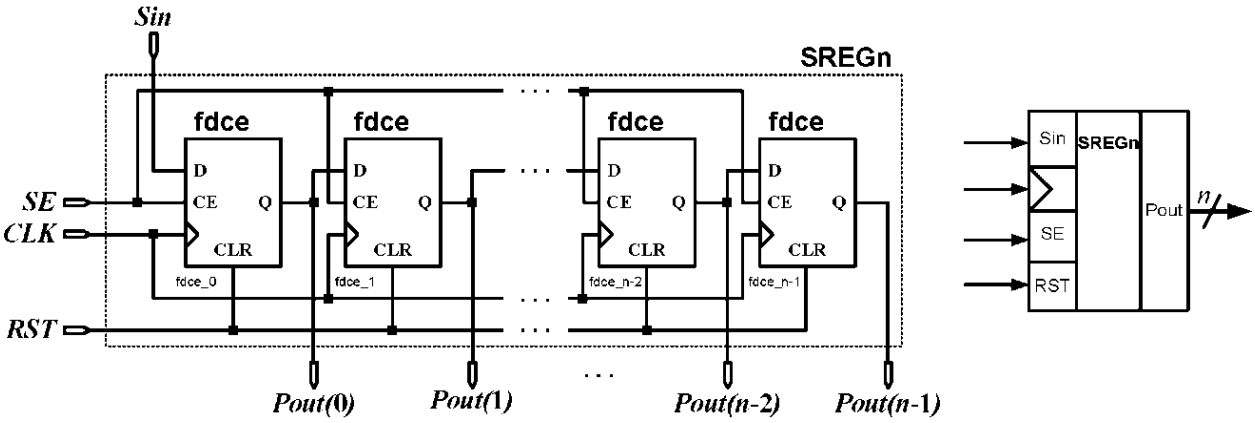



Рис. 2.30. Структура найпростішого зсувного регістра

Розглянемо функціонування наведеної схеми. При неактивному рівні сигналу на вхідному порті *SE* схема виконує роль n-розрядного регістра зберігання, при цьому на вихідній шині даних фіксується паралельний код, що зберігається на D-тригерах. У разі, коли SE = 1, при настанні фронту синхросигналу на порту *CLK* на тригері fdce_0 фіксується

значення сигналу вхідного порту *Sin*, а на всіх інших тригерах fdce_i – значення сигналів з вихідних портів Q тригерів fdce_i — 1. Таким чином, після закінчення n періодів синхросигналу *CLK* при SE = 1 значення, що зберігається на всіх тригерах, буде визначатися значеннями сигналу *Sin* в моменти настання переднього фронту синхросигналу. Іншими словами, подаена схема виконує роль перетворювача послідовного коду, що надходить на вхід *Sin*, у паралельний код, значення, якого фіксується на вихідний шині даних *Pout.*

Існує безліч варіантів опису розглянутого цифрового пристрою мовою VHDL. Розглянемо один з них, використавши структурний стиль опису і RTL-примітив fdce з бібліотеки Unisim [61] (дод. 1, лістинг 2.10).

Зворотне перетворення паралельного коду в послідовний можливе також за допомогою зсувного регістра, який має вхідну шину даних для завантаження паралельного коду і однорозрядний вихідний порт для вироблення значень розрядів послідовного коду.

Подібні цифрові схеми називаються PISO-регістрами (від англ. Parallel Input – Serial Output register). У дод. 1 в лістингу 2.11 наведено вихідний код компоненти PISOn, що є n-розрядним перетворювачем паралельного коду в послідовний. Поданий опис є роздільним, у якому процесом Main описано поведінку множини D-тригерів (аналоги примітивів fdc), а процесом Data – комбінаційна схема, яка відповідає за формування нових даних на входах всіх тригерів (внутрішня шина *sdat).* Так, при значенні керуючого сигналу *LS = 0* на входи тригерів надходять значення сигналів з вхідної шини даних *Pin.* При *LS* = 1 на шині *.sdat* формується код, значення якого дорівнює коду, що зберігається на тригерах і зсунутий на одну позицію вправо. Значення молодшого розряду сформованого коду при цьому дорівнює значенню сигналу на вході *Pin(0).*

Таким чином, процес Data описує множину з n-1 двовходових мультиплексорів, керованих значенням сигналу LS. На рис. 2.31 подано результат технологічного синтезу наведеного опису для випадку *n* = 3.

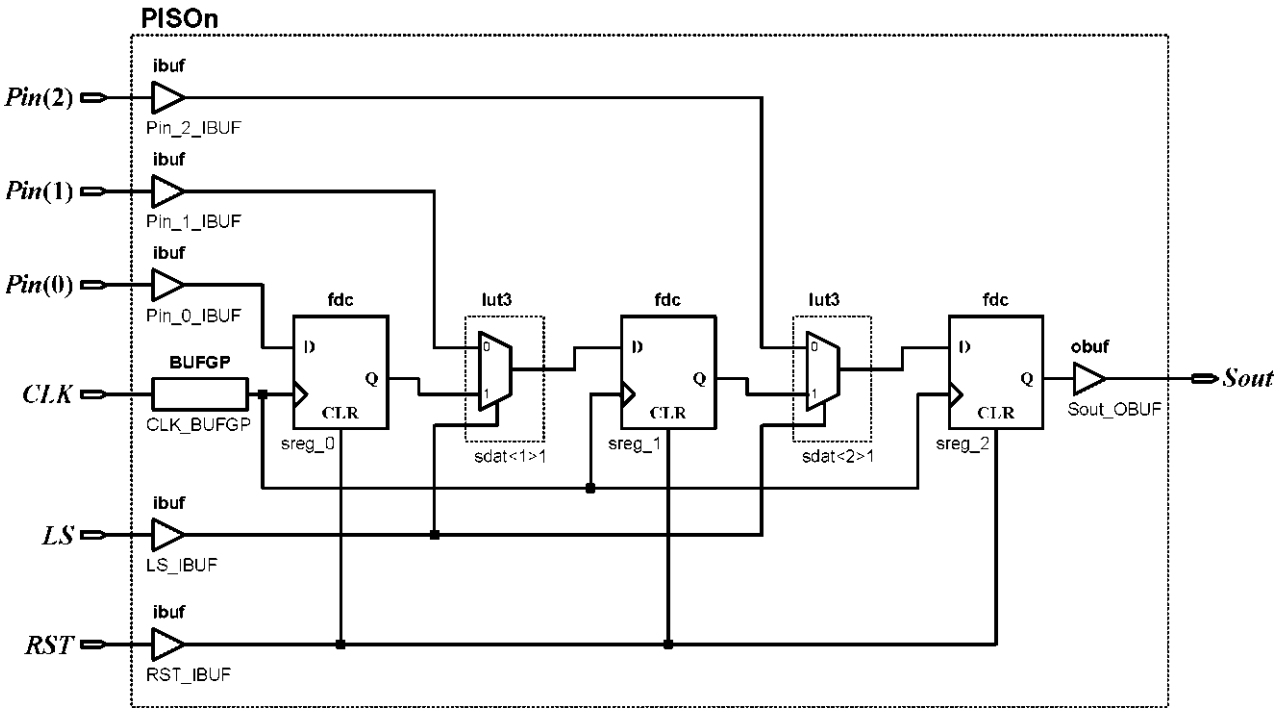



Рис. 2.31. Результат технологічного синтезу зсувного регістра для випадку $n = 3$

Як видно з наведеної схеми, при здійсненні операції зсуву значення молодшого розряду буде послідовно копіюватися в усі інші старші розряди регістра. При копіюванні певного фіксованого значення з молодшого розряду буде ускладнена схемна реалізація розглянутого пристрою через синтез додаткового lut-блока, що реалізує комбінаційну схему управління входом D-тригера sreg_0.

Зсувні регістри також можуть виконувати операцію циклічного зсуву, при якому значення молодшого розряду заміняється на попереднє значення старшого розряду. Для розглянутої схеми це можливе шляхом реалізації позитивного зворотного зв'язку між виходом Q старшого тригера і входом D молодшого тригера при здійсненні операції зсуву.

```
- - . . .
—— умова завантаження паралельних даних
if LS = '0' then
sdat <= Pin ;
- - умова зсуву регістра на один розряд вправо
else
```

```
sdat <= sreg ( n-1 ) & sreg ( 0 to n-2 );
end if ;
- - . . .
```

Подібні регістри називаються *циклічними зсувними регістрами*, або зсувними регістрами з лінійним зворотним зв'язком **LFSR** (від англ. Linear Feedback Shift Register), за допомогою яких можлива схемна реалізація генераторів різних числових послідовностей. Розглянемо структурні особливості та основні властивості функціонування подібних генераторів.

### 2.5.4. Генератори числових послідовностей

Лінійний зв'язок у LFSR-структурах досягається шляхом лінійних перетворень над значенням сигналу в ланцюзі зворотного зв'язку (сумуванням за модулем 2). Розглянемо найпростіший випадок, при якому значення сигналу в ланцюзі зворотного зв'язку лінійно перетворюється за допомогою константи c€{0,1} (рис. 2.32).

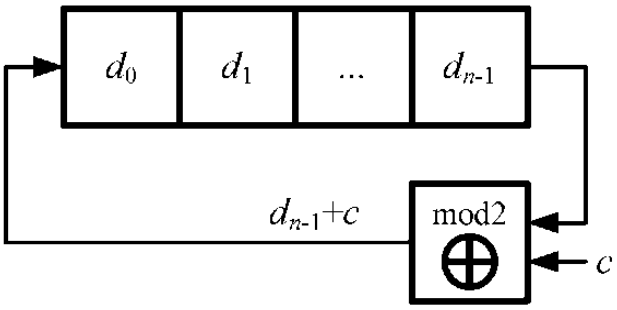



Рис. 2.32. Структура найпростішого LFSR

Припустимо, що в початковий момент часу значення, збережене на тригерах регістра зсуву, подано як $n$-мірний двійковий вектор $D = (d_0, d_1, \dots, d_{n-1})$. У перший такт роботи схеми (при настанні фронту сигналу синхронізації) значення розрядів вектора змінюється:

$$d_i = d_{i-1}, \forall\, i \in \{1, n-1\}.$$

Значення нульового розряду обчислюється як $d_0 = d_{n-1} + c$. Значення двійкового вектора D після *k*-го такту роботи зсувного регістра можна подати в матричній формі:

$$\begin{vmatrix} d_0^{(k)} \\ d_1^{(k)} \\ \dots \\ d_{n-1}^{(k)} \end{vmatrix} = \begin{vmatrix} 0 & 0 & \dots & 1 \\ 1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{vmatrix} \times \left( \begin{vmatrix} d_0^{(k-1)} \\ d_1^{(k-1)} \\ \dots \\ d_{n-1}^{(k-1)} \end{vmatrix} + \begin{vmatrix} 0 \\ 0 \\ \dots \\ c \end{vmatrix} \right) \tag{2.6}$$

або в більш компактному вигляді $D^k = V(D^{k-1} + C)$, де

$$V = \begin{vmatrix} 0 & 0 & \dots & 1 \\ 1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{vmatrix}, C^T = \begin{vmatrix} 0 \\ 0 \\ \dots \\ c \end{vmatrix}. \tag{2.7}$$

При значенні c = 0 схема LFSR являє собою генератор циклічного коду, що виробляє n-розрядні двійкові слова, період повторення яких залежить від кількості розрядів n і від початкового стану генератора. Наприклад, для n = 4 генератор циклічного коду виробляє різні числові послідовності, значення періоду яких варіюється від 1 до 4 і визначається початковим станом генератора. Так, для початкового стану $D^{(0)}$ = (1,0,0,0) послідовно виробляється така числова послідовність: $D^{(i)}$ = (0,1,0, 0), $D^{(2)}$ = (0,0,1,0), $D^{(3)}$ = (0,0,0,1). Подальший стан генератора $D^{(4)}$ = (1,0,0,0) дорівнюватиме початковому стану. Таким чином, період виробленої послідовність $P_{(1;0;0;0)}$ дорівнює 4. Подібні циклічні послідовності прийнято називати орбітами O, довжина яких L(O) дорівнює періоду вироблюваної послідовності P. Для розглянутого прикладу генератор ділить весь простір 4-мірних векторів на шість орбіт: O(i,0,0,0), $O(0,i,_M)$, $O(i_{Am})$, O(i,0,i,0), 0(0,0,0,0) та $O(_{M,M})$ (рис. 2.33).

Орбіти $O(_{0,0,00})$ та $O(_{i,i,i,i})$ є виродженими, оскільки містять тільки по одному елементу послідовності. Максимальну довжину мають такі орбіти: $L(O(_{i,0,0,0}))=L(O(_{0,i,i,i}))=L(O(_{i,i,0,0})) = 4$, а орбіта $O(_{i\ i\ 0,0})$ має довжину 2.

На практиці найчастіше застосовуються n-розрядні генератори циклічного коду з орбітою $O(_{io...o})$, які по-іншому називаються генераторами *one-hot* послідовності. Розглянемо

приклад проектування подібного генератора на основі наступного поведінкового VHDL-опису (дод. 1, лістинг 2.12).

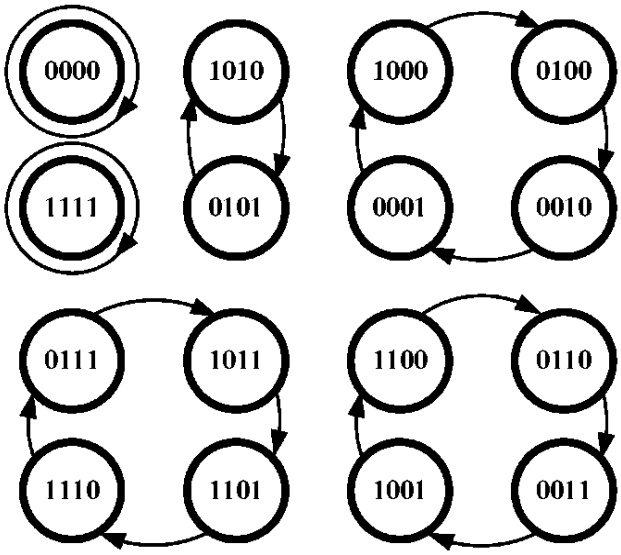



Рис. 2.33. Приклад орбіт генератора циклічного коду для $n = 4$

Якщо в ланці зворотного зв'язку LFSR значення константи с встановити рівним одиниці, то генератор почне виробляти *послідовність Джонсона*, при цьому сам генератор називається *лічильником Джонсона* [62]. Доведено, що максимальний період послідовності Джонсона дорівнює $2^n$ для всіх $n = 2^i$, (i = 1, 2,...) і для довільного початкового стану генератора [63]. Наприклад, для $n = 4$ весь простір бінарних векторів ділиться на дві орбіти довжиною 8 (рис. 2.34), а для $n = 15$ довжини орбіт різні (2, 6, 10, 30) залежно від початкового стану лічильника Джонсона.

Спроектуємо компоненту n-розрядного лічильника Джонсона ($n = 2^r$) з можливістю задавання довільного початкового стану та ініціалізації в нульовий стан (дод. 1, лістинг 2.13).

Лічильники Джонсона використовуються як генератори псевдовипадкових послідовностей для тестування цифрових пристроїв, особливо для неруйнівного тестування ОЗП [63]. *Псевдовипадкові послідовності* (ПВП) являють собою числові послідовності, частотні, імовірнісні та автокореляційні характеристики яких схожі з характеристиками істинно випадкових послідовностей, за винятком того, що вони є відтворюваними і циклічними. Серед великого різноманіття генераторів ПВП найчастіше використовуваними є генератори М-послідовностей.

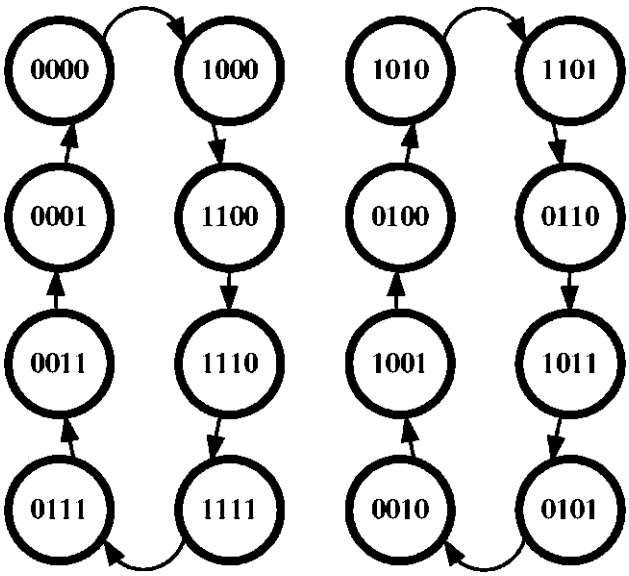



Рис. 2.34. Орбіти лічильника Джонсона для $n = 4$

*М-послідовність* – це циклічна ПВП з максимальним періодом, значення якої залежить від розмірності символів послідовності (розрядності генератора) опису.

При побудові подібних генераторів ПВП використовуються LFSR-структури, у яких формування значення в лінії зворотного зв'язку здійснюється на основі значень розрядів зсувного регістра генератора [64]. Вибір використовуваних значень розрядів генератора здійснюється на основі коефіцієнтів $\alpha_i \in \{0, 1\}; i = \overline{0, n}$ (рис. 2.35).

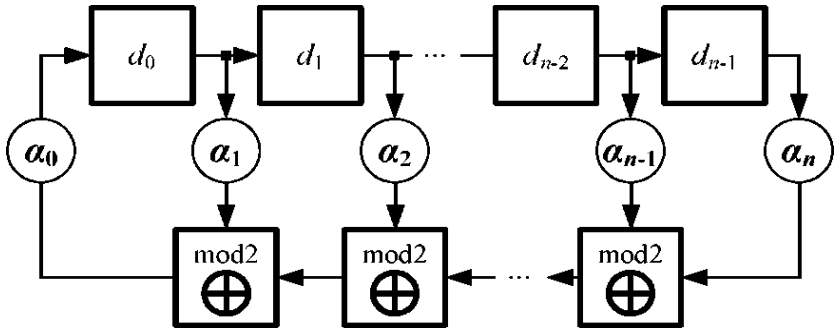



Рис. 2.35. Загальна структура генератора М-послідовності

Якщо коефіцієнт a = 1, то відповідний розряд $d_{i+1}$ регістра зсуву бере участь у формуванні нового значення молодшого розряду. В іншому випадку значення, що зберігається в розряді $d_{i+1}$, не використовується. Для розглянутих генераторів ПВП

$a_0 = a_n = 1$. У загальному випадку циклічні властивості розглянутого генератора визначаються характеристичним поліномом такого вигляду:

$$\varphi(x) = \theta \sum_{k-0}^{n} a_k \times x^k \tag{2.8}$$

У матричному поданні значення, що зберігається на зсувному регістрі генератора М-послідовності, має такий вигляд:

$$\begin{vmatrix} d_0^{(k)} \\ d_1^{(k)} \\ \dots \\ d_{n-1}^{(k)} \end{vmatrix} = \begin{vmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_{k-1} & \alpha_k \\ 1 & 0 & \cdots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 1 & 0 \end{vmatrix} \times \begin{vmatrix} d_0^{(k-1)} \\ d_1^{(k-1)} \\ \dots \\ d_{n-1}^{(k-1)} \end{vmatrix} \tag{2.9}$$

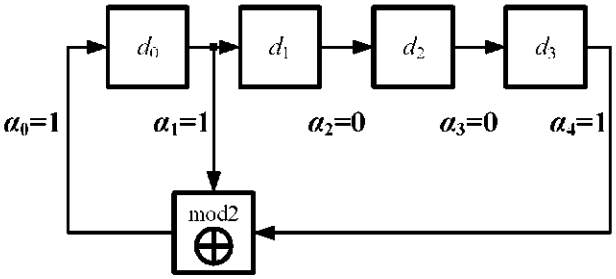
або в компактній формі $D^{(k)} = VD^{(k-1)}$, де

$$V = \begin{vmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_{k-1} & \alpha_k \\ 1 & 0 & \cdots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 1 & 0 \end{vmatrix}. \tag{2.10}$$

Період ПВП, що виробляється генератором, залежить від примітивності і незведеності полінома $\varphi(x)$. Послідовності максимальної довжини з періодом $2^n$ — 1 формуються лише в тому випадку, коли характеристичний поліном $\varphi(x)$ є примітивним і незведеним. Багаточлен $\varphi(x)$ $n$-го порядку ($n = \deg \varphi(x)$) є примітивним, оскільки поліном $x^k$ — 1 поділяється на поліном $\varphi(x)$ тільки при $k = 2^n$ — 1. Незведеним поліномом $\varphi(x)$ ступеня n є поліном, який не ділиться ні на який інший багаточлен від x зниженого ступеня. Властивістю розроблення М-послідовності також є ненульовий початковий стан регістра зсуву, інакше послідовність, що виробляється, буде складатися з одних нулів.

Розглянемо приклад проектування генератора М-послідовності, символами якої є 4-розрядні двійкові слова (n = 4). Виберемо примітивний незведений поліном четвертого ступеня, за яким будемо визначати структуру ланцюга зворотного зв'язку LFSR: $n = 4 = \deg \varphi(x)$, $\varphi(x) = 1 \ominus x \oplus x^4$. Згідно з формулою (2.9)

для обраного полінома ^(x) значення коефіцієнтів α будуть такими: $\alpha_0 = \alpha_4 = 1, \alpha_1 = 1, \alpha_2 = \alpha_3 = 0$. Виходячи з цього структура генератора М-послідовності і сама послідовність будуть виглядати так, як це показано на рис. 2.36.




| | $d_0$ | $d_1$ | $d_2$ | $d_3$ |
|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 2 | 1 | 1 | 1 | 0 |
| 3 | 1 | 1 | 1 | 1 |
| 4 | 0 | 1 | 1 | 1 |
| 5 | 1 | 0 | 1 | 1 |
| 6 | 0 | 1 | 0 | 1 |
| 7 | 1 | 0 | 1 | 0 |

| | $d_0$ | $d_1$ | $d_2$ | $d_3$ |
|---|---|---|---|---|
| 8 | 1 | 1 | 0 | 1 |
| 9 | 0 | 1 | 1 | 0 |
| 10 | 0 | 0 | 1 | 1 |
| 11 | 1 | 0 | 0 | 1 |
| 12 | 0 | 1 | 0 | 0 |
| 13 | 0 | 0 | 1 | 0 |
| 14 | 0 | 0 | 0 | 1 |
| 15 | 1 | 0 | 0 | 0 |

Рис. 2.36. Структура генератора і символи М-послідовності для $\varphi(x) = 1 \odot x \odot x^4$

При початковому стані генератора $D^{(0)}$ = (1,0,0,0) після першого такту функціонування формується новий стан

$$D^{(1)} = VD^{(0)} = \begin{vmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{vmatrix} \times \begin{vmatrix} 1 \\ 0 \\ 0 \\ 0 \end{vmatrix} = \begin{vmatrix} 1 \\ 1 \\ 0 \\ 0 \end{vmatrix}. \qquad (2.11)$$

На 15-му такті функціонування формується початковий стан генератора $D^{(i5)} = D^{(0)}$, при цьому період сформованої послідовності дорівнює $2^n - 1 = 15$, а множина символів сформованої послідовності практично становить весь векторний простір 4-мірних двійкових векторів, за винятком нульового вектора. Для можливості генерування всіх можливих 4-мірних векторів модернізуємо схему генератора таким чином, як на рис. 2.37.

При досягненні стану $D^{(14)}$ = (0,0,0,1) на виході логічного елемента nor3 сформується значення логічної одиниці, яке в сумі зі сформованим новим одиничним значенням молодшого розряду зсувного регістра визначить нульовий стан генератора $D^{(15)}$ = (0,0,0,0).

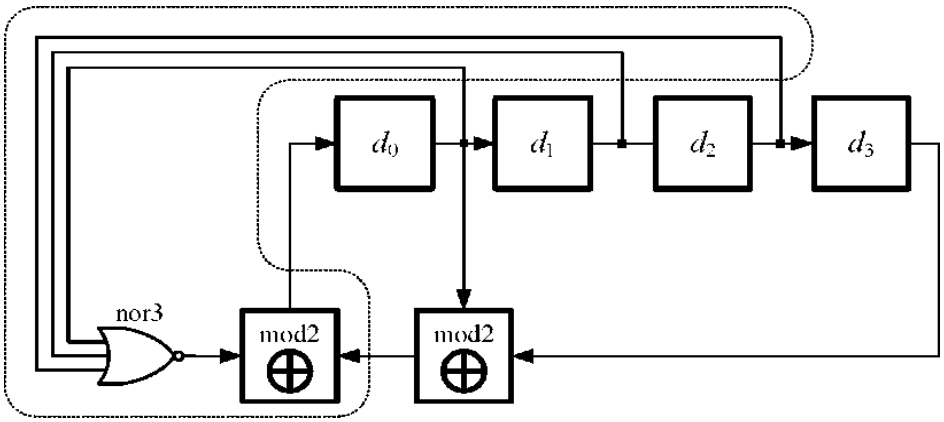



Рис. 2.37. Модифікований генератор М-послідовності для n = 4

Наступний стан молодшого розряду генератора обчислюється аналогічно: $d_0^{(16)} = \left(\overline{d_0^{(15)} + d_1^{(15)} + d_2^{(15)}}\right) \odot d_0^{(15)} \odot d_3^{(15)} = 1$, що призводить до формування початкового стану $D^{(16)} = D^{(0)} = (1,0,0,0)$.

При всіх інших значеннях розрядів $d_0$, $d_1$ та $d_2$ генератор буде виробляти символи М-послідовності, ідентичні поданим на рис. 2.36.

Спроектуємо компоненту n-розрядного генератора М-послідовності з можливістю визначення коефіцієнтів і характеристичного полінома і вироблення псевдовипадкових послідовностей з періодом, рівним $2^n$ (дод. 1, лістинг 2.14).

Для генерування М-послідовностей можна використовувати LFSR-структури з суматорами за модулем два, включеними в міжрозрядні зв'язки зсувного регістра [64]. Включення суматорів за модулем два також визначається коефіцієнтами характеристичного примітивного незведеного полінома p(x) (рис. 2.38).

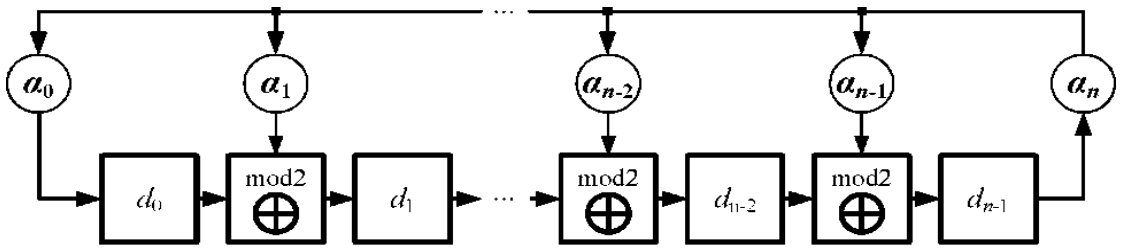
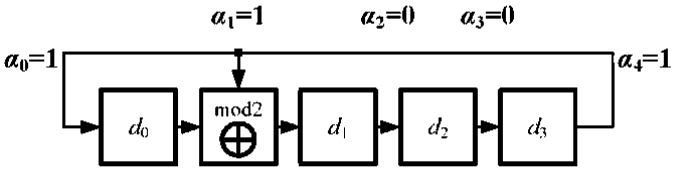



Рис. 2.38. Альтернативна структура генератора М-послідовності

Наприклад, для характеристичного полінома $\varphi(x) = 1 \ominus x \ominus x^4$ генератор М-послідовності буде складатися з регістра зсуву з кількістю розрядів, рівною $\deg \varphi(x) = 4$, і одного суматора за модулем два, включеного до зв'язку між нульовим і першим розрядами: $\alpha_0 = \alpha_4 = 1$, $\alpha_1 = 1$, $\alpha_2 = \alpha_3 = 0$ (рис. 2.39).

При нульовому коефіцієнті a ($i$ = 1,n — 1) нове значення $i$-го розряду зсувного регістра буде визначатися значенням, що зберігається в розряді з індексом $i$ — 1.

У лістингу 2.15 (дод. 1) подано VHDL-опис універсальної компоненти генератора М-послідовності з альтернативною структурою на основі суматорів за модулем два, включених до міжрозрядних зв'язків.




| | $d_0$ | $d_1$ | $d_2$ | $d_3$ |
|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 0 | 1 |
| 4 | 1 | 1 | 0 | 0 |
| 5 | 0 | 1 | 1 | 0 |
| 6 | 0 | 0 | 1 | 1 |
| 7 | 1 | 1 | 0 | 1 |

| | $d_0$ | $d_1$ | $d_2$ | $d_3$ |
|---|---|---|---|---|
| 8 | 1 | 0 | 1 | 0 |
| 9 | 0 | 1 | 0 | 1 |
| 10 | 1 | 1 | 1 | 0 |
| 11 | 0 | 1 | 1 | 1 |
| 12 | 1 | 1 | 1 | 1 |
| 13 | 1 | 0 | 1 | 1 |
| 14 | 1 | 0 | 0 | 1 |
| 15 | 1 | 0 | 0 | 0 |

Рис. 2.39. Альтернативна структура генератора і символи М-послідовності для $\varphi(x) = 1 \ominus x \ominus x^4$

### 2.5.5. Сигнатурні аналізатори

Крім генерування псевдовипадкових числових послідовностей, застосування примітивних незведених поліномів тісно пов'язано з *сигнатурним аналізом*, який є одним із часто застосовуваних методів дослідження реакцій цифрових пристроїв на тестові впливи [64]. У загальному вигляді аналіз реакцій за допомогою сигнатурного аналізу зводиться до отримання компактної оцінки (сигнатури) множини послідовних даних, що виробляються на виходах і/або внутрішніх вузлах тестованого пристрою. Отримання сигнатури потоку даних довільної довжини описується процесом розподілу багаточленів над

двійковим полем GF(2). Так, послідовний потік даних, що складається з l двійкових символів, може бути поданий як багаточлен p(x) ступеня l — 1. Наприклад, двійкова послідовність 11000011 довжиною l = 8 подається як багаточлен $p(x) = 1 \oslash x \odot x^6 \oslash x^7, deg\ p(x) = 7$.

Процес отримання сигнатури поданої послідовності може бути описаний процесом розподілу багаточлена $p(x)$ на примітивний незведений поліном $\varphi(x)$. У результаті такого поділу отримують багаточлени частки q(x) і остачі s(x), пов'язані між собою таким співвідношенням:

$$p(x) = q(x)\varphi(x) \odot s(x). \qquad (2.12)$$

Багаточлен остачі $s(x)$ є сигнатурою двійкової послідовності, поданої як багаточлен $p(x)$. Завдяки високому ступеню достовірності та відносно простій апаратній реалізації сигнатурний аналіз знайшов своє широке застосування не тільки в контролі і діагностиці засобів обчислювальної техніки, а й в апаратній криптографії і засобах захисту інформації. Пристрої, що дозволяють отримувати компактні сигнатури двійкових послідовностей, називаються *сигнатурними аналізаторами*. Як було показано в роботі [64], процеси, що протікають у генераторах М-послідовностей і сигнатурних аналізаторах, є ідентичними. Так, за основу проектування одноканального сигнатурного аналізатора можна взяти n-розрядний генератор М-послідовності, описаний характеристичним поліномом $p(x)$. Символи аналізованої послідовності підсумовуються за модулем два зі значенням старшого розряду генератора, тим самим формуючи нове значення його молодшого розряду (рис. 2.40).

Для проектування найпростіших одноканальних сигнатурних аналізаторів достатньо в VHDL-опис генератора М-послідовності (дод. 1, лістинг 2.15) додати вхідний однорозрядний порт послідовних даних для стиснення, значення якого необхідно скласти за модулем два з новим значенням молодшого розряду зсувного регістра:

```
- - . . .
— вхідний порт стискуваних даних
Sin : in std_logic;
```

```
- - . . .
sdat( 0 ) <= Sin xor sreg( alpha'high -1 );
- - . . .
```

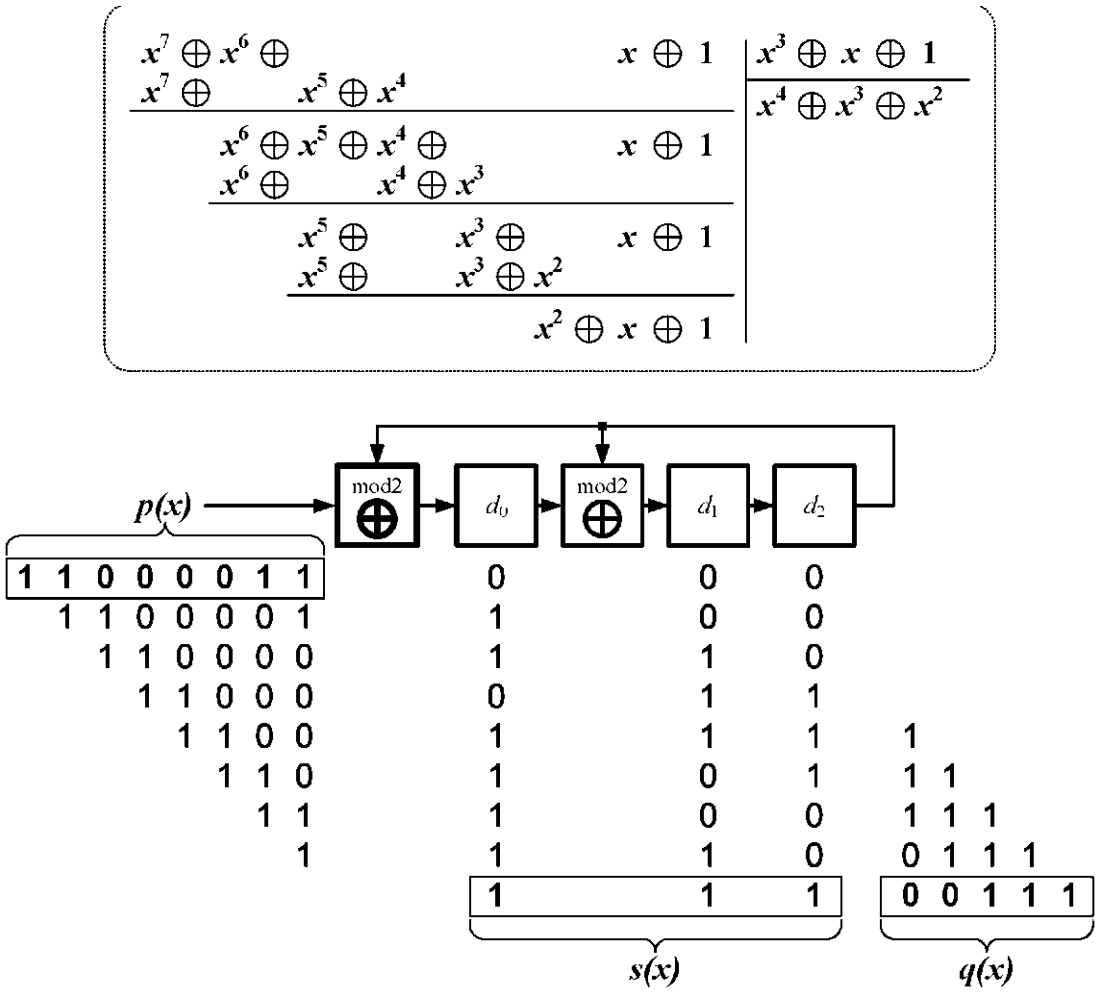



Рис. 2.40. Структура і приклад функціонування сигнатурного аналізатора для $\varphi(x) - 1 \ominus x \oplus x^3$ та $p(x) - 1 \oplus x \ominus x^6 \ominus x^7$

При цьому значення сигнатури стискуваної послідовності буде відображуватися на вихідний шині Pout.

Існує велика різноманітність як схемних реалізацій генераторів ПВП, так і сигнатурних аналізаторів. Наприклад, у тестуванні і діагностиці цифрових пристроїв широке застосування отримали багатоканальні сигнатурні аналізатори та генератори ПВП, побудовані на основі розглянутих схем.

## 2.6. Синхронні кінцеві автомати

Особливим класом послідовних пристроїв є клас *цифрових кінцевих автоматів* (FSM, Finite State Machine), найчастіше використовувані для реалізації цифрових пристроїв управління, проте з формальної точки зору довільна послідовна схема є цифровим автоматом. Самі кінцеві автомати управління класифікуються як два підкласи: автомати з фіксованим і мікропрограмним управлінням [65, 66].

Під автоматами з фіксованим управлінням будемо розуміти такі автомати, функціонування яких визначається на етапі їх проектування і не змінюється під час експлуатації.

У свою чергу автомати з мікропрограмним управлінням є більш складними пристроями, поведінка яких може динамічно змінюватися в процесі їх використання.

У цьому підрозділі розглянемо перший найпростіший тип автоматів. Через свої особливості автомати з мікропрограмним управлінням будуть розглянуті далі.

На відміну від раніше поданої узагальненої структури послідовного пристрою (рис. 2.16), структуру кінцевого автомата можна деталізувати виходячи з визначення поточного і наступного стану.

Під поточним станом автомата будемо розуміти багатовимірний двійковий вектор S, значення розрядів якого визначаються рівнями сигналів, що зберігаються на j найпростіших елементах пам'яті, сукупність яких є пам'яттю автомата. Через те, що кількість найпростіших елементів пам'яті має свою межу, кінцевим є і весь простір Q можливих станів, потужність якого оцінюється як $2^j$.

З простору можливих станів Q зазвичай виділяють підпростір допустимих станів $F \subseteq Q$, якому, зокрема, належить поточний стан автомата $S \in F$.

Значення поточного стану автомата може змінитися на новий стан $S^* \neq S,\ S^* \in F$ з настанням деякої зовнішньої події. У більшості випадків такою подією є фронт сигналу синхронізації і/або зміна рівня вхідного інформаційного сигналу. У посібнику будемо розглядати тільки випадок синхронних кінцевих автоматів, для яких обов'язковою умовою зміни поточного стану

є наявність фронту сигналу синхронізації. У зв'язку з цим пам'ять автомата може бути подана як множина найпростіших синхронних елементів пам'яті, наприклад D-тригерами.

Значення нового стану автомата $S^*$ визначається виходячи з поточного стану S і/або вхідного набору інформаційних сигналів $IP$ і може бути реалізовано як комбінаційна схема перемикача функції $\gamma: S^* = \gamma(S, IP)$.

У свою чергу автомат виробляє керуючі сигнали, об'єднані в єдиний двійковий результуючий вектор OP, значення якого також може визначатися поточним станом або значенням вхідного набору: $OP = \phi(S, IP)$.

Цифровий автомат має починати своє функціонування перебуваючи в певному стартовому стані $S_0 \in F$, який встановлюється при ініціалізації пам'яті автомата. Таким чином, узагальнена структура кінцевого цифрового автомата може бути такою, як на рис. 2.41.

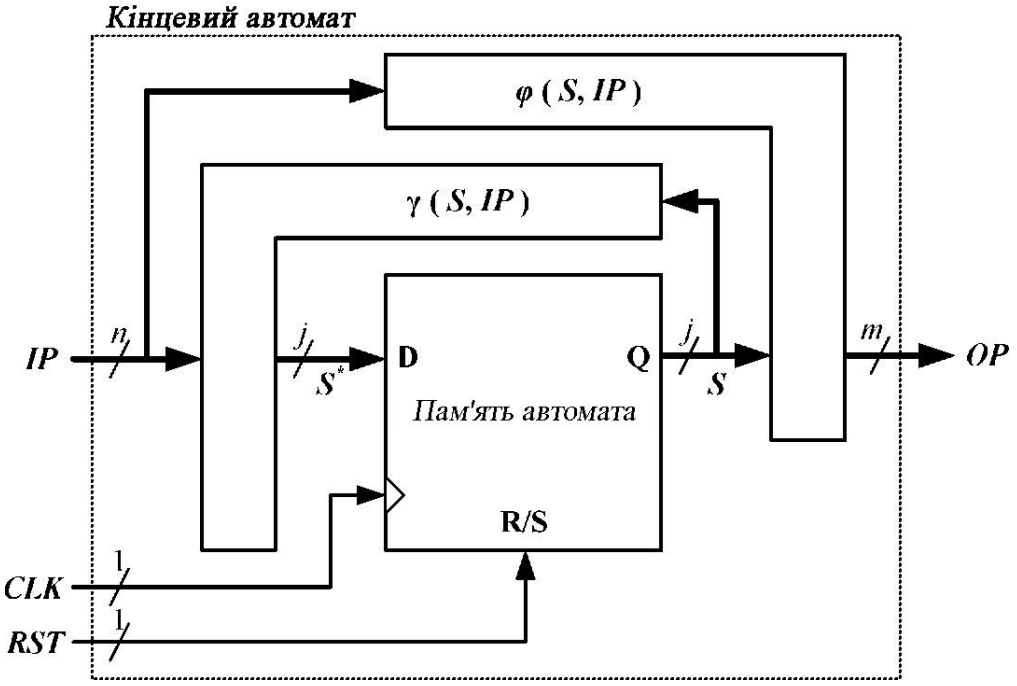



Рис. 2.41. Узагальнена структура кінцевого автомата

Залежно від типу кінцевого автомата комбінаційні схеми реалізації функцій γ і ϕ можуть бути різними або зовсім відсутні [67]. Наприклад, кінцевий автомат Мура виробляє вихідні значення тільки на основі поточного стану $OP = \phi(S)$. Для

автомата Мілі $OP = \phi(S, IP)$, а для автомата Медведєва $OP = S$ [68]. Для породжуваних автоматів і зовсім може бути відсутня вхідна шина IP, при цьому $S^* = \gamma(S)$.

Одним з наочних способів опису поведінки кінцевих автоматів є граф переходів. Багато САПР цифрових пристроїв мають ув своєму складі спеціалізовані графічні редактори, що дозволяють проектувати цифрові автомати за допомогою графів переходів.

У загальному випадку граф переходів являє собою кінцевий спрямований граф, множина вершин якого визначається множиною станів автомата F. Перехід від однієї вершини до іншої відображується спрямованою дугою, яку супроводжує опис умови переходу. Якщо умови не вказано, отже цей перехід є безумовним, виконання якого здійснюється при настанні фронту сигналу синхронізації.

Для вершин графа переходів можуть бути позначені дії, виконувані під час знаходження автомата у відповідних станах.

Стартовий стан автомата має додаткову умову, при якій здійсниться асинхронний перехід незалежно від значення поточного стану. Якщо з вершини графа переходів не виходить жодна дуга, то така вершина визначає кінцевий (тупиковий) стан автомата. Після досягнення кінцевого стану автомат припиняє своє функціонування. Відновлення роботи автомата можливе тільки при настанні умови ініціалізації, при якій автомат перейде в стартовий стан (рис. 2.42).

Проектування VHDL-опису кінцевого автомата може бути виконано з урахуванням поданої узагальненої структури (рис. 2.41) і графа переходів. З точки зору поведінкового стилю автомат може бути описаний кількома процесами, один з яких являє собою синхронну пам'ять автомата, а решта – комбінаційні схеми перемикальних функцій $\gamma$ та $\phi$.

Розглянемо приклад поведінкового VHDL-опису автомата (дод. 1, лістинг 2.16), функціонування якого задано представленим графом переходів (рис. 2.42).

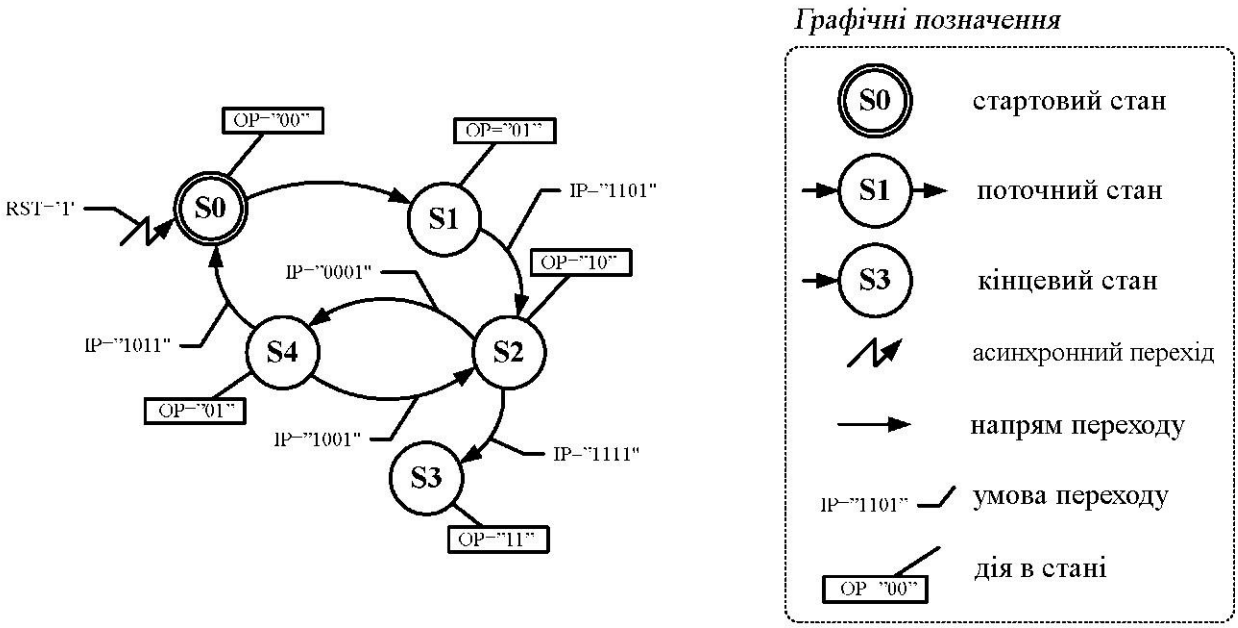



Рис. 2.42. Приклад графа станів кінцевого автомата

Наведений опис містить три процеси, один з яких FSM_mem описує поведінку синхронної пам'яті автомата. З точки зору результату синтезу можна припускати, що пам'ять автомата буде являти собою набір з D-тригерів (технологічних примітивів fdc), кількість яких залежить від кількості реалізованих станів. Однак число тригерів може залежати не тільки від потужності j множини перечислювального типу states. Оптимальна кількість використовуваних тригерів може бути оцінена як $[log_2 j]$, що для наведеного прикладу становить три тригери (j = 5). Тип, що перераховуються, є зручним інструментом не тільки при складанні проектних описів кінцевих автоматів, а й для систем функціонального моделювання, які відображують пойменовані значення станів на часових діаграмах. У той же час такий підхід при описі пам'яті автоматів не дозволяє кодувати стани автомата певними двійковими значеннями. У багатьох системах автоматизованого проектування передбачена можливість визначати вид кодування станів за допомогою налагоджувань логічного синтезатора. Наприклад, для САПР Xilinx ISE в пункті меню Synthesize::Prosess Properties::HDL Options::FSM Encoding Algorithm можна вибрати один з дев'яти передбачених способів автоматичного кодування станів автомата. Серед пропонованих алгоритмів кодування станів є можливість здійснювати

символами послідовностей One-Hot, Джонсона і Грея або вказати синтезатору на застосування оптимального способу кодування (Compact).

Для контролю за способом кодування станів автомата на рівні вихідного VHDL опису можна застосовувати таку методику, яка полягає у використанні констант типу std_logic_vector.

```
    – – . . .
    – – оголошення єдиного користувальницького підтипу
    – – для подальшого визначення імен і сигналів станів
    subtype tstate is std_logic_vector (2 downto 0);
    – – оголошення множини станів автомата
    – – із зазначенням імен станів і їх довічних еквівалентів
    constant S0 : tstate := "000";
    constant S1 : tstate := "001";
    constant S2 : tstate := "010";
    constant S3 : tstate := "100"; constant S4 : tstate := "111";
    – – сигнали поточного і наступного стану автомата
    signal current_state , next_state : tstate ;
    – – . . .
    next_state <= S3;
    – – . . .
```

Цей підхід дозволяє кожному стану задати його двійковий еквівалент, що може бути необхідним, наприклад, у випадках проектування автоматів з мінімальною перемикальною активністю [69].

Слід враховувати, що процеси, що описують комбінаційні схеми вироблення наступного стану та вихідні набори автомата, не повинні містити оператори, що призводять до синтезу елементів пам'яті. Наприклад, у наведеному лістингу в разі використання оператора null при визначенні наступного стану автомата для поточного стану S3 (рядок 54 лістингу 2.16):

```
    – – . . .
    when S3 => null ;
    – – . . .
```

результат синтезу буде містити додаткові елементи пам'яті, використані для реалізації значень сигналу next_state.

Також для запобігання небажаних колізій при проведенні функціонального моделювання поведінкових описів кінцевих автоматів, у тому числі і синтезу додаткових елементів пам'яті в операторах case, бажано вказувати дію зі зміни перемикальних функцій при всіх інших умовах **(when others** => ...).

## 2.7. Проектування запам’ятовуючих пристроїв

Крім регістрів зберігання і регістрових файлів, при проектуванні вбудованих систем часто виникає необхідність у реалізації запам'ятовуючих пристроїв (ЗП) великої інформаційної ємності. Серед усього різноманіття цифрових ЗП найчастіше реалізуються оперативні ЗП (ОЗП), що використовуються при проектуванні **мікропроцесорних** і **мікроконтролерних** систем. Типовий ОЗП нагадує двовимірну структуру найпростіших запам'ятовуючих елементів, адресація яких здійснюється відповідно до логічної організації даних, що зберігаються. Функціональна структура ОЗП наведена на рис. 2.43 і містить двовимірний масив запам'ятовуючих елементів, регістр адреси, регістр даних, дешифратори рядків і стовпців і схему управління читанням/записом.

ОЗП містить масив з $2^m \times n$ запам'ятовуючих елементів, кожен з яких здатен зберігати один інформаційний біт. Операції запису і читання виконуються над логічними структурами даних, так звані слова, що мають розмірність n бітів. Таким чином, ОЗП здатен зберігати $2^m \times n$ розрядних слів. Доступ до слів, які зберігаються, здійснюється за допомогою *m*-розрядної адреси. Операції читання і запису слів можуть здійснюватися в довільному порядку. Через це ОЗП ще називають пам'яттю з довільною вибіркою. При здійсненні операції читання спочатку на адресну шину AB подається *m*-розрядне значення адреси необхідного слова, що тимчасово зберігається в регістрі адреси. Залежно від логічної організації даних, що зберігаються, подане значення адреси розділяється на адресу рядка і адресу стовпця, що надходять на відповідні дешифратори. Запам'ятовуючі елементи, які стоять на перетині вибраного рядка і вибраних

стовпців, утворюють множину елементів, що зберігають слово, яке адресується, значення якого тимчасово зберігається в регістрі даних і стає доступним на вихідній шині даних DB. При здійсненні операції запису нове значення адресованого слова записується з вхідних портів шини DB в регістр даних, а потім до вибраної множини елементів пам'яті. Таким чином, ОЗП може функціонувати в трьох режимах: режим читання даних, режим запису даних і режим зберігання. За функціонування ОЗП у всіх режимах відповідає цифрова схема, що у свою чергу управляється відповідними зовнішніми сигналами, об'єднаними в шину управління СВ.

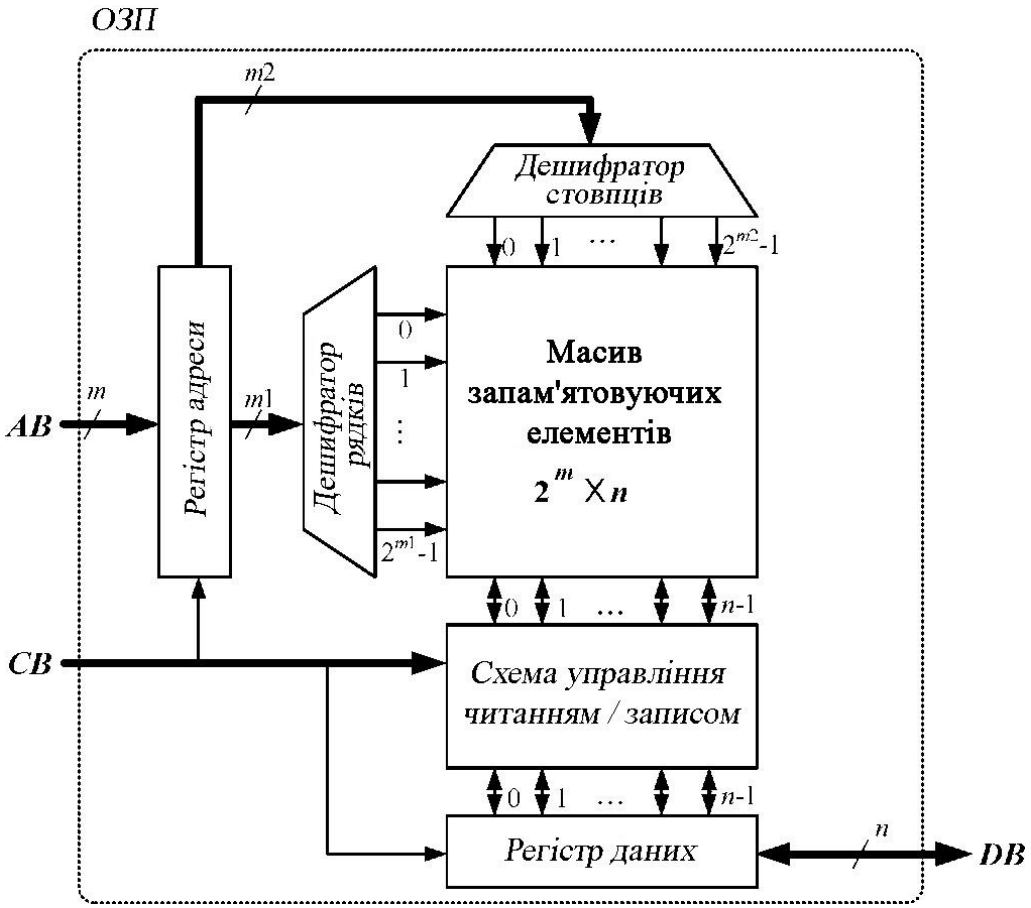



Рис. 2.43. Узагальнена структура ОЗП

Залежно від технології виготовлення ОЗП можуть мати додаткові внутрішні схеми управління. Наприклад, динамічні ОЗП (ДОЗП) мають на додачу пристрій управління регенерацією і регістр регенерації [70]. Крім того, ОЗП, як і регістрові файли, можуть мати кілька незалежних портів читання/запису, додаткові внутрішні схеми виявлення і/або корекції помилок тощо.

Через те, що масив запам'ятовуючих елементів ОЗП складається з транзисторних схем [70], не існує можливості скласти синтезовані поведінкові описи таких пристроїв. Для проектування відокремлених модулів ОЗП розробники використовують так звані компілятори пам'яті (Memory Compilers), що являють собою автоматизовані системи топологічного проектування інтегральних схем запам'ятовуючих пристроїв на основі обраної технологічної бібліотеки [71].

Мова VHDL не має лінгвістичних конструкцій, що дозволяють однозначно описувати цифрові модулі ОЗП. Однак мова VHDL дає можливість створювати функціональну абстракцію ОЗП за допомогою використання двовимірних користувальницьких сигналів синтезованих типів bit_vector або std_logic_vector. Для більшості засобів логічного синтезу складені функціональні моделі ОЗП будуть синтезовані в множину відповідних базових запам'ятовуючих елементів (тригерів або засувок) і цифрові блоки, необхідні для імітації роботи ОЗП.

Для технологій ПЛІС типу FPGA реалізація модулів ОЗП може бути здійснена одним з трьох способів:

- за допомогою вбудованих модулів ОЗП (BlockRAM);
- використання блоків LUT (реалізація так званої розподіленої пам'яті (Disrtibuted RAM));
- базових конфігурованих елементів пам'яті блоків CLB.

Для можливості технологічного синтезу VHDL-описів модулів ОЗП системи автоматизованого проектування пропонують розробникам ряд підходів, що полягають у складанні VHDL-конструкцій, які «*розпізнаються*». Так, наприклад, проектувальнику може бути надано шаблонний опис ОЗП, який він може модернізувати щодо заміни імен ідентифікаторів, розрядності шин даних і адреси. Якщо дотримуватися цих обмежень, то існує велика ймовірність успішного синтезу відповідного модуля ОЗП. В іншому випадку, коли проектувальник, наприклад, здійснить мультиплексування вхідної і вихідної шин даних або визначить операцію читання даних як асинхронну, результат синтезу буде непередбачуваним і часто неадекватним з точки зору очікувань проектувальника.

Інша методика полягає в автоматичному розпізнаванні засобом синтезу вихідного VHDL-опису щодо наявності моделі

ОЗП з подальшим використанням готових рішень для складання схемної реалізації. При цьому на розсуд проектувальнику віддається рішення щодо вибору остаточного вигляду схемної реалізації. Розглянемо приклад синтезу найпростішого модуля ОЗП, вихідний VHDL-опис якого подано в дод. 1 – лістинг 2.17.

За замовчуванням, результатом технологічного синтезу наведеного опису ОЗП будуть вісім тригерів типу fde (згідно з наведеним описом процесу WRP), сім блоків LUT для реалізації схем адресації тригерів і два тристабільних буфери для реалізації двобічної шини даних (процес RDP).

Для FPGA типу SPARTAN-3E є можливість використовувати LUT-блоки при реалізації синхронного ОЗП. Для отримання відповідного результату технологічного синтезу необхідно вказати засоби синтезу шляхом встановлення таких параметрів: Synthesize :: Properties :: HDL Options :: RAM Extraction в значення «ввімкнено» і RAM Style в значення «Distributed». Результатом технологічного синтезу при таких параметрах будуть два LUT-блоки в конфігурації синхронного ОЗП, подані як технологічні примітиви RAM16x 1S (рис. 2.44).

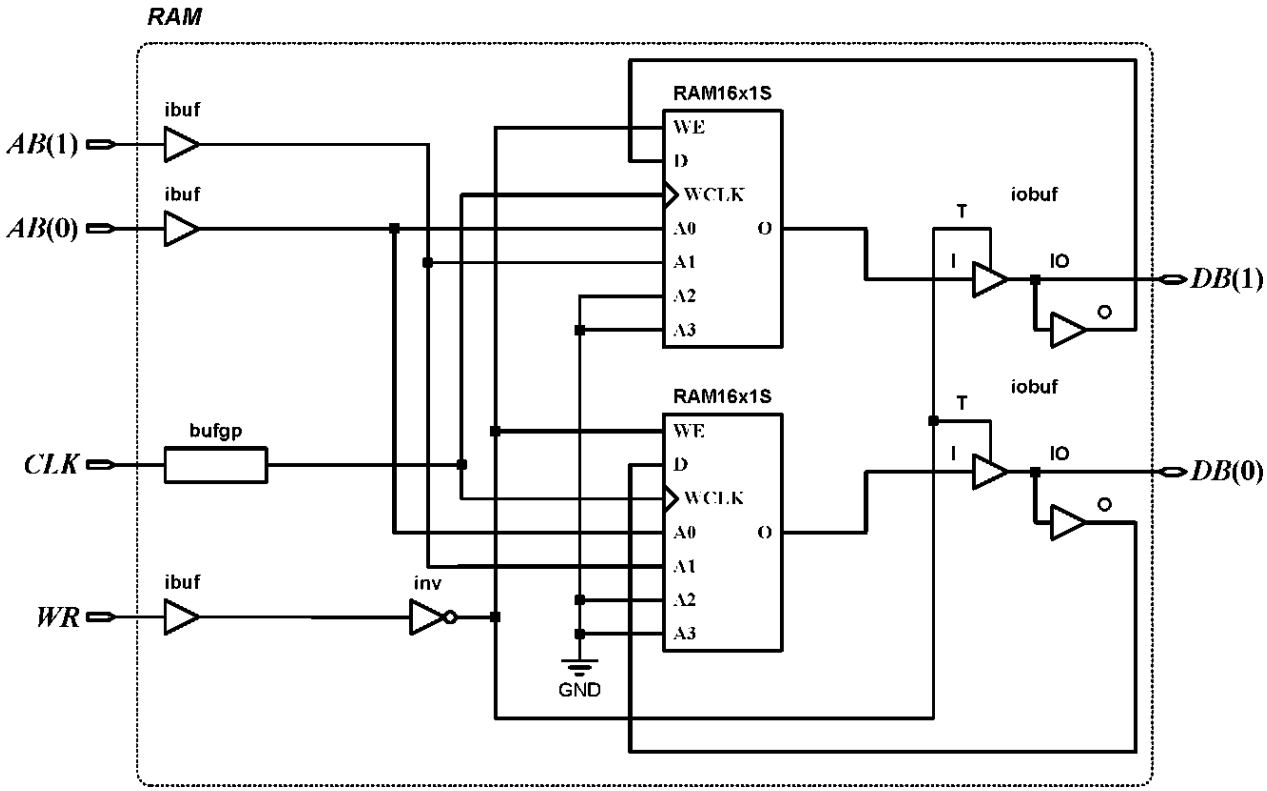



Рис. 2.44. Результат використання розподіленої пам'яті при реалізації ОЗП для FGPA

Один LUT-блок може бути використаний як ОЗП з максимальною конфігурацією збережених даних 16x1 біт (m = 4, n = 1). Запис даних відбувається при настанні переднього фронту сигналу синхронізації, що подається на вхідний порт WCLK за наявності низькорівневого сигналу на вході дозволу запису WE. Читання збережених даних здійснюється асинхронно шляхом подачі значення адреси на входи A0 - A3 при WE='1'.

У структурі FPGA SPARTAN-3E є вбудовані апаратні блоки конфігурованих двопортових ОЗП, щоі називаються Block RAM [55]. Максимальна кількість модулів Block RAM дорівнює 36 для кристалів XC3S1600E при загальному інформаційному обсязі 663552 біти. Кожен модуль Block RAM може бути налагоджений як однопортовий або двопортовий ОЗП, при цьому логічна організація збережуваних даних може варіюватися від 16Kx1 до 256x72. Для багатьох варіантів логічної організації передбачено використання вбудованого механізму перевірки на парність (parity). Наприклад, для організації збережених даних 2Kx9 для кожного однобайтового слова, що зберігається, передбачений додатковий біт коду перевірки на парність. Крім перерахованих особливостей, модулі Block RAM можуть бути використані для реалізації ПЗП з фіксованим контентом, значення якого вказується на стадії складання проектного опису. Для кожного варіанта логічної організації Block RAM існує відповідний технологічний примітив. Наприклад, примітив **RAMB16_S4** відповідає модулю Block RAM з конфігурацією 4Kx4.

Використання вбудованих модулів ОЗП може бути здійснено різними способами, у тому числі шляхом зазначення відповідних налагоджувань синтезатора для автоматичного розпізнавання користувальницьких описів запам'ятовуючих пристроїв з подальшою їх реалізацією на Block RAM. Для автоматичного розміщення результату технологічного синтезу даного опису (дод. 1, лістинг 2.17) необхідно встановити такий параметр синтезатора: Synthesize :: Properties :: HDL Options :: RAM Style в значення «Block». З урахуванням того що у вбудованих модулях Block RAM реалізована синхронна операція читання, слід скоригувати вихідний опис процесу RDP так:

```
- - . . .
- - процес, що описує синхронну операцію
читання
```

```
- - спільно з набором тристабільних буферів
RDP: process ( WR, CLK, sRAM, adrreg )
begin
if WR = '1 ' then
if rising_edge( CLK ) then
DB <= sRAM( adrreg );
end if ;
else
`DB <= ( others => 'Z' );
end i f ; end process ;
- - . . .
```

Результатом технологічного синтезу модифікованого опису буде схема, зображена на рис. 2.45.

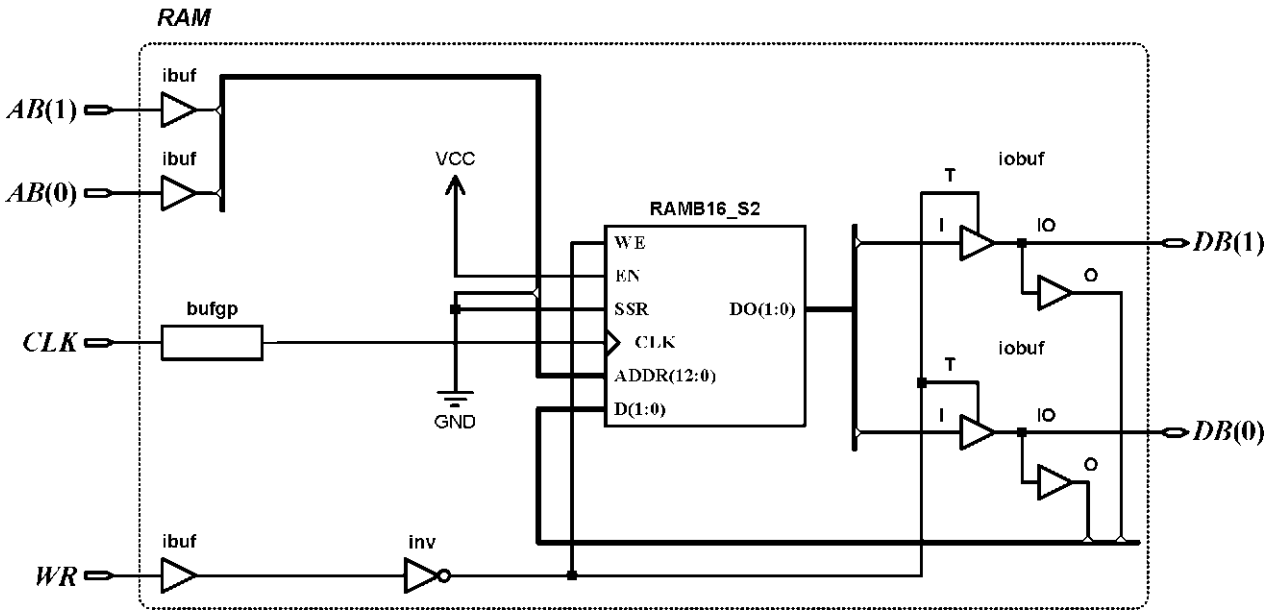



Рис. 2.45. Результат технологічного синтезу ОЗП з використанням вбудованого модуля Block RAM

Для реалізації описаного ОЗП був використаний модуль Block RAM в конфігурації 8Kx2, для якого 11 старших розрядів шини адреси *ADDR* примусово встановлені в нульове значення.

Крім вищеописаних методів складання проектних описів ОЗП існують альтернативні варіанти, що полягають, наприклад, у використанні готових IP-компонент запам'ятовуючих пристроїв. Так, для САПР Xilinx ISE існує можливість застосування

програмного засобу під назвою Core Generator [72], що дозволяє автоматизувати процес синтезу модулів ОЗП при складанні проектних описів цифрових систем.

## 2.8. Помилки проектних описів

Складання проектних описів цифрових систем за допомогою HDL-мов є автоматизованим процесом, у якому визначальну роль все ж таки відіграє розробник, незважаючи на наявність таких засобів, як Core Generator або Language Templates [73]. Некоректне використання методів і засобів автоматизованого проектування, у тому числі і неправильне складання проектних описів, може породжувати *помилки проектування*. Усі помилки проектування можна класифікувати на три види [74]: мовні помилки, помилки описів і приховані помилки. Розглянемо докладніше ці види помилок.

*Мовні помилки* є помилками вихідних HDL-описів і поділяються на *синтаксичні* та *семантичні*. Синтаксичні помилки часто виникають при некоректному складанні мовних конструкцій HDL-описів, неправильному записі операторів, неправильному наведенні типів використовуваних об'єктів тощо. Виявлення подібних помилок відбувається на перших стадіях синтезу при функціонуванні синтаксичного і лексичного аналізаторів відповідної HDL-мови. Усуваються синтаксичні помилки шляхом коригування вихідного опису за інформацією, що надається системами САПР. Семантичні помилки є більш замкненими і латентними. Як правило, вони не виявляються на стадії синтаксичного і лексичного аналізу, є «прозорими» для засобів логічного і фізичного синтезу, але можуть проявитися, наприклад, при функціонуванні готового пристрою або на стадії функціонального моделювання. Однією з найпоширеніших семантичних помилок є некоректне складання списку чутливих сигналів процесу при складанні поведінкового VHDL-опису. Розглянемо семантичну помилку на такому прикладі:

```
_ _ . . .
signal F, A, B : std_logic;
_ _ . . .
```

```
MAIN: process ( A ) — Помилка!
Begin
- - . . .
F <= A xor B ;
end process;
- - . . .
```

У списку чутливих сигналів процесу *Main* не вказано сигнал *B*. При функціональному моделюванні наведеного процесу значення сигналу *F* не змінюватиметься при фіксованому значенні сигналу *A* та змінюваному значенні сигналу *B*. У загальному випадку послідовність сигналів, що формуються на виході *F*, буде відрізнятися від послідовності, яка має вироблятися на виході логічного вентиля XOR2. У той же час результатом логічного синтезу об'єкта *F* буде двовходовий вентиль XOR2. Для усунення подібних помилок можна користуватися методикою, що полягає в перерахуванні імен всіх сигналів у списку чутливих сигналів, значення яких використовуються операторами конкретного процесу.

При усуненні всіх синтаксичних і семантичних помилок переходять до пошуку та усунення *помилок описів*, які поділяються на *структурні* і *логічні*. Під структурними помилками розуміють некоректний опис карт міжз'єднань цифрових компонент за допомогою операторів **port map**, а під логічними помилками – використання мовних операторів і компонент, відмінних від очікуваних. Наприклад, до логічних помилок можна віднести некоректне використання послідовного оператора if мови VHDL, що найчастіше призводить до синтезу небажаних елементів пам'яті. Виявлення таких помилок ускладнено через їх маскування засобами функціонального моделювання. Їх виявлення можливе за допомогою детального вивчення результатів RTL- і технологічного синтезу.

У разі усунення помилок опису залишається ймовірність наявності *прихованих помилок*, до яких найчастіше відносять *помилки синхронізації*, що здатні порушити логіку функціонування готового цифрового пристрою, а також можуть призводити до реалізації потенційно небезпечних схем, здатних виробляти небажані («паразитні») імпульси [75, 110].

Типовою помилкою синхронізації можна вважати опис і реалізацію так званих комбінаційних сигналів синхронізації, що можуть призвести до перекосу фази синхросигналу і викликати некоректне функціонування готового цифрового пристрою. Розглянемо опис такого комбінаційного сигналу синхронізації на такому прикладі:

```
- - . . .
Port — ...
— порт сигналу синхронізації
C : in std_logic;
— порт сигналу дозволу синхронізації
EN : in std_logic;
- - . . .
signal xC, D, Q : std_logic;
- - . . .
xC <= C and EN ;
MAIN: process ( xC, D )
Begin
- - . . .
if rising_edge( xC ) then
Q <= D ;
end if ;
end process;
- - . . .
```

Як видно, сигнал xC є вихідним сигналом комбінаційної схеми, що використовується в процесі MAIN як сигнал синхронізації. При реалізації цифрових пристроїв на ПЛІС проблема використання комбінаційних сигналів синхронізації посилюється непередбачуваністю затримки їх поширення після процесів розміщення і трасування на програмованих ресурсах міжз'єднань. Для вирішення подібних проблем бажано використовувати загальний сигнал синхронізації для всіх послідовних схем, побудованих на основі базових тригерів з вхідним портом управління CE (Clock Enable), значення сигналів для якого можуть виробляти комбінаційні схеми.

Знаходження й усунення помилок проектних описів здійснюється на всіх стадіях проектування цифрових пристроїв: від початкового функціонального моделювання до стендового

випробування першого прототипу. Даний процес є ітераційним і евристичним через наявність багатьох чинників, одним з яких є людський чинник проектувальника. Незважаючи на це, різні компанії-виробники САПР і технологій виготовлення цифрових пристроїв намагаються давати рекомендації щодо усунення та мінімізації кількості помилок проектних описів [76].

**Контрольні питання**

1. Чим відрізняється процесор від мікропроцесора?
2. Що таке мікропроцесорний комплект?
3. Наведіть основні характеристики мікропроцесорів.
4. Наведіть класифікаційні ознаки мікропроцесорів.
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.
8. Що являє собою принцип ЗМ при побудові мікропроцесорних систем?
9. Принцип модульної організації МПС.
10. Магістральний спосіб обміну інформацією в МПС.
11. Мікропрограмна реалізація управління в МПС.
12. Що являє собою принцип агрегатування?
13. Організація одно- і багатомагістральних мікропроцесорних систем.
14. Типова структура тришинної МПС.
15. Призначення схеми синхронізації та початкового скидання.
16. Організація доступу до пам'яті МПС.
17. Організація доступу до пристроїв введення/виведення.
18. Призначення таймера в МПС.
19. Організація прямого доступу до пам'яті в МПС.
20. Призначення контролера переривань у МПС.

# 3. ПРОЕКТУВАННЯ ВБУДОВАНИХ ПРИСТРОЇВ, СИСТЕМ І ПРОГРАМНИХ ЗАСОБІВ З МІКРОПРОГРАМНИМ УПРАВЛІННЯМ

## 3.1. Цифрові пристрої з мікропрограмним управлінням

Цифрові пристрої з *мікропрограмним управлінням* (МПУ) являють собою пристрої, що виробляють послідовності керуючих сигналів, необхідних для досягнення конкретного результату шляхом реалізації мікропрограми. На відміну від цифрових кінцевих автоматів з жорсткою логікою функціонування, автомати з МПУ можуть змінювати своє функціонування при незмінній апаратній структурі. Існує безліч варіантів реалізації пристроїв з МПУ [1, 54, 65, 66, 77], наприклад метод проектування на основі ПЗП, для якого мікропрограма являє собою набір інструкцій, що зберігаються в ПЗП, при цьому кожна інструкція включає адресну (значення адреси наступної інструкції) і операційну (значення керуючих сигналів) частини. Зі зміною вмісту ПЗП є можливість реалізувати довільну послідовність інструкцій і керуючих сигналів. Обмеженням для такого методу реалізації пристроїв з МПУ є інформаційна ємність і організація ПЗП. У цьому розділі будемо розглядати один з варіантів реалізації універсального пристрою з МПК, схожого з варіантами реалізації мікроконтролерних ядер з фіксованим набором інструкцій і лінійною пам'яттю мікропрограм.

Під мікропрограмою будемо розуміти сукупність керуючих інструкцій, певна послідовність яких є відображенням алгоритму управління. У свою чергу *інструкція* являє собою двійкове слово, значення якого визначає функціонування пристрою управління.

В основі пристроїв з МПУ лежать цифрові кінцеві автомати, що виробляють керуючі сигнали залежно від свого поточного стану і значення поточної інструкції, що виконується. Інструкції зазвичай зберігаються на зовнішніх запам'ятовуючих пристроях, з яких вони послідовно витягуються. Таким чином, узагальнену структуру пристрою з МПУ можна подати як сукупність таких основних блоків: пам'ять мікропрограми, цифровий кінцевий автомат і безліч підпорядкованих (керованих) блоків (рис. 3.1) [1].

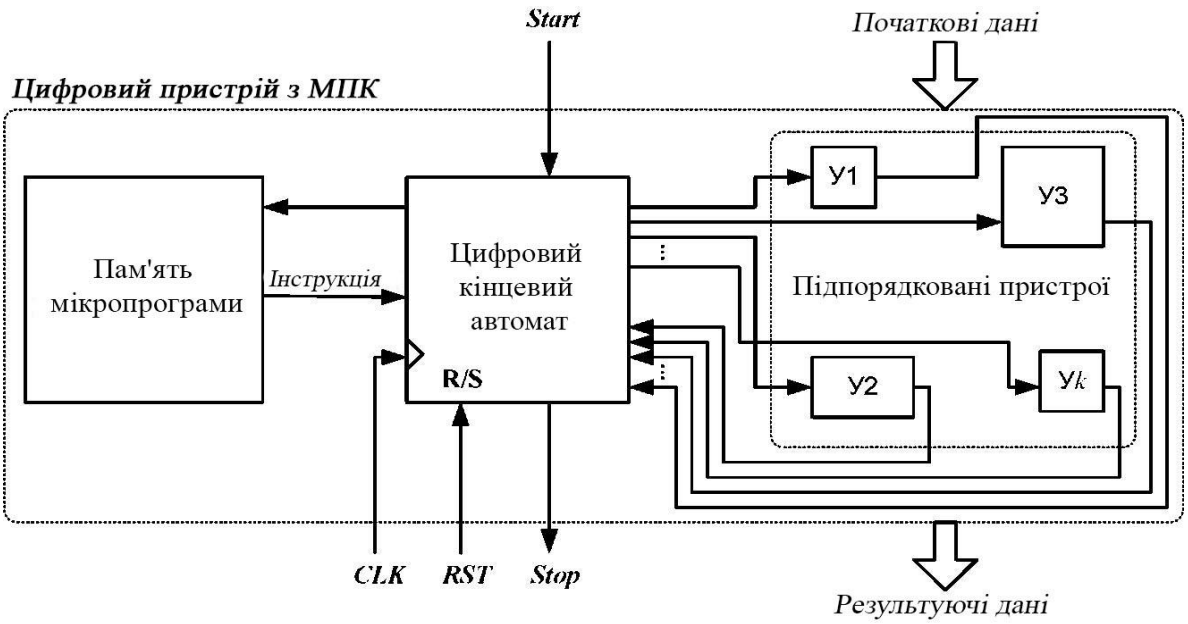



Рис. 3.1. Узагальнена структура пристрою з МПУ

Сукупність підпорядкованих блоків утворює тракт обробки вихідних даних (datapath) з метою отримання результуючих даних відповідно до алгоритму, поданого у вигляді мікропрограми. Початок процесу обробки вихідних даних задається сигналом *Start*, який ініціює кінцевий автомат для вироблення відповідних керуючих сигналів. Закінчення процесу обробки сигналізується виробленням автоматом сигналу *Stop*, значення якого може бути використано сторонніми пристроями.

Функціонування цифрового кінцевого автомата, що виконує роль центрального пристрою управління, залежить від початкових даних, що надходять, сигналів зворотних зв'язків підпорядкованих пристроїв і значень інструкцій. Таким чином, функціонування цифрового пристрою з МПУ може бути скориговано або кардинально змінено шляхом зміни інструкцій мікропрограми при фіксованій апаратурі пристрою управління і тракту оброблюваних даних. Ця функціональна перевага знайшла широке застосування в обчислювальних цифрових пристроях, таких як мікропроцесори та мікроконтролери.

Завдання проектування цифрового пристрою з МПУ може бути інтерпретовано як завдання реалізації певного алгоритму обробки даних за допомогою цифрових пристроїв з урахуванням відповідних обмежень. Істотними обмеженнями є часові і

апаратурні ресурси, необхідні для реалізації обраного алгоритму з урахуванням специфіки потоку початкових даних і конкретної технології виготовлення цифрового пристрою. Для успішного синтезу пристрою з МПУ необхідно враховувати всі наявні обмеження і виходячи з цього вибирати подальшу стратегію проектування. Істотним внеском до ефективності реалізації алгоритму є синтез підпорядкованих пристроїв тракту обробки даних. Так, можливість паралельної обробки даних, використання спеціалізованих блоків апаратних прискорювачів дозволить істотно спростити микропрограмму і реалізувати її за менший проміжок часу. За іншої стратегії, яка, наприклад, націлена на оптимізацію кількості підпорядкованих пристроїв, основним завданням є спільне проектування пристрою управління і керуючої мікропрограми, які в сукупності мають ефективно реалізувати алгоритм у наявних обмеженнях [78].

У загальному випадку завдання проектування цифрового пристрою з МПУ можна подати як три підзавдання:

1) синтез підпорядкованих пристроїв тракту обробки даних;

2) проектування формату інструкцій і мікропрограми управління;

3) синтез кінцевого автомата пристрою управління.

У свою чергу завдання синтезу тракту обробки даних включає такі підзавдання:

1) визначення обмежень часових і апаратних ресурсів;

2) проектування графа обчислювального процесу;

3) оптимізація графа обчислювального процесу, що передбачає планування і об'єднання операцій, вирішення завдань синхронізації тощо.

Для успішного управління наявними трактом обробки даних необхідна наявність відповідної керуючої мікропрограми, ефективність виконання якої визначається вирішенням таких завдань:

1) проектування необхідних інструкцій, визначення їхніх форматів і способів кодування;

2) складання керуючої мікропрограми та її оптимізація з урахуванням наявних обмежень і набору інструкцій;

3) синтез запам'ятовуючого пристрою, необхідного для зберігання мікропрограми.

Синтез кінцевого автомата пристрою управління базується на технічних особливостях тракту обробки даних, запам'ятовуючого пристрою мікропрограми і набору керуючих інструкцій.

Перераховані вище завдання можуть вирішуватися в довільному порядку, при цьому результати, отримані на одній стадії проектування, можуть вплинути на хід вирішення інших завдань. Подібна методика перехресного проектування носить назву **co-design,** яка означає спільне проектування як апаратної, так і програмної складової цифрового пристрою [79]. Архітектурні та функціональні зміни однієї складової завжди тягнуть за собою зміни іншої складової в рамках обумовлених обмежень з метою вирішення цільового завдання проектування.

Виклад перерахованих вище завдань будемо розглядати на такому прикладі. Нехай необхідно синтезувати цифровий пристрій, який обчислює значення алгебраїчного виразу

$$F = a \cdot x^2 + b \cdot x + c. \qquad (3.1)$$

де $a$, $b$, $c$, $x$ є додатними цілими числами, значення яких належать [1, 10-15].

Для обчислення наведеного виразу знадобляться такі функціональні пристрої, як двійковий цілочисельний помножувач і двійковий цілочисельний суматор. З урахуванням того, що вихідні аргументи $a$, $b$, $c$, $x$ можуть бути закодовані чотирма чотирирозрядними двійковими векторами, результат обчислення поданого виразу F вимагає 12 двійкових розрядів, яких буде достатньо для кодування довільного числа з діапазону [1, 10, 36, 15].

Синтез спеціалізованого цифрового пристрою, що обчислює значення виразу (3.1), може бути нераціональним як з точки зору кількості вхідних і вихідних портів сигналів оброблюваних і результуючих даних, так і з точки зору кількості необхідних функціональних блоків. Визначення способу зберігання і послідовної подачі вихідних даних є одним із першочергових завдань при проектуванні цифрових пристроїв з мікропрограмним управлінням.

## 3.2. Проектування тракту обробки даних

### 3.2.1. Визначення обмежень часових і апаратних ресурсів

Функціонування пристрою з МПУ починається за зовнішньою подією, наприклад за установленням високого рівня сигналу на вхідному порту Start (рис. 3.1), після якого цифровий автомат виконує всі необхідні дії з обробки вихідних даних для отримання необхідного результату. Після закінчення всіх дій автомат виробляє сигнал на вихідному порті *Stop*, значення якого сигналізує про готовність результуючих даних. Час, що минув від моменту установлення сигналу *Start* до моменту закінчення процесу обробки, назвемо *операційним часом* і позначимо його як $T = t_{Stop} — t_{Start}$, де $t_{Stop}$ є моментом часу закінчення останньої операції, $t_{Start}$ — моментом часу початку першої операції. Одним з обмежень при проектуванні пристрою з МПУ може бути вимога щодо виконання всієї процедури обробки даних за певний інтервал часу T*. При цьому пристрій має бути спроектовано таким чином, щоб виконувалася нерівність $T < T^*$.

Вихідні обмеження апаратних ресурсів можуть бути відображені в кількості зовнішніх портів тракту обробки даних, способі надходження вихідних даних для обробки, кількостіі і типі підпорядкованих пристроїв тракту тощо. Апаратні обмеження можуть істотно вплинути на значення T. Якщо умова $T < T^*$ є несуттєвою або нею можна знехтувати, то вибирається стратегія проектування тракту обробки даних з меншою кількістю підпорядкованих пристроїв з метою скорочення апаратурних ресурсів. В іншому випадку тракт проектується таким чином, що апаратні обмеження стають несуттєвими, що призводить до використання додаткових підпорядкованих пристроїв або їх дублювання.

Для попереднього оцінювання часових і апаратних ресурсів проектованого тракту складають граф обчислювального процесу, який дає наочне уявлення про послідовність виконуваних операцій при обробці вихідних даних [80].

### 3.2.2. Проектування графа обчислювального процесу

Алгоритм обробки вихідних даних можна подати як спрямований граф, вершинами якого є відповідні алгебраїчні чи

інші операції. Дуги графа визначають надходження вихідних, проміжних і результуючих даних для кожної операції. Для ілюстрації паралельності виконуваних операцій відповідні вершини графа розташовуються на одному горизонтальному рівні, при цьому «проходження» графа зверху вниз відображує весь обчислювальний процес у цілому. Кількість вершин і рівнів графа визначається не тільки використаними операціями, а й залежністю операцій від проміжних результатів і можливості паралельного виконання операцій. Введемо такі позначення: нехай множина всіх операцій, необхідних для обчислення виразу, позначається $V = \{v_1, v_2, ..., v_k\}$, де $v_i$ ($i$ є {0, k}) – базова операція, k – потужність множини V, і визначає загальну кількість операцій. Для розглянутого прикладу k = 5 і $V = \{v_i, v_2, v_3, v_4, v_5\}$. Всі використані операції можуть належати різним класам і мати свої пріоритети. Наприклад, використані операції належать двом класам $v_i$ е {П, Σ}, де П – операція множення, Σ – операція додавання. Таким чином, для обчислення виразу (3.1) необхідна така множина операцій двох класів: $V = \{v_1, v_2, v_3, v_4, v_5\} = \{П, П, Σ, П, Σ\}$. З урахуванням того, що операція П має найвищий пріоритет, і з урахуванням можливості одночасного виконання різних операцій граф обчислювального процесу може бути зображений так, як на рис. 3.2 [1].

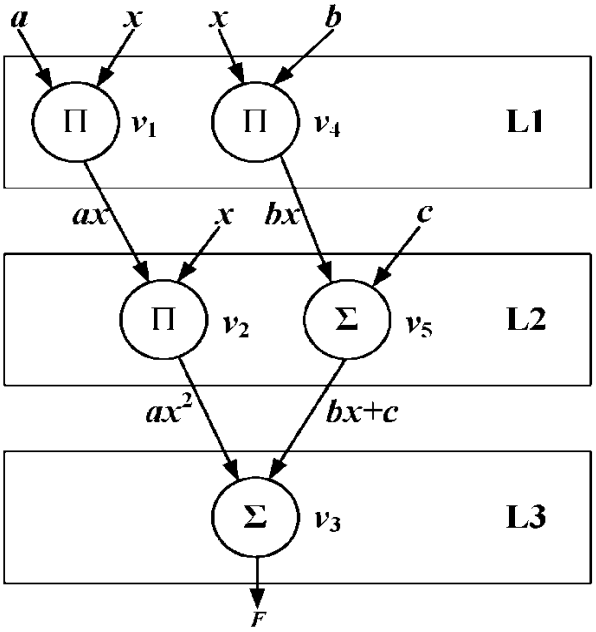



Рис. 3.2. Приклад графа обчислювального процесу

П'ять вершин графа розташовані на трьох рівнях L1, L2 і L3. Одночасність виконання операцій П на рівнях L1 і L2 обумовлена їхніми пріоритетами і незалежністю вихідних даних. Наприклад, операція Σ рівня L3 не може бути виконана на інших рівнях через залежності її вихідних даних від результатів виконання операцій з великим пріоритетом. При цьому множину V можна подати як множину рівнів таким чином: V = {L1,L2,L3}, де L1 = {$v_1$,$v_4$}, L2 = {$v_2$,$v_5$}, L1 = {va}.

Кількість рівнів графа визначається кількістю використовуваних операцій і оцінюється як $[\log_2 k] \le N_L \le k$ в разі вироблення одного значення за допомогою $k$ операцій. Для розглянутого прикладу кількість рівнів $N_l = 3$ є мінімальним при застосуванні $k = 5$ операцій. Таким чином, шуканий вираз може бути подано як обчислювальний граф з не менш ніж трьома рівнями. Послідовність рівнів визначає і послідовність виконуваних операцій. Час виконання кожного рівня залежить від типу використовуваних у ньому операцій. Нехай операція П має час виконання, рівний $\Delta_П$, а операція $\Sigma = \Delta_\Sigma$. Тоді час виконання одного рівня оцінюється виразом $\Delta_{Lj} = \max(\Delta_{v1,\dots,}\Delta_{vm})$, де $Lj = \{v_1, \dots, v_m\}, j \in [1, N_L]$.

З урахуванням, що час виконання однієї операції $\Delta_{vi} \in \{\Delta_П, \Delta_\Sigma\}$, величина $\Delta_{Lj}$ може набувати одного з двох значень: $\Delta_П$ або $\Delta_\Sigma$. Для розглянутого випадку маємо $\Delta_{L1=}\Delta_П, \Delta_{L2} = \Delta_П$ та $\Delta_{L3} = \Delta_\Sigma$.

Загальний операційний час може бути оцінений виразом

$$T = \sum_{j-1}^{N_L} \Delta_{Lj} = \Delta_{L1} + \Delta_{L2} + \Delta_{L3} = 2\Delta_П + \Delta_\Sigma.$$

### 3.2.3. Зв'язування операцій

Розглянутий приклад графа обчислювального процесу може бути використаний для реалізації апаратури тракту обробки даних, що включає три блоки помножувачів і два блоки суматорів, входи і виходи яких об'єднані відповідно до реалізованих операцій на всіх рівнях Lj. Однак такий підхід при синтезі тракту загрожує зайвими апаратними витратами. Тільки один рівень з усіх є активним при виконанні відповідних операцій, всі інші операції вже ніколи не будуть

використовуватися або будуть використовуватися на наступних рівнях. З точки зору апаратних блоків це означає таке: при виконанні операцій першого рівня функціонуватимуть два помножувачі, які в подальшому не будуть використані, а акумулятор третього рівня буде використаний тільки один раз для отримання кінцевого результату. Для оптимізації кількості використаних операційних блоків можна застосувати процедуру *зв'язування* операцій, яка означає повторне використання одного і того самого блока при реалізації різних рівнів графа обчислювального процесу [80]. Зв'язування операцій відмічається на графі замкненими областями, що включають вершини однотипних операцій, які перебувають на різних рівнях. Наприклад, операції $v_1$ та $v_2$ можуть бути зв'язані і виконані на одному блоці помножувача, так само як і операції $v_5$ та $v_3$ можуть бути виконані на одному суматорі (рис. 3.3). Таким чином, для проектування тракту знадобляться два помножувачі і один акумулятор, які будуть повторно використані при реалізації різних рівнів процесу обробки даних [1].

Побудова графа обчислювального процесу також залежить від початкових обмежень використовуваного апаратного ресурсу тракту обробки даних. Припустимо, що для синтезу тракту є можливість використовувати один блок помножувача і один блок суматора, при цьому існує можливість їх одночасного функціонування. У цьому випадку граф обчислювального процесу перетвориться до вигляду, поданого на рис. 3.4.

Наявне обмеження збільшує кількість використовуваних рівнів до чотирьох, при цьому операційний час оцінюється як $T = 3\Delta_{\Pi} + \Delta_{\Sigma}$. У випадку ще більшого обмеження, наприклад використання тільки однієї операції на одному рівні, операційний час може зрости до значення $T = 3\Delta_{\Pi} + 2\Delta_{\Sigma}$.

Для достовірного функціонування операційних блоків одного рівня необхідно фіксування значень аргументів на їхніх вхідних портах, щоб уникнути здійснення некоректних даних.

Досягається це за рахунок використання регістрових схем, що тимчасово зберігають оброблювані дані. Регістрові схеми зображуються на границях усіх рівнів графа обчислювального процесу, а завдання управління такими регістрами є завданням синхронізації тракту оброблюваних даних. Зобразимо наявність

регістрів зберігання для графа з рис. 3.3, при цьому кожному регістру дамо номер із зазначенням даних, що зберігаються (вихідних, проміжних і результуючих) (рис. 3.5).

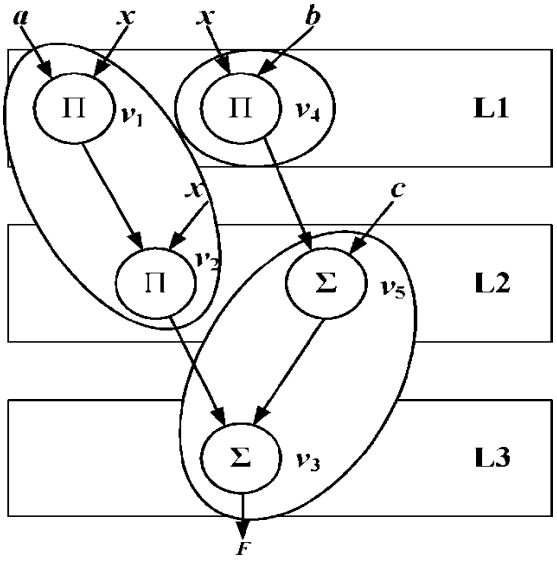



Рис. 3.3. Приклад зв'язування операцій

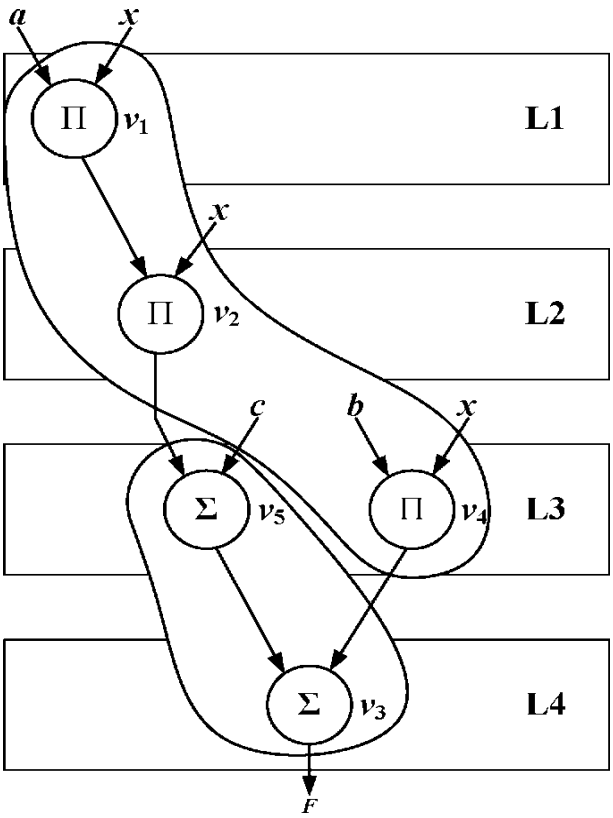



Рис. 3.4. Приклад графа обчислювального процесу при обмеженні апаратних ресурсів

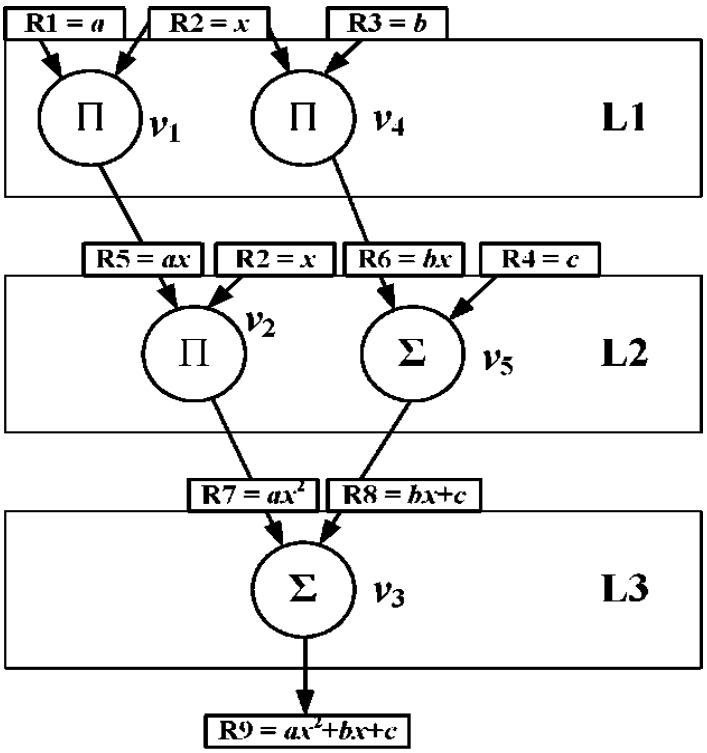



Рис. 3.5. Граф обчислювального процесу з зазначенням регістрів зберігання

Так, регістри ***R1, R2, R3*** та R4 зберігають вихідні дані для обробки, регістр R9 зберігає значення результату, а інші регістри – проміжні результати рівнів L1 і L2.

З урахуванням послідовного виконання операцій рівнів *Lj* кількість регістрів зберігання може бути мінімізовано шляхом їх повторного використання.

Наприклад, значення регістрів R1 і R3 після виконання операцій рівня L1 далі не використовуватимуться. У зв'язку з цим регістри R5 і R6 можуть бути замінені незастосовуваними регістрами R1 і R3.

З огляду на це граф обчислювального процесу може бути модифікований, як це показано на рис. 3.6.

Таким чином, кількість регістрів зберігання може бути скорочена з дев'яти до чотирьох. У загальному випадку організація і функціонування регістрів зберігання має істотний внесок до архітектури цифрового пристрою з МПУ.

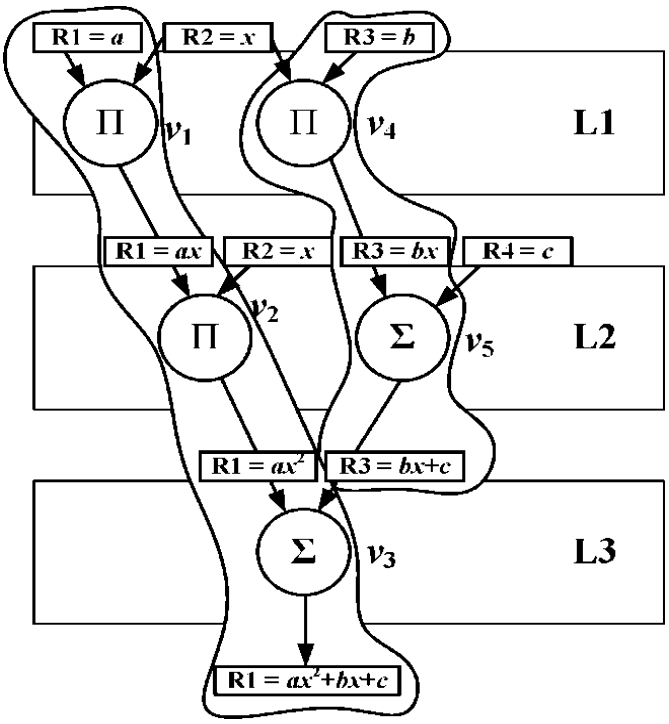



Рис. 3.6. Приклад повторного використання регістрів зберігання

## 3.3. Архітектури тракту оброблюваних даних

Як було показано, наявність регістрів у тракті оброблюваних даних необхідна для тимчасового зберігання проміжних результатів при переході від одного рівня графа обчислювального процесу до наступного. Один операційний блок вимагає наявності трьох регістрів зберігання: два регістри для зберігання вихідних даних (операндів) і один регістр для зберігання результату. Регістри операндів заповнюються оброблюваними даними, які надходять на вхід тракту обробки, і тільки кінцевий результат транслюється з регістра результату на вихід тракту. При вирішенні завдань оптимізації кількості регістрів зберігання регістри операндів і регістри результатів можуть поєднуватися і повторно використовуватися. У зв'язку з цим доцільним є доповнення тракту запам'ятовуючим пристроєм, у який попередньо записуються всі дані для подальшої обробки. Це дозволяє не тільки економити кількість інтерфейсних ліній для подачі оброблюваних даних, а й раціонально використовувати регістри. Доповнена структура тракту обробки даних подана на рис. 3.7 і містить ОЗП оброблюваних даних, блок операційних пристроїв і блок регістрів зберігання.

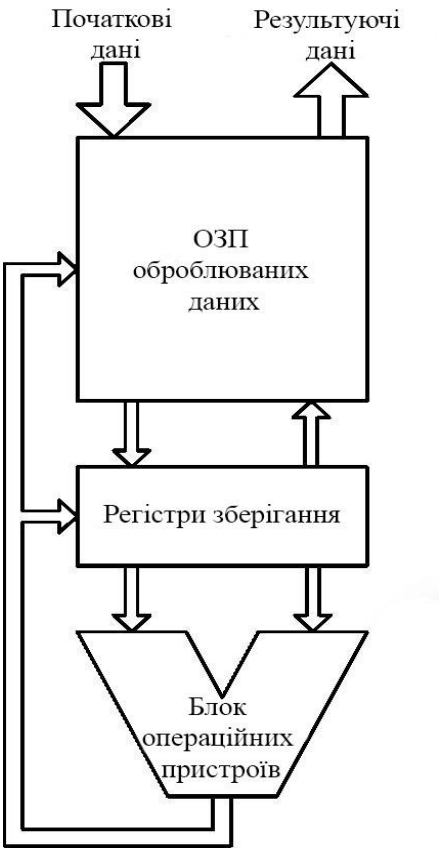



Рис. 3.7. Структура тракту обробки даних

Залежно від вирішуваних завдань регістри зберігання і ОЗП оброблюваних даних можуть бути суміщені. У загальному випадку принципи проектування подібних трактів обробки даних є архітектурними основами цифрових пристроїв з МПУ. Серед усього різноманіття архітектурних рішень з синтезу запам'ятовуючих пристроїв зберігання даних, що обробляються, можна виділити три основні [81]: стекова архітектура; акумуляторна архітектура; архітектура регістрів загального призначення.

Перераховані архітектури використовуються при проектуванні мікропроцесорних і мікроконтролерних обчислювальних пристроїв і можуть застосовуватися при синтезі трактів обробки даних для пристроїв з МПУ.

*Стекова архітектура розуміє наявність запам'ятовуючого пристрою з послідовним доступом типу* LIFO (Last In First Out), що виконує роль блока регістрів зберігання операндів і результатів. Оброблювані дані, попередньо записані в ОЗП, послідовно передаються в стекову пам'ять, звідки вони надходять на входи відповідного операційного блока. Після спрацьовування

операційного блока значення результату записується в стекову пам'ять. По закінченні обчислювального процесу кінцевий результат з стеку передається в ОЗП, з якого він може бути доступний стороннім пристроям.

Розглянемо приклад стекової архітектури, адаптованої для знаходження значення виразу (3.1). Припустимо, що ОЗП даних має організацію 8x8 біт і зберігає попередньо записані вихідні дані за такими адресами: 0 - a, 1 - b, 2 - c, 3 - x. Нехай кінцевий результат виразу f буде записаний за адресою 4. ОЗП даних є слово-орієнтованою пам'яттю з довільною вибіркою типу RAM. Блок операційних пристроїв містить один суматор і один помножувач, операнди для яких надходять з вершини стекової пам'яті *TOS* (Top Of the Stack) та елемента пам'яті, наступним за вершиною стека TOS+i. Результат, що виробляється операційним блоком, записується в TOS. Остаточний результат обчислень буде записаний з TOS у пам'ять даних за адресою 4. Структурf такого тракту у вигляді схеми, зображенf на рис. 3.8.

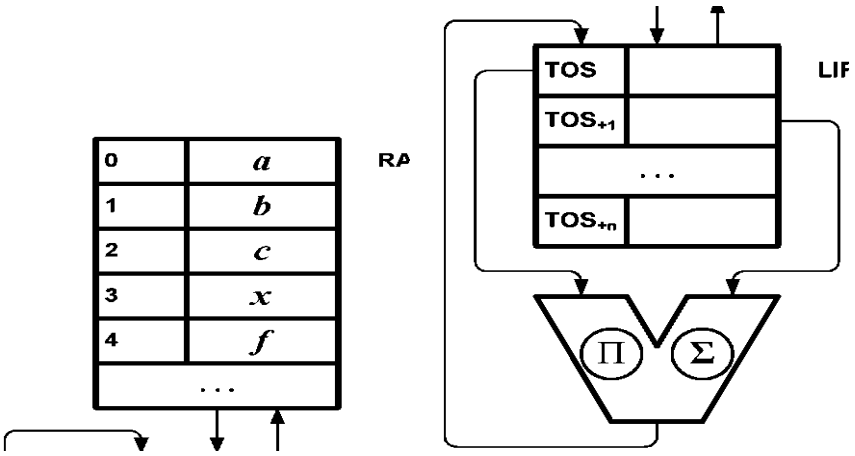



Рис. 3.8. Приклад стекової архітектури

Для обчислення значення виразу (3.1) з використанням поданої апаратури тракту обробки даних необхідно виконати послідовність дій, які можуть подаватися як послідовність певних мікрооперацій над пам'яттю даних, стеком, помножувачем і суматором. Наприклад, для отримання значення *a x x* необхідно виконати таку послідовність мікрооперацій:

1) зчитати з пам'яті даних за адресою 0 значення *a* і записати його на вершину стека TOS;

2) зчитати з пам'яті даних за адресою 3 значення *x* і записати його на вершину стеку TOS, при цьому значення *a* буде доступно в стеці за адресою TOS+1;

3) провести операцію множення, при якій на входи блока помножувача надійдуть значення з елементів стеків TOS та TOS+1, а результат операції буде записаний на вершину стеку;

4) значення результату записати до пам'яті даних за певною адресою для можливості його використання при подальших обчисленнях або його зчитування сторонніми пристроями.

Наявність подібних мікрооперацій дає можливість скласти з них послідовність, виконання якої призведе до отримання очікуваного результату. Така послідовність називається *мікропрограмою*, виконання якої призводить до вироблення цифровим кінцевим автоматом певної послідовності керуючих сигналів для операційних блоків.

*Акумуляторна архітектура* розуміє наявність спеціального регістра (акумулятора), який виконує роль регістра зберігання результату виконаної операції і регістра одного з операндів. Структура тракту з акумуляторною архітектурою подана на рис. 3.9, вона має такі блоки, як пам'ять даних, регістр-акумулятор і блок операційних пристроїв.

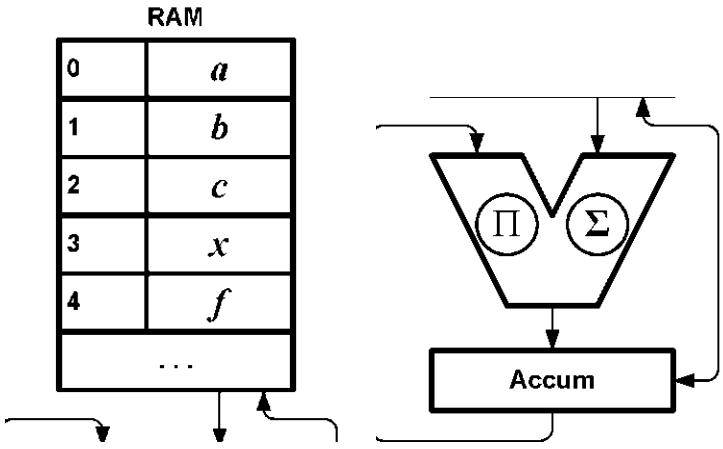



Рис. 3.9. Приклад акумуляторної архітектури

Базовими мікрооперaціями для цієй архітектури можуть бути такі дії:

1) завантаження адресованих даних з пам'яті даних в акумулятор;

2) збереження значення акумулятора в пам'яті даних за певною адресою;

3) виконання операції, для якої вказується адреса другого операнда, що зберігається в пам'яті даних, з автоматичним збереженням результату в акумуляторі.

Подібна архітектура дозволяє ефективно реалізовувати послідовності однотипних операцій, для яких значення проміжного результату накопичується в регістрі-акумуляторі.

*Архітектура регістрів загального призначення* (GPR, від англ. General Purpose Register) розуміє використання елементів пам'яті даних як регістри операндів, проміжних і кінцевих результатів. При цьому всі регістри є рівноправними і не реалізують специфічні функції, наприклад, як регістр-акумулятор. Структура тракту обробки даних з архітектурою GPR подана на рис. 3.10.

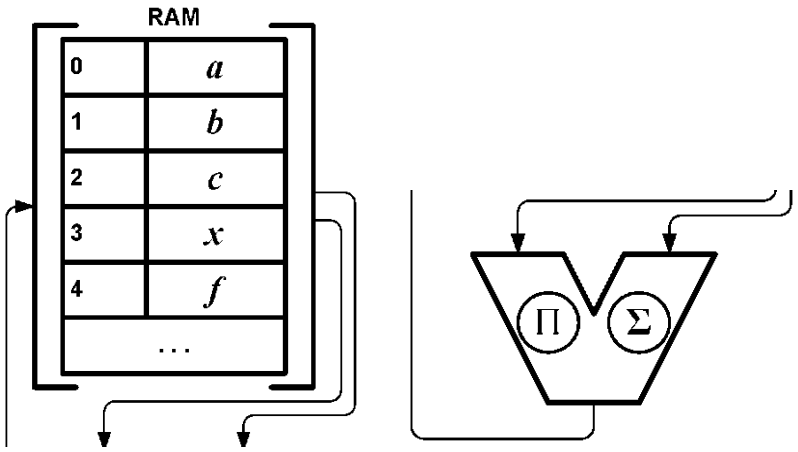



Рис. 3.10. Приклад архітектури регістрів загального призначення

З урахуванням того, що пам'ять даних може бути реалізована як багатопортовий регістровий файл, можна визначити таку базову мікрооперацію: виконання операції, для якої вказуються адреси операндів, що зберігаються в пам'яті даних, і адреса, за якою має бути записаний результат.

Тип архітектури визначає не тільки структуру тракту обробки даних, а й служить відправною точкою при проектуванні мікропрограми, яка у свою чергу впливає на організацію пам'яті інструкцій і функціонування пристрою управління.

## 3.4. Проектування набору інструкцій і складання мікропрограми

Кожна мікрооперація розуміє виконання відповідної інструкції пристроєм управління. Значення інструкції залежно від виду архітектури має визначати джерела операндів, тип виконуваної операції і приймач результату. Кожній інструкції присвоюється мнемонічне ім'я і визначається кількість і вид операндів (аргументів). Після складання набору необхідних інструкцій можна приступати до проектування мікропрограми.

Для стекової архітектури визначимо два типи інструкцій: інструкції, що відповідають за пересилання даних між пам'яттю і стеком, та інструкції арифметичних операцій. Для пересилання даних з пам'яті на вершину стеку необхідне зазначення адреси слова, що зберігається в пам'яті. При пересиланні значення, що зберігається на вершині стеку, до пам'яті даних також потрібно мати адресу, за якою буде здійснена операція запису. Виходячи з цього визначимо такі інструкції:

— **PUSH** *adr* : пересилання даних, що зберігаються в пам'яті за адресою *adr,* на вершину стека TOS;

— **POP** *adr* : пересилання значення вершини стеку в пам'ять за адресою *adr.*

З урахуванням того, що при виконанні операцій множення і додавання в стековій архітектурі джерелами операндів завжди є елементи TOS і TOS+i, відповідні інструкції можуть бути безоперандними:

— **MUL** : операція множення, результат записується на вершину стеку;

— **ADD** : операція додавання, результат записується на вершину стеку.

Через те, що вихідні дані зберігаються в пам'яті за фіксованими адресами, як операнди інструкцій PUSH і POP можна вказувати ім'я відповідної змінної. Беручи до уваги наведені вище обмеження складемо мікропрограмму для знаходження значення f (дод. 2, лістинг 3.1).

Кожен рядок наведеного лістингу містить одну інструкцію, забезпечену рядковим коментарем, розташованим після оператора ';'. Таким чином, наведена мікропрограма може бути

розглянута як вихідний код мовою асемблера цифрового пристрою з МПУ стекової архітектури. Виконання десяти інструкцій у чітко визначеній лінійній послідовності призведе до обчислення значення виразу (3.1) і запису отриманого результату до пам'яті даних за адресою 4. Закінчення процесу обчислення також має бути передбачено для можливості вироблення пристроєм управління відповідного значення на вихідному порту *Stop* (рис. 3.1). Для реалізації зупинки процесу обробки даних можна ввести спеціальний регістр-лічильник із задаванням кількості ітерацій, відповідних кількості виконуваних інструкцій. Однак одним з раціональних рішень є введення додаткової інструкції зупинки. Виконання подібної інструкції викликає припинення виконання всіх інших інструкцій з виставленням сигналу закінчення *Stop*. Таким чином, для реалізації повноцінної мікропрограми обчислення виразу (3.1) для стекової архітектури знадобиться п'ять базових інструкцій і мікропрограма, що складається з 11 послідовно виконуваних інструкцій.

Кожна інструкція має нести інформацію про тип виконуваної операції і про використані операнди, якщо такі є. Для розглянутого прикладу дві інструкції **PUSH** та **POP** є одноопераднними, а решта **(ADD, MUL, HALT)** — безоперандними. Перераховані інструкції ототожнюються з п'ятьма унікальними операціями. У зв'язку з цим кожна інструкція має бути забезпечена відповідною унікальною міткою для її однозначної ідентифікації пристроєм управління. У цьому випадку загальним форматом всіх інструкцій стекової архітектури може бути структура даних, що містить два поля: <тип операції> <адреса>. Поле типу операції може складатися, як мінімум, з трьох двійкових розрядів, необхідних для ідентифікації наявних інструкцій ($[\log_2 5]=3$). Адресне поле також може складатися, як мінімум, з трьох розрядів, необхідних для адресації даних, що зберігаються в адресному діапазоні від 0 до 4. На основі цього сформуємо таблицю кодування інструкцій стекової архітектури (табл. 3.1).

Символ $a \in \{0,1\}$ визначає відповідний розряд адреси пам'яті даних, символ $x \in \{0,1\}$ – будь-яке значення. На основі інформації з таблиці кодування інструкцій можна уявити спроектовану раніше мікропрограмму в бінарному вигляді (дод. 2, лістинг 3.2).

Таблиця 3.1

Таблиця кодування інструкцій для стекової архітектури

| Мнемоніка | Код операції | | | Адреса | | |
|---|---|---|---|---|---|---|
| PUSH | 0 | 0 | 0 | a | a | a |
| POP | 0 | 0 | 1 | a | a | a |
| ADD | 0 | 1 | 0 | x | x | x |
| MUL | 0 | 1 | 1 | x | x | x |
| HALT | 1 | 0 | 0 | x | x | x |
| Номер розряду | 5 | 4 | 3 | 2 | 1 | 0 |

Таким чином, бінарне подання мікропрограми має інформаційний обсяг, що дорівнює 66 бітам, а для зберігання мікропрограми необхідний цифровий запам'ятовуючий пристрій з підтримкою словників організацією 16x6 біт.

Міркуючи аналогічним чином, визначимо основні інструкції, необхідні для складання мікропрограми для акумуляторної архітектури. Як і в разі стекової архітектури, для акумуляторної архітектури введемо три типи інструкцій: інструкції, що відповідають за пересилання даних між пам'яттю і акумулятором, інструкції арифметичних операцій і єдину інструкцію зупинки обчислювального процесу. З урахуванням специфіки акумуляторної архітектури перші два типи інструкцій будуть одноопераднимиі:

— **LOAD** *adr* : пересилання даних, що зберігаються в пам'яті за адресою *adr,* в акумулятор;

— **STORE** *adr* : пересилання значення акумулятора в пам'ять за адресою *adr;*

— **ADD** *adr* : операція додавання, операндами якої є значення акумулятора і значення, що зберігається за адресою *adr,* результат операції записується в акумулятор;

— **MUL** *adr* : операція множення, операндами якої є значення акумулятора і значення, що зберігається за адресою *adr,* результат операції записується в акумулятор;

— **HALT**: безоперандна операція зупинки.

Складемо таблицю кодування представлених інструкцій (табл. 3.2).

Таблиця 3.2

Таблиця кодування інструкцій для акумуляторної архітектури

| Мнемоніка | Код операції | | | Адреса | | |
|---|---|---|---|---|---|---|
| LOAD | 0 | 0 | 0 | a | a | a |
| STORE | 0 | 0 | 1 | a | a | a |
| ADD | 0 | 1 | 0 | a | a | a |
| MUL | 0 | 1 | 1 | a | a | a |
| HALT | 1 | 0 | 0 | x | x | x |
| Номер розряду | 5 | 4 | 3 | 2 | 1 | 0 |

Мікропрограма обчислення значення виразу (3.1) для акумуляторної архітектури та її бінарне подання з урахуванням наведеної таблиці буде виглядати так, як у дод. 2, лістинг 3.3.

Мікропрограма складається з семи інструкцій, а її бінарне подання має інформаційний обсяг, рівний 42 бітам. Для зберігання мікропрограми необхідний цифровий запам'ятовуючий пристрій з підтримкою словників організацією 8x6 біт.

При проектуванні мікропрограми для GPR-архітектури можна враховувати, що пам'ять даних являє собою регістровий багатопортовий файл. Для двоопераднних інструкцій кращим рішенням є використання регістрового файла з двома портами читання і одним портом запису, що дозволить за одне звернення зчитувати два значення. З огляду на це для GPR-архітектури введемо такі інструкції:

— **ADD** *adr1, adr2, adr3:* операція додавання двох операндів, значення яких зберігаються за адресами *adr2* і *adr3*, а результат операції записується в пам'ять за адресою *adr1;*

— **MUL** *adr1, adr2, adr3:* операція множення двох операндів, значення яких зберігаються за адресами *adr2* і *adr3*, а результат операції записується в пам'ять за адресою *adr1;*

— **HALT** : безоперандна операція зупинки.

Таблиця кодування поданих інструкцій може виглядати так, як у табл. 3.3.

На основі наявних інструкцій складемо мікропрограмму і її бінарне подання (дод. 2, лістинг 3.4).

Таблиця 3.3

Таблиця кодування інструкцій для GPR-архітектури

| Мнемоніка | Код операції | | Адреса | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ADD | 0 | 0 | **a**1 | **a**1 | **a**1 | a2 | a2 | a2 | a3 | a3 | a3 |
| MUL | 0 | 1 | **a**1 | **a**1 | **a**1 | a2 | a2 | a2 | a3 | a3 | a3 |
| HALT | 1 | 0 | X | x | x | x | x | x | x | x | x |
| Номер розряду | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Подана мікропрограма містить п’ять інструкцій і має інформаційний обсяг, рівний 55 бітам, що вимагає наявності відповідного ЗП словникової організації 8x11 біт. Розроблені мікропрограми для різних архітектур тракту обробки даних можуть бути використані для попереднього оцінювання апаратних витрат, необхідних для реалізації пам'яті мікропрограми, і часових витрат, необхідних для її виконання. Основними показниками для оцінювання є кількість використаних інструкцій у мікропрограмі та її інформаційний об’єм у бітах. На рис. 3.11 подана гістограма, що відображує дані показники так: центр кожного кола розташований у точці з координатами, числові значення яких дорівнюють кількості використаних інструкцій, а радіуси кіл прямо пропорційні інформаційному об’єму відповідних мікропрограм.

Чим вище знаходиться коло на наведеній гістограмі, тим більшу кількість інструкцій необхідно виконати для повної реалізації мікропрограми. Чим більше площа кола, тим більший інформаційний об’єм запам'ятовуючого пристрою, необхідного для зберігання бінарного коду мікропрограми. Компромісним вибором з поданих архітектур для реалізації обчислення виразу (3.1) є акумуляторна архітектура, що вимагає використання запам'ятовуючого пристрою з мінімальним інформаційним об’ємом для зберігання мікропрограми. При цьому число використовуваних інструкцій є усередненою відносно стекової і GPR-архітектури.

Наведені критерії вибору архітектурного рішення є далеко не повними. Наприклад, з точки зору апаратної реалізації пристрою управління, зокрема схем декодування, краще

використовувати стекову архітектуру, для якої використовуються одноопера́ндні інструкції, на відміну від трьохоперандних для GPR-архітектури. Однак визначальну роль у виборі відповідної архітектури відіграє початкове завдання або низка завдань, що вимагають реалізації на цифровому пристрої.

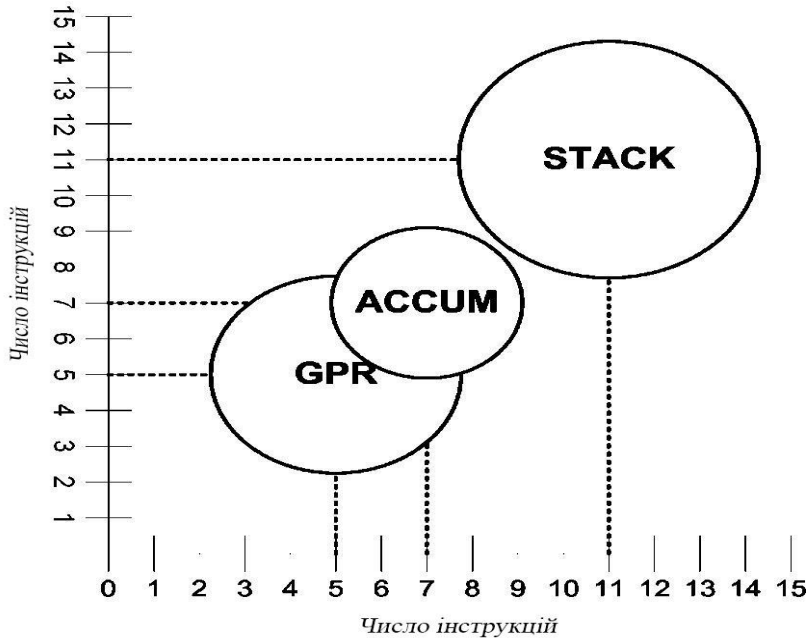



Рис. 3.11. Оцінна гістограма мікропрограм для різних архітектур

## 3.5. Проектування пристроїв управління

Основним призначенням пристрою управління є узгодження функціонування апаратури тракту обробки даних і пам'яті мікропрограм. У загальному випадку завдання синтезу пристрою управління можна інтерпретувати як завдання проектування цифрового кінцевого автомата, який виробляє послідовність сигналів для управління підпорядкованими пристроями, які по суті є послідовністю виконуваних інструкцій мікропрограми, що зберігається. Для реалізації обчислювального алгоритму мікропрограми з урахуванням її лінійності пристрій управління може функціонувати відповідно до такої послідовності:

1) встановити значення регістра адреси пам'яті мікропрограми *adr* = 0;

2) при встановленні вхідного сигналу *Start*, рівним 1, перейти до п. 3;

3) зчитати з пам'яті мікропрограми за адресою *adr* значення інструкції *inst;*

4) провести декодування інструкції *inst* з метою визначення її типу; у разі декодування інструкції зупинки (**HALT**) перейти до п. 9;

5) за необхідності зчитати значення операндів інструкції з пам'яті даних;

6) виконати операцію, кодовану інструкцією *inst*;

7) за необхідності записати результат виконаної операції в пам'ять даних;

8) збільшити значення адреси adr на одиницю і перейти до п. 3;

9) виробити значення вихідного сигналу *Stop* = 1.

Наведена послідовність вимагає наявності таких функціональних блоків пристрою управління:

— двійкового лічильника адреси інструкції **IC;**
— регістра обраної інструкції **RI;**
— регістра адреси даних **RA;**
— регістра обраних даних **RD;**
— регістра операційного коду **RO;**
— дешифратора операційного коду **DCO;**
— контролера доступу до пам'яті мікропрограми **ROMC;**
— контролера доступу до пам'яті даних **RAMC;**
— цифрового кінцевого автомата **FSM.**

Розглянемо приклад деталізованої структури пристрою управління для акумуляторної архітектури (рис. 3.12).

Функціональний блок FSM відповідає за злагоджену роботу всіх інших блоків, у тому числі і за вироблення значень сигналу Stop. Ініціалізація обчислювального процесу починається зі встановлення активного значення сигналу на вхідному порті RST пристрою управління. При цьому блок FSM виробляє сигнал установлення значення лічильника IC в нульовий стан. Виконання обчислювального процесу починається при настанні переднього фронту сигналу CLK після установлення активного значення сигналу на вхідному порту Start. Блок FSM за допомогою контролера ROMC виробляє необхідні керуючі сигнали для здійснення операції читання інструкції, яка зберігається за адресою, що визначається значенням лічильника IC. Вибране значення інструкції записується в регістр RI. При

цьому старші три розряди регістра RI записуються в регістр операційного коду RO, а молодші три розряди – у регістр адреси даних RA. Далі блок FSM декодує старші розряди регістра RI для визначення типу інструкції. При виборі інструкції HALT блок FSM припиняє своє функціонування і виробляє середнє арифметичне значення сигналу на вихідному порту *Stop*. При декодуванні інструкції STORE управління передається контролеру RAMC, який здійснює запис значення регістра Accum у пам'ять даних за адресою, що визначається значенням регістра RA.

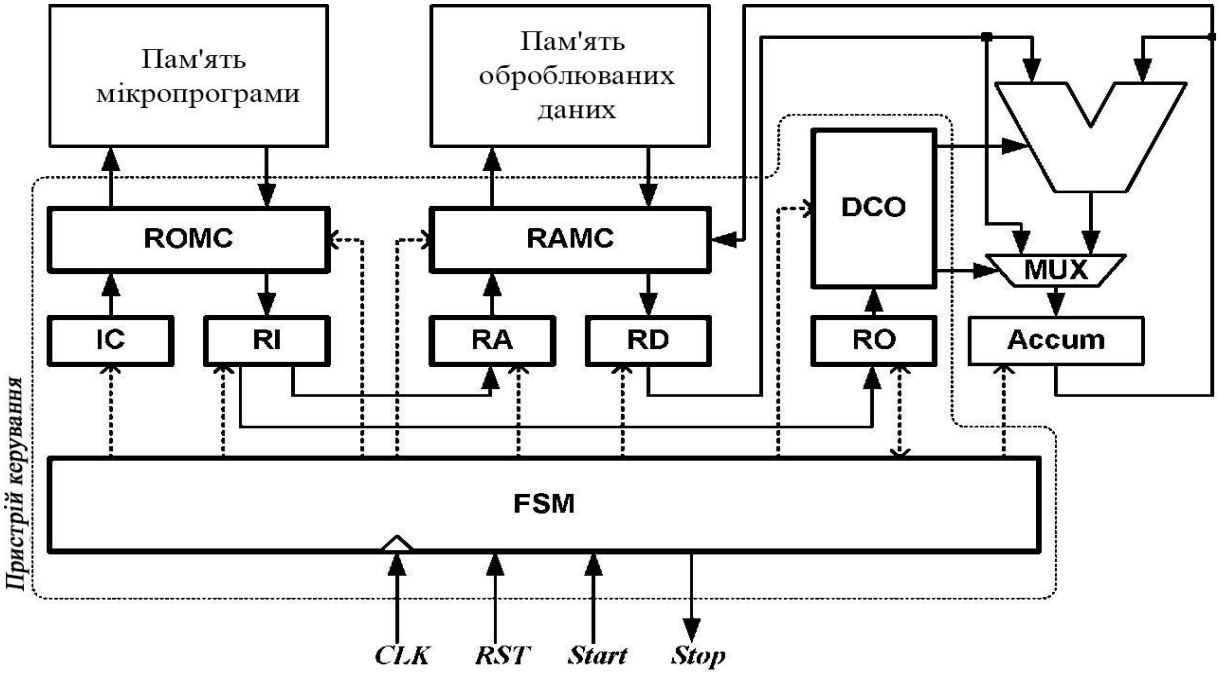



Рис. 3.12. Структура пристрою управління для акумуляторної архітектури

Виконання інструкцій LOAD, ADD і MUL вимагає початкового вилучення операнда з пам'яті оброблюваних даних. У зв'язку з цим після декодування перерахованих інструкцій блок FSM передає управління контролера RAMC для здійснення операції читання даних за адресою, що визначається значенням регістра RA. Прочитане значення розміщується в регістр RD. При виконанні інструкції LOAD блок FSM виробляє необхідні керуючі сигнали для дешифратора DCO, який забезпечує запис значення регістра RD в акумулятор. У разі виконання інструкцій ADD і MUL здійснюється відповідна операція додавання або

множення, першим операндом для якої є значення регістра RD, другим – значення акумулятора Accum. Результат виконаної операції записується в регістр Accum.

Після виконання операції (за винятком операції зупинки) блок FSM виробляє відповідний керуючий сигнал, що забезпечує збільшення значення лічильника IC на одиницю. Потім блок FSM повторює свій операційний цикл до закінчення обчислювального процесу.

На основі вищеописаного складемо діаграму станів кінцевого автомата функціонального блока FSM (рис. 3.13).

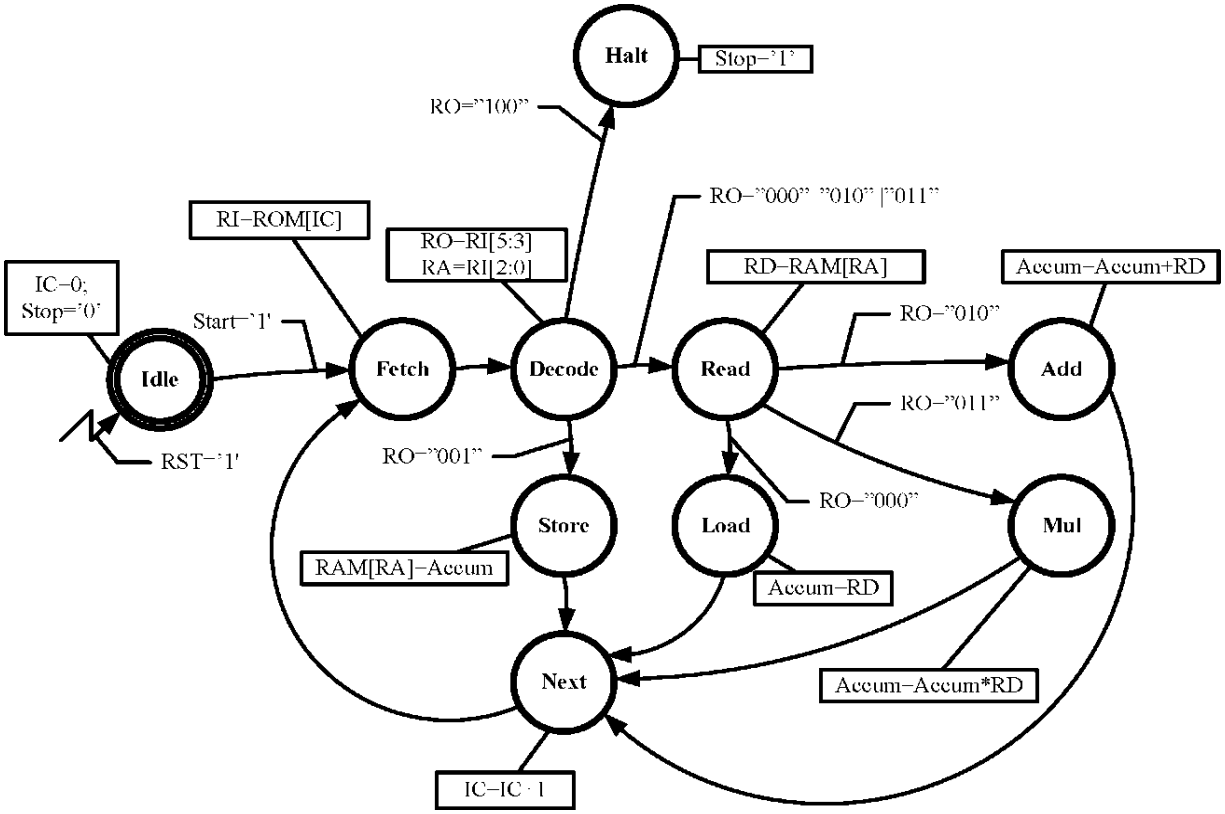



Рис. 3.13. Діаграма станів кінцевого автомата пристрою управління

Стан **Idle** є ініціалізуючим станом цифрового автомата, у який він перемикається з настанням асинхронної події на вхідному порті *RST*. При цьому лічильник IC ініціалізується нульовим значенням, і значення сигналу на вихідному порті *Stop* також встановлюється в нуль. Перехід у наступний стан можливий лише за умови, що на вхідному порті *Start* буде встановлено середнє арифметичне значення.

У стані **Fetch** здійснюється звернення до пам'яті мікропрограми з метою отримання значення інструкції, що зберігається за адресою IC, з подальшим її записом у регістр RI.

Після вилучення інструкції автомат перемикається в стан Decode, в якому здійснюється копіювання коду операції з регістра RI в регістр RO і копіювання адресного поля з регістра RI в регістр RA. При значенні RO = «100» обрана інструкція зупиняє операцію, при виконанні якої автомат перемикається в стан **Halt** і виробляє середнє арифметичне значення сигналу на вихідному порті Stop.

На поданій діаграмі стан **Halt** є тупиковим станом, перехід з якого можна здійснити тільки за умови RST = '1' у стан **Idle**. Якщо в стані **Decode** значення RO = «001», то автомат переходить у стан **Store**, у якому записується значення акумулятора в пам'ять даних за адресою, що визначається значенням регістра RA (табл. 3.2).

Три інструкції (LOAD, ADD і MUL) вимагають попереднього читання значення явного операнда з пам'яті даних. У зв'язку з цим після декодування перерахованих інструкцій автомат перемикається в стан **Read**, ув якому зчитується значення з пам'яті даних за адресою, що визначається значенням регістра RI, із записом цього значення в регістр RD.

Автомат, перебуваючи в одному з трьох станів (**Load, Add, Mul**), здійснює відповідну операцію, яка призводить до зміни значення регістра Accum. Після виконання декодованої інструкції автомат перемикається в стан **Next**, у якому здійснюється збільшення значення лічильника IC з подальшим переходом автомата в первісний стан **Fetch.**

Маючи інформацію про функціонування пристрою управління, можна оцінити час виконання кожної інструкції і всієї мікропрограми в цілому. Наприклад, виконання інструкцій ADD і MUL вимагає п’яти періодів сигналу синхронізації кінцевого автомата пристрою управління, а виконання інструкції зупинки – трьох періодів (рис. 3.14).

З огляду на це час виконання всієї мікропрограми, що містить сім інструкцій, становить 33 періоди синхронізації (дод. 2, лістинг 3.3). Якщо мікропрограма є лінійною, як у розглянутому прикладі, то зміну значення лічильника IC можна

здійснювати безпосередньо після вибору виконуваної інструкції, наприклад у стані **Decode.** При цьому стан **Next** вимикається, а переходи зі станів **Load, Store, Add** і **Mul** здійснюються безпосередньо в стан **Fetch**. Вилучення стану **Next** дозволить скоротити час виконання мікропрограми на шість періодів синхронізації, що складає близько 18 % загального часу виконання.

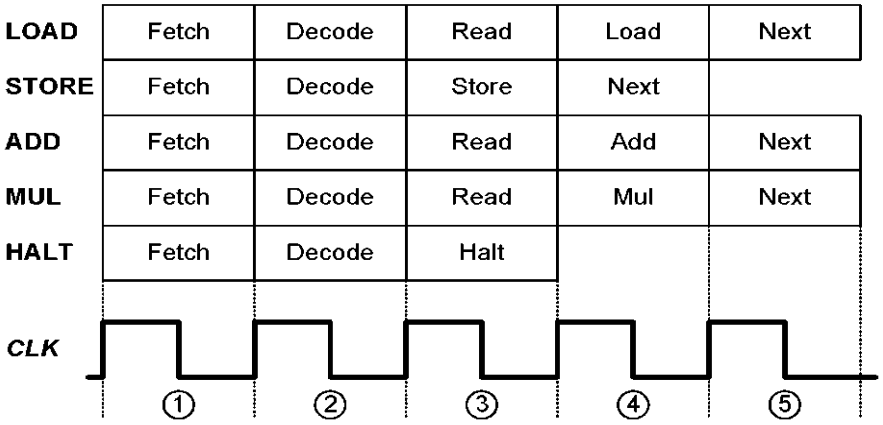



Рис. 3.14. Діаграма виконання інструкцій для акумуляторної архітектури

Час виконання мікропрограми можна ще скоротити шляхом реорганізації обробки потоку виконуваних інструкцій за допомогою *конвеєра*. Конвеєрне виконання інструкцій означає одночасне спрацьовування різних функціональних блоків пристрою управління для послідовно добуваних інструкцій. Наприклад, після вилучення інструкції, що зберігається за адресою *adr*, автомат переходить у стан її декодування, при цьому існує можливість одночасно витягувати наступну інструкцію, що зберігається за адресою *adr* + 1, тощо. Однією з умов застосування конвеєрного принципу виконання інструкцій є наявність однакових стадій їх виконання.

Для розглянутого прикладу всі інструкції можна привести до єдиних чотирьох послідовних стадій виконання:

1) **FETCH** – витягування інструкції з пам'яті мікропрограми;

2) **DECODE** – декодування витягнутої інструкції (ініціалізація регістрів RO і RA);

3) **READ** – читання даних з пам'яті з записом їх у регістр RD;

4) **WRITE** – запис результату в акумулятор або пам'ять даних.

По суті стадії **READ** і **WRITE** є зайвими для відповідних інструкцій **STORE** і **HALT**, але є необхідними для організації конвеєрної обробки. Наприклад, якщо в поточному періоді сигналу синхронізації здійснюється четверта стадія для інструкції *i*, то одночасно з цим існує можливість здійснювати стадію **READ** для інструкції *i* + 1, декодувати інструкцію *i* + 2 і вибирати інструкцію *i* + 3. Беручи до уваги одночасність виконання різних стадій для різних інструкцій, складемо діаграму конвеєрної обробки мікропрограми, поданої в лістингу 3.2 дод. 2 (рис. 3.15).

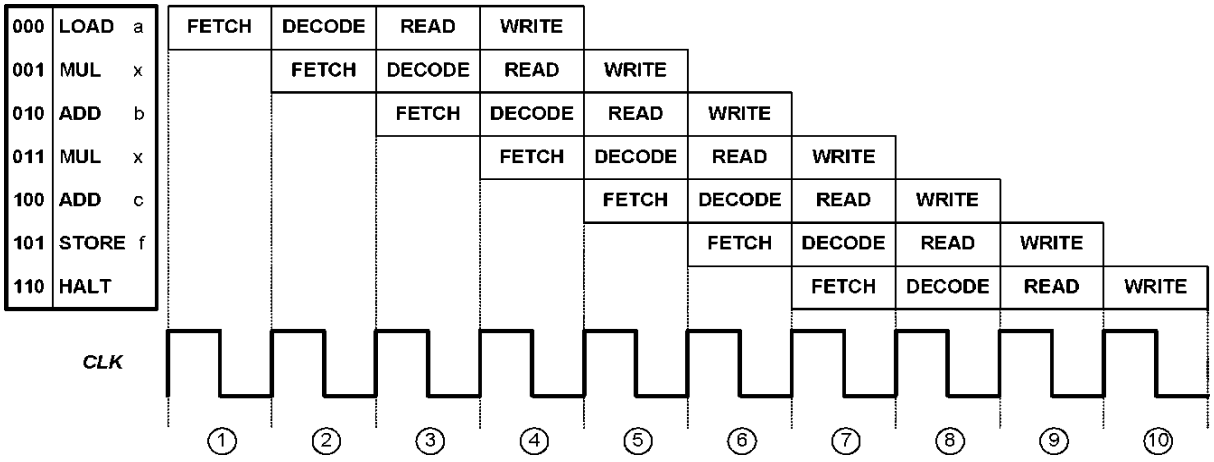



Рис. 3.15. Діаграма конвеєрного виконання мікропрограми

Як видно з наведеної діаграми, час виконання мікропрограми дорівнює 10 періодам синхроімпульсу CLK, що практично в три рази менше часу виконання для першого варіанта реалізації. Для забезпечення конвеєрного виконання потоку інструкцій необхідна реорганізація функціональних блоків пристрою управління з метою послідовної передачі результуючих даних від однієї стадії до наступної. З огляду на те, що дані передаються за допомогою регістрів, визначимо, які регістри і яким чином використовуються на відповідних стадіях виконання:

1) **FETCH** — читання з регістра IC, запис у регістри IC і RI;

2) **DECODE** — читання з регістра RI, запис у регістри RO і RA;

3) **READ** — читання з регістрів RO і RA, запис у регістр RD;

4) **WRITE** — читання з регістрів RO, RA, RD.

Як видно, практично на всіх стадіях здійснюються операції читання і запису різних регістрів. Введемо для кожного з перерахованих регістрів їхні копії, які на початку кожної стадії будуть доступні для операції читання, а в кінці кожної стадії будуть модифікуватися новим значенням, що буде доступно на початку наступної стадії. Нехай ім'я регістра з індексом **_r** означатиме копію регістра, значення якого буде доступно для читання на початку стадії, а ім'я регістра з індексом **_w** – копію регістра, значення якого буде модифікуватися в кінці стадії. Нехай перехід від однієї стадії конвеєра до іншої здійснюється синхронно з настанням переднього фронту сигналу синхронізації. Тоді для визначення початку кожної стадії також будемо використовувати передній фронт сигналу синхронізації, а для визначення закінчення стадії – задній фронт сигналу синхронізації. На рис. 3.16 подана діаграма передачі даних між копіями регістрів на різних стадіях конвеєрної обробки для двох інструкцій, що зберігаються за суміжними адресами і на *i*+1 пам'яті мікропрограми.

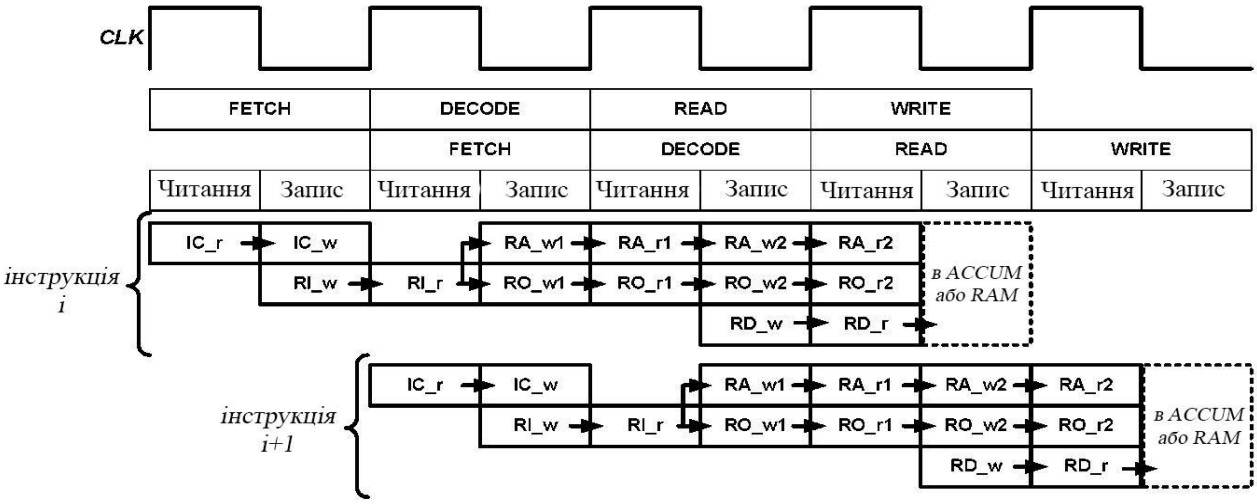



Рис. 3.16. Діаграма передачі даних між копіями регістрів

Наприклад, значення лічильника IC інкрементується в кінці стадії FETCH (запис у копію регістра IC_w) для можливості використання нового значення (читання копії регістра IC_r) на

початку стадії FETCH для наступної інструкції. Значення регістрів RO і RA мають бути доступними на двох останніх стадіях конвеєра для можливості декодування відповідної операції при здійсненні доступу до пам'яті даних. На останній стадії WRITE спочатку відбувається підготовка необхідних копій реєстрів RO, RA і RD для подальшого здійснення операції запису в акумулятор або пам'ять RAM.

Розглянемо докладніше приклад проектування цифрового пристрою з мікропрограмним управлінням з використанням мови VHDL для цільової технології FPGA.

## 3.6. Приклад проектування цифрового пристрою з мікропрограмним управлінням

Почнемо проектування зі складання поведінкових VHDL-описів пам'яті мікропрограми, пам'яті оброблюваних даних і тракту обробки даних у вигляді окремих компонент. Потім спроектуємо дві компоненти пристрою управління: одна компонента буде являти собою звичайний цифровий кінцевий автомат, друга – конвеєрний пристрій управління. Після проектування перерахованих компонент скористаємося структурним описом для їх складання в єдину модель цифрового пристрою з мікропрограмним управлінням. Далі наведемо результати функціонального моделювання двох варіантів цифрових пристроїв і оцінимо апаратурні витрати на їх реалізацію.

Почнемо з компоненти MROM запам'ятовуючого пристрою, призначеного для зберігання мікропрограми. Спроектуємо компоненту MROM як ПЗП інформаційною ємністю 6х8 біт з можливістю асинхронного читання довільно адресованої інструкції у вигляді двійкового слова розмірністю 6 біт. Значення бінарного подання виконуваної мікропрограми задамо як константні значення безпосередньо в VHDL-описі (дод. 2, лістинг 3.5).

Як видно з наведеного опису, читання інструкції можливе за умови подачі значення її адреси на вхід адресної шини *ADR* і одиничного значення сигналу на вхідний порт *RE* (рис. 3.17).

Компоненту запам'ятовуючого пристрою, що зберігає оброблювані дані, опишемо у вигляді ОЗП інформаційною

ємністю 8x8 біт з попередньо встановленими значеннями змінних відповідно до їхнього розміщення, поданому раніше на рис. 3.9. У дод. 2 (лістинг 3.6) наведено поведінковий опис компоненти пам'яті даних MRAM, а на рис. 3.18 – часову діаграму її функціонального моделювання.

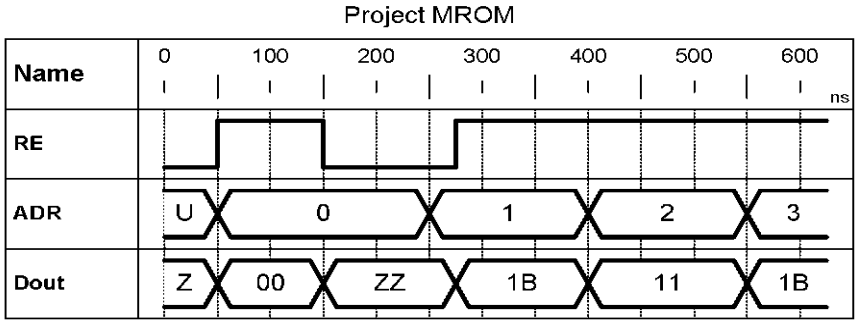



Рис. 3.17. Часова діаграма функціонального моделювання компоненти MROM

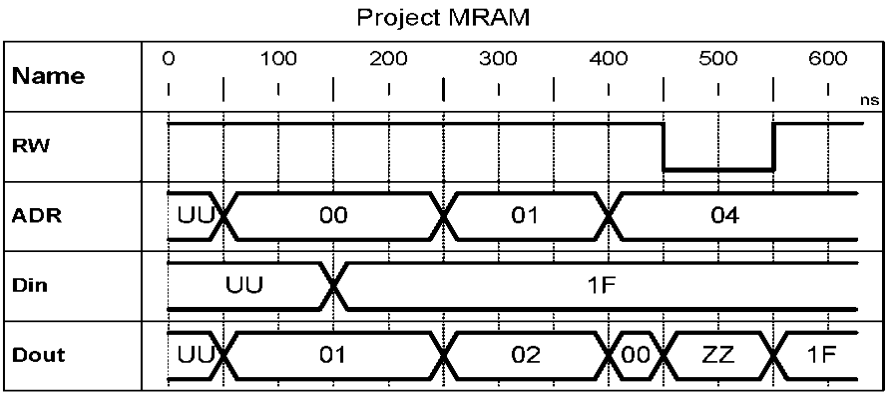



Рис. 3.18. Часова діаграма функціонального моделювання компоненти MRAM

Переходимо до проектування компоненти тракту обробки даних DPTH. Тракт обробки включає такі компоненти: двійковий суматор, двійковий помножувач, декодер типу операції і синхронний регістр-акумулятор. На входи тракту будуть подаватися значення операнда, обраного з пам'яті даних, і значення коду операції, обраного з пам'яті мікропрограми. На вихід тракту транслюватиметься значення, що зберігається в регістрі-акумуляторі. Для опису схем суматора і помножувача

скористаємося високорівневими операторами додавання і множення для сигналів типу integer (дод. 2, лістинг 3.7).

У розглянутому прикладі результати операцій додавання і множення обмежуються молодшими вісьмома розрядами для можливості їх запису і зберігання в пам'яті даних. Часова діаграма функціонального моделювання компоненти DPTH наведена на рис. 3.19.

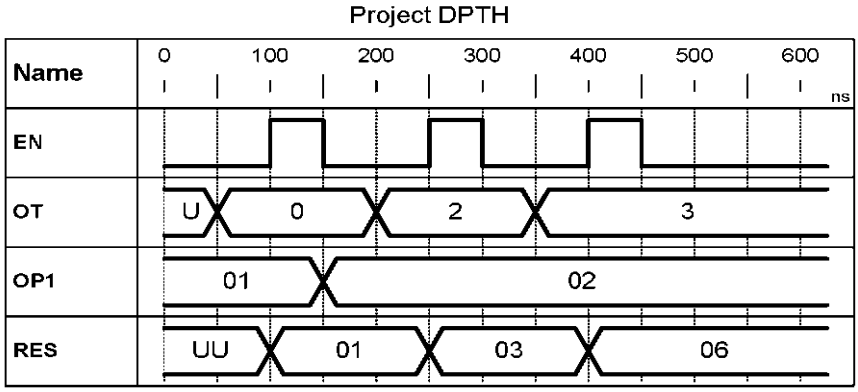



Рис. 3.19. Часова діаграма функціонального моделювання тракту обробки даних

На часовій діаграмі показано приклад, що імітує послідовне виконання інструкцій LOAD, ADD і MUL (OT = {0,2,3}). Значення вихідної шини RES є копіями значень регістра ACCUM.

Переходимо до проектування пристрою управління. Спочатку складемо VHDL-опис першого варіанта пристрою управління CTRL1, взявши за основу цифровий кінцевий автомат, який послідовно буде обирати і виконувати інструкції з компоненти MROM згідно з діаграмою, поданою на рис. 3.13. У дод. 2 (лістинг 3.8) наведено поведінковий опис компоненти CTRL1, що виробляє керуючі сигнали для компонент MROM, MRAM і DPTH.

Альтернативним варіантом реалізації пристрою управління є компонента CTRL2, яка забезпечує конвеєрну обробку потоку інструкцій, які обираються з пам'яті мікропрограм. Поведінковий опис компоненти CTRL2 подано в дод. 2 (лістинг 3.9).

Після того як всі компоненти готові, переходимо до реалізації структурного опису компоненти вищого рівня ієрархії TOP, у якій визначимо міжз'єднання всіх компонент. У дод. 2

(лістинг 3.10) наведено структурний опис компоненти TOP. Для використання різних варіантів компоненти пристрою управління необхідно вказати ім'я компоненти CTRL1 або CTRL2.

Після складання VHDL-опису компоненти TOP переходимо до її функціонального моделювання. Спочатку змоделюємо компоненту TOP з включеною в неї компонентою пристрою управління CTRL1, спроектованою на основі кінцевого автомата (рис. 3.20).

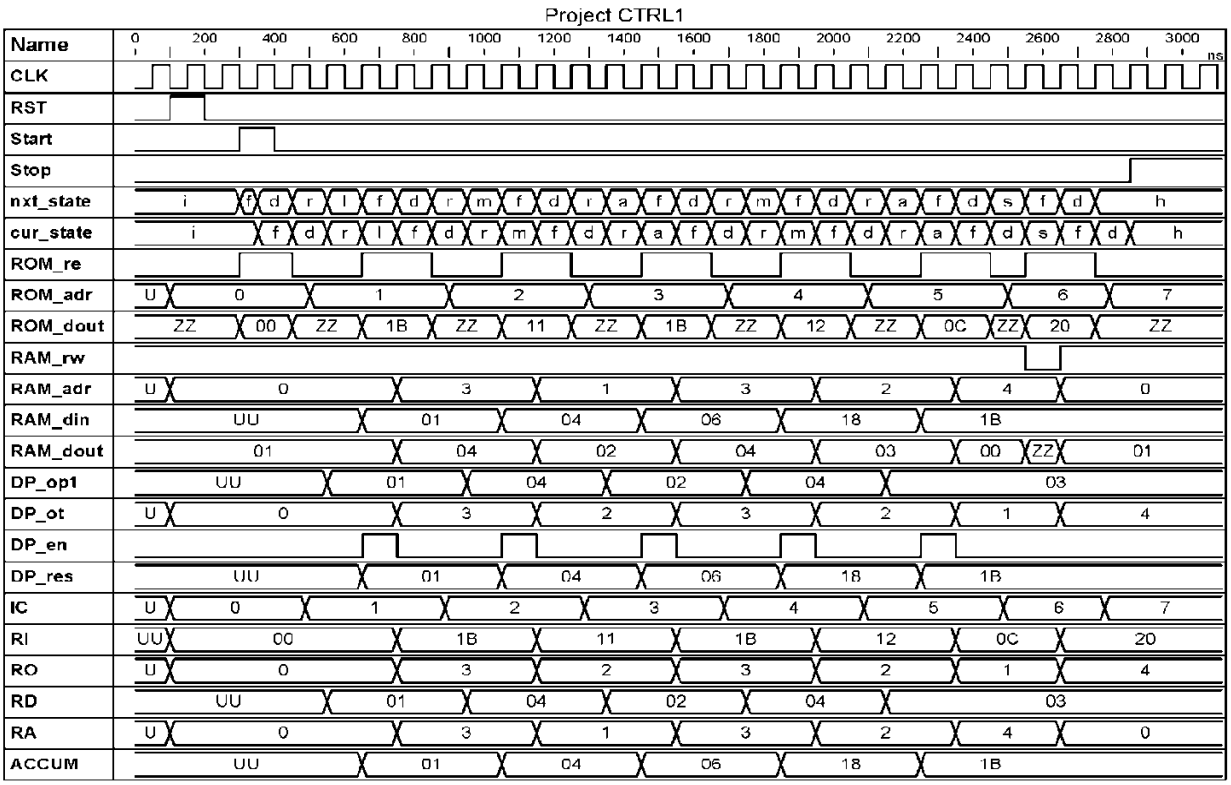



Рис. 3.20. Часова діаграма функціонального моделювання пристрою управління CTRL1

Частота сигналу синхронізації CLK дорівнює 10 МГц. У період з 100 до 200 нс на часовій шкалі моделювання сигнал ініціалізації утримувався в стані логічної одиниці. При цьому автомат брав поточне ініціалізуюче значення I (Idle). Після процесу ініціалізації значення сигналу *Start* встановлюється в активний стан, що сприяє початку обчислювального процесу в момент часу, рівний 350 нс, за умови настання переднього фронту сигналу синхронізації. У цей момент автомат перемикається в стан вибору інструкції F (Fetch), у якому

виробляється середнє арифметичне значення сигналу дозволу читання для компоненти MROM (ROM_re = '1'). Значення прочитаної інструкції записується в регістр RI, і автомат перемикається в стан декодування D (Decode). Обчислювальний процес триває до моменту декодування інструкції HALT (період з 2750 до 2850 нс), при якому автомат перемикається у відповідний стан H з виробленням одиничного значення сигналу *Stop*.

Час виконання обчислювального процесу T розрахуємо так: $T = t_{Stop} - t_{Start} = 2850 - 300 = 2550$ нс, де $t_{Stop}$ – момент установлення сигналу Stop='1', $t_{start}$ – момент установлення сигналу Start='1'. З урахуванням того, що період сигналу синхронізації $P_{CLK} = 100$ нс, час виконання можна оцінити як $T = 25{,}5 \times P_{CLK}$. Наведене значення часу виконання може бути використано тільки для порівняльного оцінювання швидкодії першого і другого варіантів реалізації цифрового пристрою з різними типами пристроїв управління.

На наведеній часовій діаграмі також можна спостерігати послідовну зміну значення регістра ACCUM компоненти DPTH і момент запису результуючого значення 1B (всі значення подані в шістнадцятковому форматі) у пам'ять даних за адресою 4 (POP f).

На рис. 3.21 подана часова діаграма функціонального моделювання пристрою управління CTRL2 в складі компоненти TOP.

Частота сигналу синхронізації також була обрана рівною 10 МГц. У тій самій послідовності, що і для компоненти CTRL1, були згенеровані значення сигналів на вхідних портах RST і *Start*. Час виконання обчислювального процесу в цьому випадку дорівнює $T_2 = t_{Stop} - t_{start} = 950$ нс. Значення $T_2$ в кількості періодів синхросигналу CLK оцінюється як $T_2 = 9{,}5 \times P_{CLK}$. Таким чином, значення коефіцієнта лінійного прискорення Kp виконання вихідної мікропрограми цифровим пристроєм з конвеєрною організацією становить $K_p = \frac{T_1}{T_2} \approx 2{,}68$.

Функціональне моделювання всіх розглянутих компонент проводилось за допомогою програмного засобу Active-HDL 8.2 [82]. Технологічний синтез двох версій проекту цифрового пристрою з мікропрограмним управлінням здійснювався за допомогою САПР Xilinx ISE [83] для ПЛІС SPARTAN-3E XC3s500e-5FG320. Результати технологічного синтезу наведені в табл. 3.4.

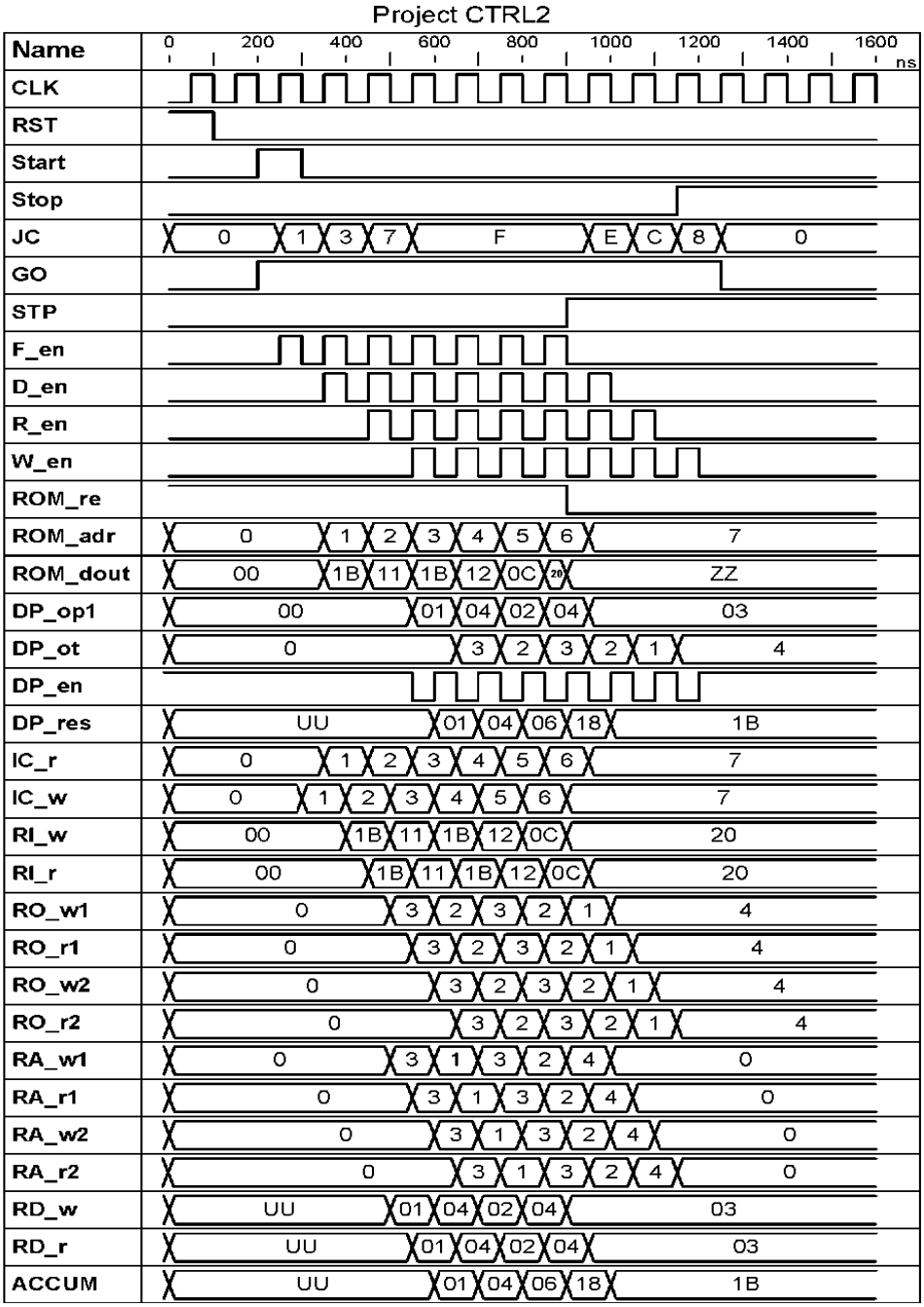



Рис. 3.21. Часова діаграма функціонального моделювання пристрою управління CTRL2

Найбільші апаратурні витрати має компонента пам'яті даних RAMQ, яка для двох варіантів реалізації становить понад 50 % загальних витрат для всього пристрою.

Таий результат викликано реалізацією пам'яті з використанням тригерів Slice-блоків, необхідних для зберігання 8 x 8 = 64 біт даних.

Таблиця 3.4

Результати технологічного синтезу спроектованих компонент

| Компонента | Технологічний примітив | | | |
|---|---|---|---|---|
| | Slices | Slice Flip-Flops | 4-input LUTs | MULT18x 18 |
| ROMC | 4 | 0 | 7 | 0 |
| RAMC | 53 | 64 | 41 | 0 |
| DPTH | 22 | 8 | 41 | 1 |
| CRTL1 | 14 | 24 | 18 | 0 |
| CTRL2 | 32 | 47 | 30 | 0 |
| TOP(CTRL1) | 93 | 96 | 107 | 1 |
| TOP(CTRL2) | 111 | 119 | 119 | 1 |

Найменшими витратами характеризується компонента MROM, призначена для зберігання мікропрограми. Для першого варіанта реалізації витрати становлять близько 3,5 %, для другого варіанта – близько 2,9 %. Для реалізації компоненти пристрою управління CTRL1 знадобилося близько 20 % усіх витрат апаратури цифрового пристрою, а для реалізації компоненти CTRL2 – 32,3 %.

У відносному порівнянні окремих компонент управління витрати на реалізацію CTRL2 перевершують більш ніж на 80 % компоненту CTRL1. Загалом перехід від використання кінцевого автомата до конвеєрної обробки потоку мікрокодів вимагав близько 17 % додаткової апаратури.

Незважаючи на істотне збільшення продуктивності, для розглянутого прикладу конвеєрна організація обчислювального процесу може потребувати великих апаратурних і часових витрат, пов'язаних з можливістю простоїв окремих стадій конвеєра [84].

У загальному випадку простої конвеєра можливі при реалізації інструкцій умовних і безумовних переходів, нелінійних змінах значення лічильника інструкцій або за необхідності очікування готовності проміжного результату, який обчислюється при виконанні попередніх інструкцій, тощо [85]. Усунення подібних простоїв можливе шляхом перегляду послідовності виконуваних операцій і/або введенням додаткової апаратури, наприклад для реалізації швидкого пересилання даних.

**Контрольні питання**

1. Що являє собою принцип ЗМ при побудові мікропроцесорних систем?
2. Принцип модульної організації МПС.
3. Магістральний спосіб обміну інформацією в МПС.
4. Мікропрограмна реалізація управління в МПС.
5. Що являє собою принцип агрегатування?
6. Організація одно- і багатомагістральних мікропроцесорних систем.
7. Типова структура тришинної МПС.
8. Призначення схеми синхронізації та початкового скидання.
9. Організація доступу до пам'яті МПС.
10. Організація доступу до пристроїв введення/виведення.
11. Призначення таймера в МПС.
12. Організація прямого доступу до пам'яті в МПС.
13. Призначення контролера переривань у МПС.
14. Поясніть поняття модульності, магістральності і мікропрограмованості МПС під час проектування.
15. Перелічіть завдання, вирішувані розробниками під час проектування МПС.
16. Перелічіть основні етапи проектування МПС.
17. Назвіть концептуальні рівні опису МПС під час проектування і розроблення.
18. Перелічіть основні методи контролю правильності проектування МПС.
19. Якими властивостями має володіти проектована МПС для виконання етапу її налагодження?
20. Перелічіть види несправності під час проектування МПС.
21. Назвіть причини фізичної і суб'єктивної несправностей МПС.
22. Поясніть поняття діагностики несправності, налагодження.

## 4. Запам'ятовуючі пристрої вбудованих систем

Проектування контролепридатних цифрових пристроїв базується на двох основних концепціях: спостережливості і керованості [75]. Під *спостережливістю* розуміють можливість трансляції логічних значень внутрішніх полюсів цифрової схеми на її вихідні порти (зовнішні виводи). *Керованість* визначає можливість встановлювати необхідні логічні значення на внутрішніх полюсах цифрової схеми за допомогою її вхідних портів (зовнішніх виводів). Подані концепції дозволяють здійснювати подачу тестових впливів, що виробляються генератором тестових послідовностей (ГТП), який підключається до вхідних портів тестованої схеми. У свою чергу пристрій аналізу реакцій схеми на тестові впливи підключається до вихідних портів.

Однією з основних проблем проектування контролепридатних цифрових пристроїв є використання додаткових зовнішніх виводів для підключення тестового обладнання. У разі реалізації цифрового пристрою як корпусного НВІС наявність додаткових контактів може бути лімітованою або взагалі неприйнятною. Для подолання цього обмеження широко застосовуються два підходи:

1) використання послідовних каналів передачі тестових даних і приймання реакцій від пристрою;
2) реалізація вбудованого самотестування цифрових пристроїв.

Перший підхід заснований на проектуванні внутрішньої схемотехніки цифрового пристрою з використанням зсувних регістрів, що дозволяють реалізовувати подачу тестових впливів і захоплення реакцій при мінімальній кількості додаткових виводів. Самотестування цифрових пристроїв, у тому числі використання першого методу, полягає в реалізації схем ГТП і аналізатора спільно з функціональним ядром цифрового пристрою. При цьому мінімальна кількість додаткових зовнішніх виводів може дорівнювати двом: вхід ініціалізації процедури самотестування і вихід результату тестування (рис. 4.1).

Ще одним важливим аспектом проектування контролепридатних цифрових пристроїв є реалізація можливості

окремого тестування комбінаційних схем і схем пам'яті пристрою. У зв'язку з цим довільний цифровий пристрій може бути подано як сукупність всіх комбінаційних підсхем, об'єднаних у єдину комбінаційну схему спільно з множиною елементів пам'яті, які утворюють послідовні підсхеми пристрою.

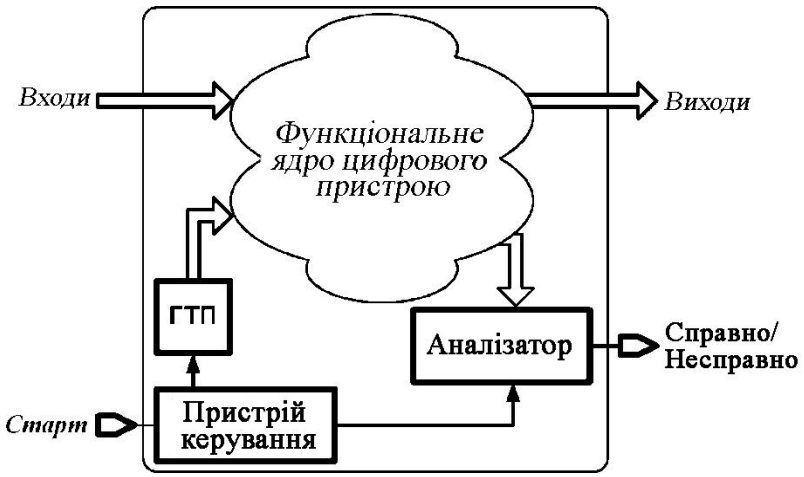



Рис. 4.1. Узагальнена схема вбудованого самотестування

Проектування самотестованих цифрових пристроїв ґрунтується на застосуванні вбудованих засобів для формування тестових послідовностей і аналізу реакцій, що формуються на проміжних полюсах і виходах цифрових пристроїв. Процедура аналізу здійснюється шляхом стискання реакцій пристрою в короткі ключові слова з подальшим їх порівнянням з еталонними значеннями. Ідея самотестування базується на таких основних принципах [86, 87]:

1) генератор тестових послідовностей вбудовується в тестований цифровий пристрій;

2) вихідні реакції на тестові впливи зберігаються тестованим цифровим пристроєм у вигляді компактних характеристик: ключових слів (сигнатур, синдромів, контрольних сум тощо);

3) проведення процедури самотестування полягає тільки в ініціалізації тестування і аналізі його результату;

4) для побудови самотестованих цифрових пристроїв необхідно використовувати мінімальну кількість додаткових зовнішніх виводів пристрою та незначний обсяг додаткової апаратури.

У наш час застосовуються два основних підходи для проектування самотестованих цифрових пристроїв. Перший з них заснований на використанні універсальних модулів для побудови генераторів тестів і аналізаторів вихідних реакцій. Найчастіше як універсальний модуль використовується вбудований блок для логічного аналізу BILBO (Built-In Logic Block Observer) [88]. У цьому випадку як вбудований генератор тестових впливів використовується генератор псевдовипадкових тестових послідовностей (ГПТП), який організовується в одному з режимів блоком BILBO. ГПТП блока BILBO складається з регістра зсуву і невеликої кількості додаткових логічних елементів. Зв'язки між розрядами регістра зсуву і додатковими елементами визначаються примітивним неприведеним поліномом $\varphi(X)$ і є фіксованими для конкретного блока BILBO. ГПТП формує псевдовипадкові тестові набори, що подаються на входи цифрової схеми, на виходах і проміжних полюсах якої формуються реакції на тестові впливи. Очевидно, застосування універсального модуля BILBO зумовлює формування однієї і тієї самої тестової послідовності для всіх самотестуємих пристроїв, незалежно від їхньої архітектури. Введення декількох режимів генератора тестів лише незначно розширює функціональні можливості блока BILBO і помітно ускладнює його реалізацію.

Для отримання компактних оцінок результату самотестування в блоці BILBO використовується спосіб стискання вихідних реакцій у короткі сигнатури. У цьому випадку блок BILBO перетворюється в багатоканальний сигнатурний аналізатор (БСА), який так само, як і ГПТП, описується примітивним поліномом $\varphi(X)$. Будь-яка відмінність сигнатури від її очікуваного значення буде свідчити про наявність несправності в цифровому пристрої, а її збіг з попередньо отриманою сигнатурою показує, що з дуже великою часткою ймовірності пристрій знаходиться в справному стані. Структура БСА, як і ГПТП, є строго фіксованою для конкретного універсального блока BILBO, що знижує його ефективність. З метою усунення визначуваного недоліку першого підходу – низької ефективності побудови самотестованих цифрових пристроїв, що полягає у використанні універсальних блоків, великого поширення отримав другий підхід, заснований на синтезі ГПТП і БСА для кожного самотестованого пристрою.

Процедура синтезу ГПТП і БСА заснована на детальному описі пристрою і обліку всіх його особливостей. Даний підхід відрізняється істотною трудомісткістю, проте він дозволяє забезпечити більш високу ефективність самотестування. Тоді досягається максимальна повнота покриття можливих несправностей цифрового пристрою.

Проектування самотестованих пристроїв полягає не тільки в застосуванні вбудованих генераторів тестів і аналізаторів вихідних реакцій, а також у проектуванні пристрою з урахуванням його тестування в автономному режимі. Для цього широко використовуються методи контролепридатного проектування, що застосовуються в практиці побудови сучасних цифрових пристроїв. Незважаючи на велике різноманіття методів контролепридатного проектування, найбільшу застосовність знаходять методи проектування послідовних цифрових схем, які використовують ідею сканування станів елементів пам'яті [89]. Застосування даного підходу дозволяє ефективно використовувати в самотестованих пристроях схеми ГПТП і БСА. Їх практична реалізація вимагає незначного обсягу додаткової апаратури, оскільки як елементи пам'яті ГПТП і БСА можуть бути використані елементи пам'яті єдиного регістра зсуву ланцюга сканування.

## 4.1. Методи і засоби контролепридатного доступу

Серед усього наявного різноманіття методів контролепридатного доступу [90] слід виділити такі:

1. *Метод сканування елементів пам'яті.* Сутність цього методу полягає в додаванні до структуриу цифрового пристрою додаткових мультиплексорів, що дозволяють перемикати входи елементів пам'яті пристрою. Таким чином, у режимі тестування мультиплексори об'єднують всі елементи пам'яті пристрою в єдиний регістр зсуву, що дозволяє досягти додаткових можливостей при його тестуванні. Усі елементи пам'яті при цьому тестуються ізольовано від комбінаційної частини. Подальший стан пристрою може бути встановлений незалежно від його поточного стану. Крім того, вихідні значення комбінаційної частини пристрою, що надходять на елементи

пам'яті, можуть бути легко спостережувані шляхом сканування станів елементів пам'яті.

2. *Метод наскрізного зсувного регістра.* Даний метод носить назву LSSD (Level-Sensitive Scan Design) і є стандартною технікою проектування цифрових пристроїв, запропонованою фірмою IBM [91]. Відповідно до даного методу кожен системний елемент пам'яті замінюється двовходовим тригером L1 і одновходовим тригером L2, які в режимі тестування об'єднуються в єдиний зсувний регістр. Тригер L2 застосовується, як правило, тільки в процесі тестування, а тригер L1 призначений як для роботи в системі, так і проведення тестового діагностування.

3. *Метод довільного сканування (Random Access Scan Technique).* Цей метод є одним із різновидів методу сканування станів елементів пам'яті. Подібно до класичної схеми методів сканування він також дозволяє порівняно просто встановлювати кожен елемент пам'яті в потрібний стан і спостерігати його на зовнішніх контактах цифрового пристрою. Відмінністю є використання для цього елементів пам'яті з довільною адресацією, що дозволяє незалежно встановлювати, скидати або аналізувати стан будь-якого з них.

4. *Метод сканування граничних елементів пам'яті.* Подальшим розвитком ідеї сканування станів елементів пам'яті є метод проектування цифрових схем, реалізованих за архітектурою сканування граничних елементів пам'яті (Boundary-Scan Architecture) [92]. Основна ідея методу граничного сканування полягає в утворенні в режимі тестування зсувного регістра з елементів пам'яті цифрового пристрою, розташованих між її зовнішніми контактами і функціональним ядром.

З огляду на широке застосування не тільки для тестування, а й вирішення завдань внутрішньосистемного програмування вбудованих систем розглянемо докладніше метод сканування граничних елементів пам'яті.

## 4.2. Метод сканування граничних елементів пам'яті

Приклад ідеї реалізації граничного сканування для цифрового пристрою наведено на рис. 4.2, де всі елементи,

пов'язані з вхідними контактами $x_1$, $x_2$, $x_3$, …, $x_n$ і вихідними $y_1$, $y_2$, $y_3$, …, $y_m$, об'єднані в єдиний регістр зсуву.

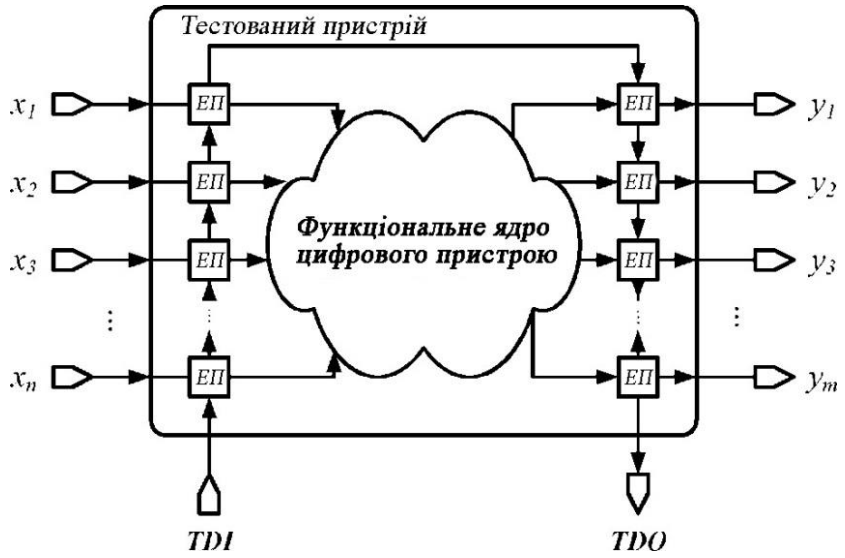



Рис. 4.2. Структура цифрового пристрою за методом сканування граничних елементів пам'яті

Ланцюг сканування регістра зсуву функціонує тільки в режимі тестування, входом якої є контакт TDI (Test Data Input), а виходом – *TDO* (Test Data Output). Для організації регістра зсуву і здійснення мікрооперації зсуву для нього необхідні як мінімум ще два додаткових входи цифрового пристрою. Наявність ланцюга сканування дозволяє в процесі тестування записувати і зчитувати в послідовному режимі тестову інформацію. У загальному випадку застосування методу граничного сканування полягає в реалізації JTAG-архітектури (Joint Test Action Group) [93], що дозволяє мати доступ і управляти засобами підтримки тестування цифрового пристрою. Загальна структура цифрової компоненти, побудованої за архітектурою JTAG, подана на рис. 4.3.

Елементи пам'яті *C0, C1, C2, C3* і *C4* утворюють послідовний шлях сканування, який носить назву регістра граничного сканування (BSR, Boundary Scan Register). У режимі нормального функціонування цифрової компоненти BSR-регістр безперешкодно транслює значення сигналів від зовнішніх контактів до функціонального ядра і назад.

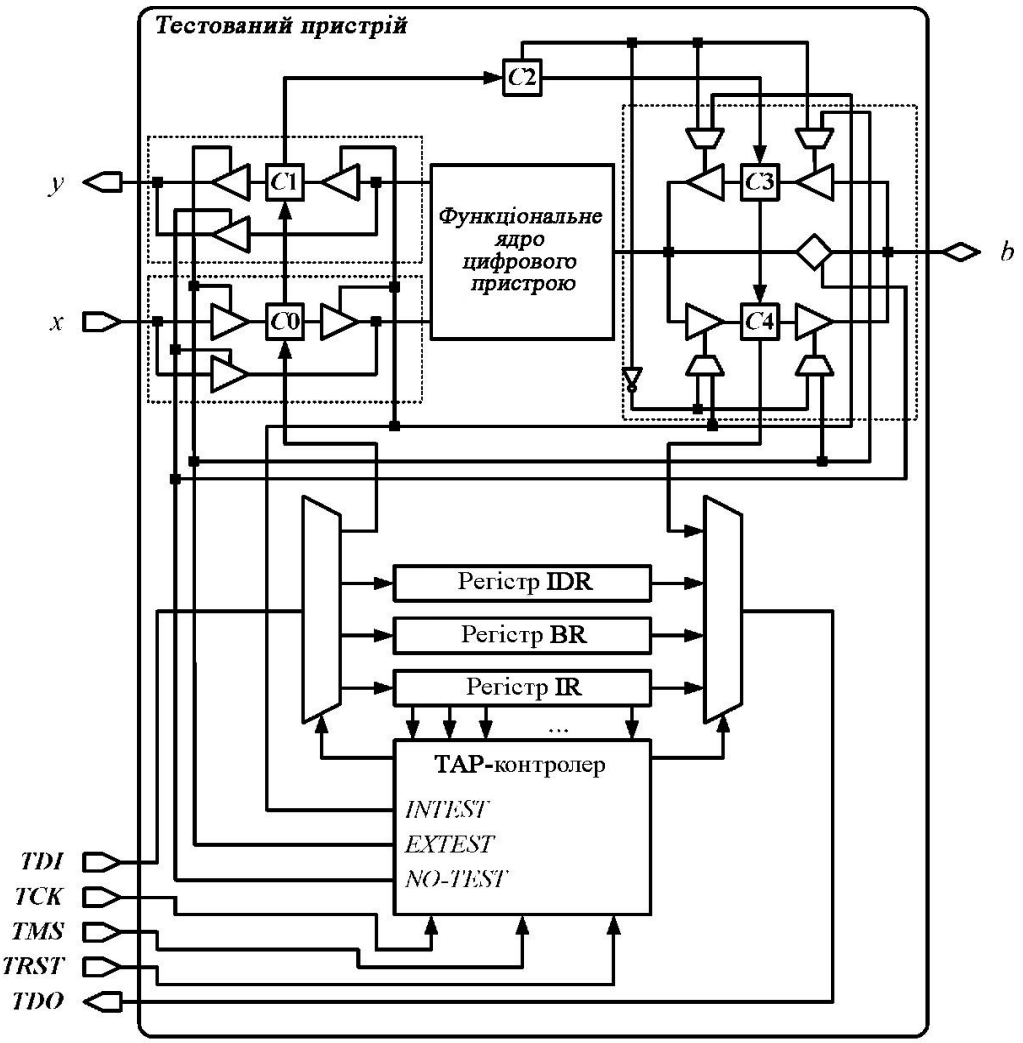



Рис. 4.3. Структура JTAG-сумісного цифрового пристрою

### 4.2.1. Режими функціонування JTAG-пристроїв

Основним елементом JTAG-архітектури є порт тестування Test Access Port (TAP), який складається з TAP-контролера і обов'язкових чотирьох контактів, за допомогою яких можна управляти граничним ланцюгом сканування. TAP-контролер може встановлювати BSR-регістр у два режими: зовнішнього (ETM, External Test Mode) і внутрішнього тестування (ITM, Internal Test Mode).

У режимі ETM функціональне ядро цифрового пристрою відключається від зовнішніх контактів, при цьому значення на вихідних полюсах y і b визначаються запам’ятовуючими елементами C1 і C4, а елементи C0 і C3 здатні захоплювати значення сигналів, що надходять, на вхідних контактах x і b .

Режим ITM також розуміє відключення функціонального ядра пристрою від його зовнішніх контактів, але при цьому значення вихідних сигналів пристрою захоплюються і записуються на елементи пам'яті C1 і C4 регістра BSR, а значення вхідних сигналів для функціонального ядра визначаються станами елементів C0 і C3.

Елемент пам'яті C2 BSR-регістра фактично не пов'язаний з будь-яким зовнішнім контактом цифрового пристрою. Значення, яке зберігає C2, використовується для управління елементами пам'яті, підключеними до двонаправленого контакту b. Залежно від значення, що зберігається, контакт b може розглядатися як вхід, при цьому буде задіяний елемент C3, або як вихід, який пов'язаний з елементом C4 регістра граничного сканування.

На рис. 4.4 схематично зображено два режими BSR-регістру: ETM (контакт b використовується як вхід) і ITM (контакт b використовується як вихід).

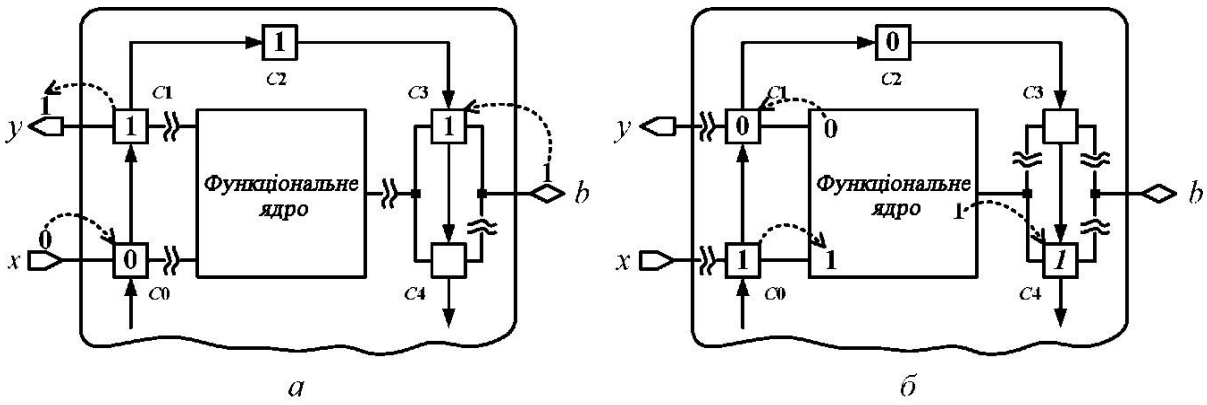



Рис. 4.4. Режими BSR-регістра: а – режим ETM; б – режим ITM

### 4.2.2. Структура і функціонування TAP-контролера

TAP-контролер, керуючий BSR-регістром, має п'ять інтерфейсних ліній, підключених до зовнішніх виводів цифрового пристрою.

Лінія *TRST* (Test ReSeT) передає асинхронний сигнал ініціалізації TAP-контролера (є необов'язковою відповідно до стандарту IEEE 1149-1).

Лінія *TCK* (Test ClocK) передає вхідний сигнал синхронізації для TAP-контролера. Синхронізація контролера

здійснюється за переднім фронтом сигналу з частотою, що, як правило, не перевищує 10 МГц.

Лінія *TMS* (Test Mode Select) передає керуючий сигнал, який визначає новий стан TAP-контролера.

Лінія *TDI* (Test Data Input) є послідовним каналом передачі даних від зовнішніх пристроїв до молодшого розряду регістра BSR або спеціальних регістрів TAP-контролера.

Лінія TDO (Test Data Output) є послідовним каналом передачі даних від старшого розряду BSR-регістра або спеціальних регістрів TAP-контролера до зовнішнього обладнання.

За наявності в обчислювальній системі кількох цифрових компонент, що підтримують JTAG-архітектуру, доцільно проводити об'єднання їхніх BSR-регістрів у єдиний ланцюг сканування JTAG (рис. 4.5).

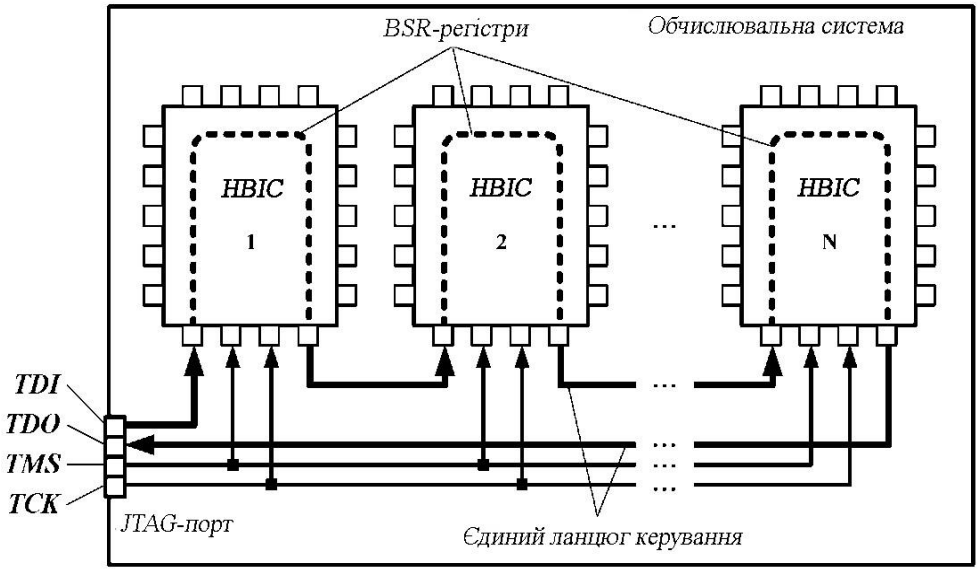



Рис. 4.5. Приклад формування єдиного ланцюга сканування JTAG

При цьому виводи TMS і TCK всіх JTAG-сумісних пристроїв паралельно підключаються до загальних ліній TMS і TCK відповідно. Контакт TDI JTAG-порту обчислювальної системи підключається до входу TDI TAP-порту першої цифрової компоненти, вихід TDO якої підключається до входу TDI наступного компонента тощо. Вихід TDO останнього JTAG-сумісного пристрою підключається до контакту TDO JTAG-порту, утворюючи тим самим єдиний ланцюг граничного сканування.

Управління BSR-регістром і спеціалізованими JTAG-регістрами організується TAP-контролером, який являє собою цифровий кінцевий автомат з 16 станами. Діаграма станів TAP-контролера наведена на рис. 4.6.

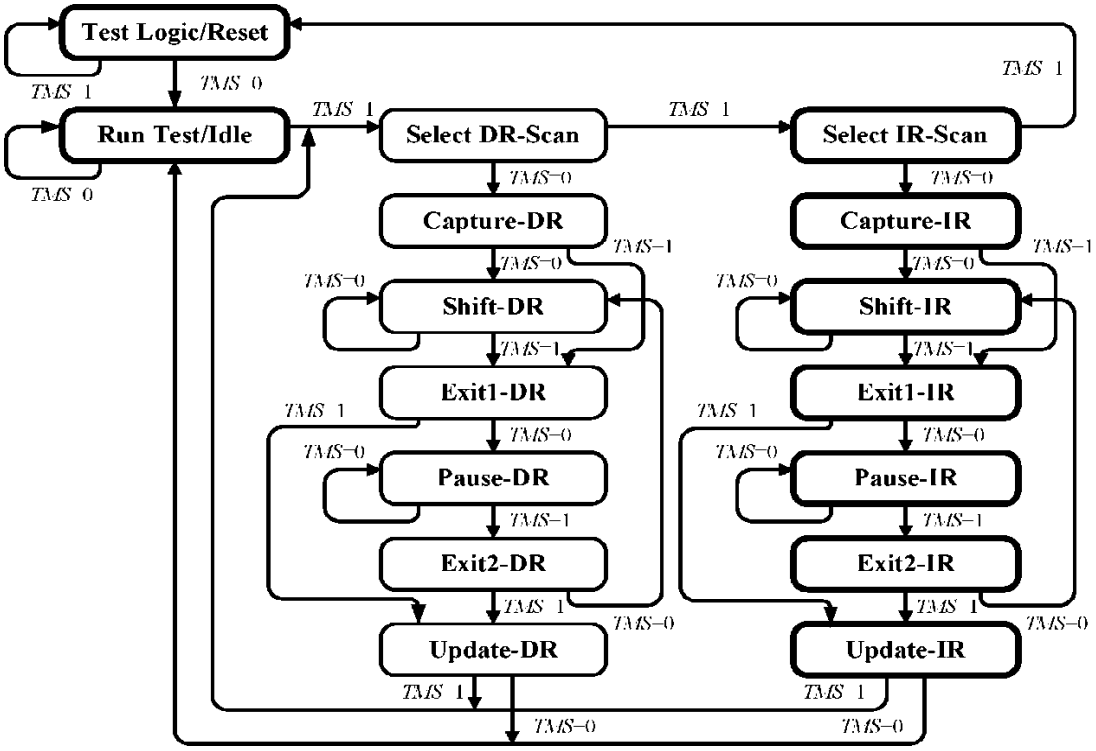



Рис. 4.6. Діаграма станів TAP-контролера

Згідно з діаграмою з кожного стану може бути здійснено два переходи, контрольованих одним сигналом TMS. Можна виділити дві основні частини діаграми станів: послідовність станів сканування регістрів даних (IDR, BR і BSR) і послідовність станів сканування регістра інструкцій IR. Яким саме регістром даних буде управляти TAP-контролер, залежить від поточного значення регістра IR. Для завантаження нової інструкції в регістр IR використовується така послідовність станів TAP-контролера:

-> *TEST LOGIC/RESET ;* ініціалізація TAP - контролера

-> *RUN TEST/IDLE ;* стан очікування

-> *SELECT DR-SCAN ;* вибір процедури сканування регістрів даних

*-> SELECT IR-SCAN ;* вибір процедури сканування регістра IR

*-> CAPTURE-IR ;* захоплення старого значення регістра IR

*-> SHIFT-IR->...; зсув регістра IR, повторюється в залежності від довжини інструкції, при цьому нове значення інструкції побітно подається на вхід TDI, а старе значення висувається на вихід TDO*

*-> EXIT1-IR ; вихід з процедури сканування регістра IR*

*-> UPDATE-IR ; оновлення нового значення в регістрі IR*

*-> RUN TEST/IDLE; стан очікування*

Зміна стану TAP-контролера відбувається за переднім фронтом синхроімпульсу TCK, якому передує установлення певного значення сигналу TMS. Послідовність станів, необхідних для запису нового значення в один з регістрів даних, виглядає так:

*-> TEST LOGIC/RESET; ініціалізація TAP-контролера*

*-> RUN TEST/IDLE; стан очікування*

*-> SELECT DR-SCAN; вибір процедури сканування регістрів даних*

*-> CAPTURE-DR ; захоплення старого значення регістра даних*

*-> SHIFT-DR -> ...; зсув регістра даних, повторюється залежно від його довжини, при цьому нове значення побітно подається на вхід TDI, а старе значення висувається на вихід TDO*

*-> EXIT1-DR ; вихід з процедури сканування регістра даних*

*-> UPDATE-DR ; оновлення нового значення в регістрі даних*

*-> RUN TEST/IDLE; стан очікування*

У всіх реалізаціях JTAG-архітектура, відповідно до стандарту IEEE 1149-1 присутні три регістри даних [93]:

1. IDR (ID Register) – регістр, який зберігає унікальний ідентифікатор цифрового пристрою.

2. BR (Bypass Register) – однорозрядний регістр обходу, який з'єднує між собою вхід TDI і вихід TDO. Передача даних при цьому затримується на один такт синхроімпульсу TCK. Регістр BR використовується для обходу однієї або декількох цифрових компонент обчислювальної системи, об'єднаних у єдиний ланцюг сканування JTAG.

3. BSR (Boundary Scan Register) – безпосередньо ланцюг граничного сканування цифрової компоненти.

Для проведення тестового сеансу за допомогою граничного ланцюга сканування JTAG спочатку необхідно завантажити відповідну інструкцію в регістр IR, а потім завантажити/вивантажити значення одного з регістрів даних. Стандартом IEEE 1149-1 закріплені так звані обов'язкові інструкції, які мають бути присутніми у всіх реалізаціях JTAG сумісних цифрових пристроїв [93]. Перерахуємо ці інструкції:

1. **BYPASS:** задавання режиму обходу поточного ланцюга граничного сканування через регістр BR.

2. **EXTEST:** як регістр даних обирається BSR, який підключається до TDI- і TDO-виводів. У стані CAPTURE-DR зовнішні контакти цифрової компоненти управляються розрядами BSR-регістра: значення сигналів на входах записуються в елементи пам'яті BSR, а значення на виходах визначаються станами відповідних розрядів BSR. При виході TAP-контролера зі стану SHIFT-DR старе захоплене значення BSR-регістра послідовно зсувається на вихід TDO. Нове значення BSR-регістра послідовно надходить з виводу TDI при вході TAP-контролера в стан SHIFT-DR.

3. **IDCODE:** як регістр даних обираються IDR-регістри, які підключаються до зовнішніх виводів TDI і TDO. При досягненні TAP-контролера стану CAPTURE-DR значення унікального ідентифікатора цифрової компоненти (як правило, це код виробника, номер партії, версія пристрою тощо) паралельно записується в регістр IDR. При кожному виході контролера зі стану SHIFT-DR значення ідентифікатора послідовно зсувається на вихід TDO.

4. **INTEST:** при виконанні цієї інструкції BSR-регістр підключається до зовнішніх контактів TDI і TDO. При вході контролера в стан CAPTURE- DR значення вихідних сигналів функціонального ядра цифрової компоненти записуються в відповідні розряди BSR-регістра. При кожному виході контролера зі стану SHIFT-DR значення BSR-регістра зсувається на вихід TDO, при цьому нове значення BSR-регістра послідовно надходить зі входу TDI на початок стану SHIFT-DR. Нове значення BSR-регістра приймається функціональним ядром при досягненні стану UPDATE-DR.

### 4.2.3. Запам'ятовуючі елементи JTAG-регістрів

Розглянемо докладніше внутрішню структуру запам'ятовуючого елемента, на основі якого можуть бути побудовані всі регістри даних JTAG-архітектури. Так, комірка регістра граничного сканування BSC (Boundary Scan Cell) складається з двох елементів пам'яті і має такі виводи (рис. 4.7):

SI (Serial In) – вхід послідовних даних, що надходять від попередньої BSC-комірки;

PI (Parallel In) – вхід паралельних даних, прийнятих від функціонального ядра цифрового пристрою;

SO (Serial Output) – вихід послідовних даних, які надходять на SI-вхід подальшої BSC-комірки;

MODE (Test/Normal) – керуючий вхід, що визначає режим BSC-комірки;

*CLOCKS/CONTROLS* – входи синхронізації і управління, що визначають функціонування BSC-комірки.

Відповідно до наведеного функціонування регістра граничного сканування визначимо поведінковий VHDL-опис BSC-комірки (дод. 3, лістинг 4.1).

Відповідно до стандарту IEEE 1149-1 BSC-комірки стробується двома роздільними сигналами: *Clock-DR* і *Update-DR*. Фронт першого сигналу слід безпосередньо перед фронтом сигналу TCK, а фронт *Update-DR* – перед спадом синхросигналу TCK. Слід зазначити, що в деяких реалізаціях JTAG-архітектура BSC-комірки проектується з використанням двох синхронних D-тригерів.

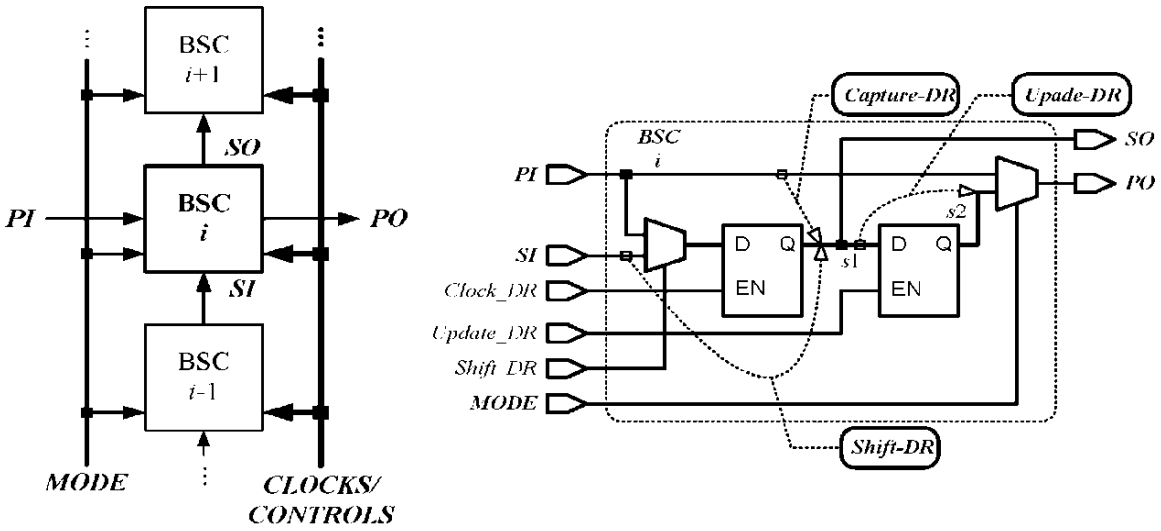



Рис. 4.7. Структура елемента пам'яті регістра граничного сканування

Структура комірки регістра обходу наведена на рис. 4.8 і містить три входи: *SI* (за допомогою TAP-контролера з'єднується з TDI-входом JTAG-компоненти), *Shift-DR* (керуючий сигнал TAP-контролера, що забезпечує мікрооперацію зсуву регістра даних) і *Clock-DR* (стробувальний сигнал, що забезпечує коректне функціонування BR-регістра). Єдиний вихід *SO*-регістра обходу за допомогою TAP-контролера комутується на контакт TDO JTAG-компоненти. На рис. 4.9 зображена логічна схема комірки регістра інструкцій ШС (Instruction Register Cell).

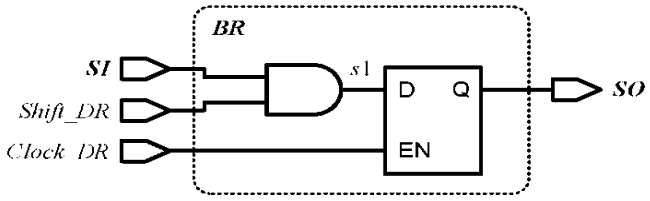



Рис. 4.8. Структура регістра обходу BR і його VHDL-опис

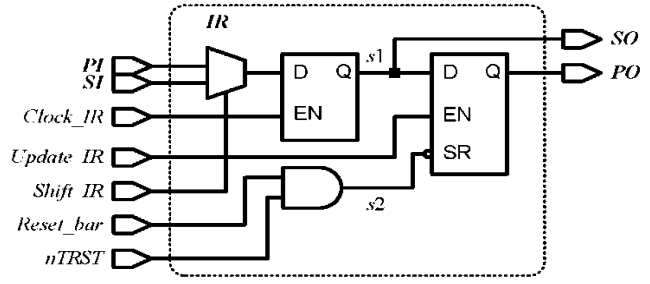



Рис. 4.9. Структура IRC-комірки

```
Entity BR is
    Port ( SI : in bit;
        Shift_DR : in bit;
        Clock_DR : in bit;
        SO : out bit );
End BR;
Architecture Behavioral of BR is
signal s1: bit;
Begin
s1 <= Shift_Dr and SI;

MAIN: process( SI, Clock_DR, s1 )
begin
                if Clock_DR = '1' then
         SO <= s1;
    end if;
end process;
End Behavioral;
```

Імена вхідних і вихідних сигналів IRC схожі з іменами і призначеннями сигналів BSC. D-засувка стробується сигналом *Update-IR* і може бути встановлена в середнє арифметичне значення (SR = «1») або скинута в нульове значення (SR = «0») одним з двох асинхронних сигналів: *Reset-bar* або *nTRST*. На входи паралельних даних PI регістра IR подається двійкове значення, яке визначається розробником цифрової компоненти. Виняток становлять два молодших розряди IR, на які подаються фіксовані значення 0 і 1 відповідно. Виходи паралельних даних *PO* підключені до входів декодера інструкцій TAP-контролера. Відповідні виводи SI і SO IRC-комірок об'єднані між собою, безпосередньо утворюючи зсувний регістр IR. Наступний приклад ілюструє один з можливих варіантів опису IRC-комірок мовою VHDL.

Подані VHDL-описи (дод. 3, лістинг *4.2)* запам'ятовуючих елементів можуть бути використані для подальшого проектування JTAG-сумісних цифрових компонент.

## 4.3. Сфери застосування апаратури граничного сканування

Перша версія стандарту IEEE 1149-1 була розроблена групою JTAG (Joint Test Action Group), первинною метою якої було створення промислового стандарту для методу тестування з'єднувальних ліній цифрових компонент, розташованих на друкованих платах. Гнучкі можливості JTAG-архітектура, які забезпечують високий ступінь спостереження та управління для цифрових компонент, знайшли своє застосування і в інших сферах. Так, за допомогою JTAG може бути здійснено тестування периферійних контактів, організація самотестування НВІС, тестування НВІС ОЗП, тестування цифрових компонент, які не підтримують JTAG-архітектуру (рис. 4.10).

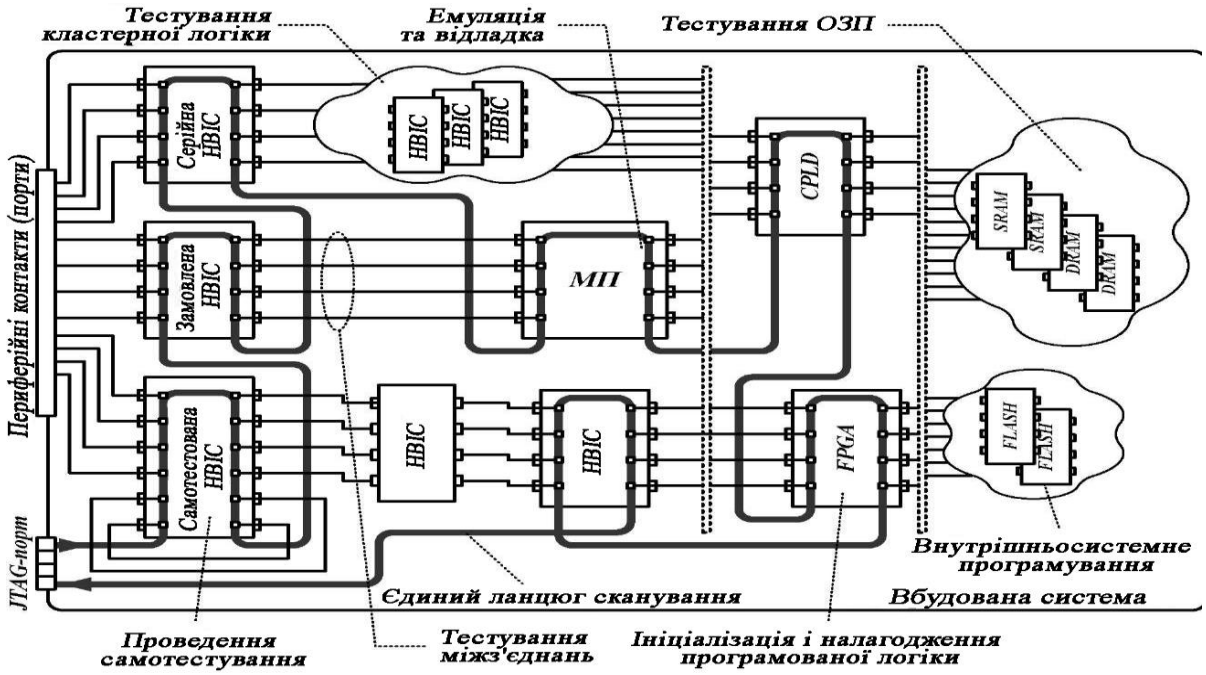



Рис. 4.10. Варіанти застосування IEEE 1149-1 стандарту для вбудованих систем

Зовнішній доступ до контактів цифрових компонент вбудованих систем за допомогою стандарту JTAG дозволяє вирішувати такі завдання, як програмування Flash-ПЗП, не використовуючи спеціалізовані фізичні інтерфейси і обладнання. Програмування таких НВІС, як FPGA і CPLD, також може бути здійснено за допомогою JTAG. Безпосередній і непрямий доступ

до фізичних контактів цифрових компонент дозволяють використовувати JTAG-інтерфейс для емуляції, верифікації та налагодження таких апаратних засобів, як мікропроцесори, мікроконтролери і НВІС програмованої логіки.

## 4.4. Проектування JTAG-сумісного цифрового пристрою

Розглянемо повний цикл проектування JTAG-сумісного цифрового пристрою з використанням мови VHDL. У попередньому підрозділі був детально розглянутий механізм сканування граничних елементів пам'яті з VHDL-описом базових елементів основних регістрів JTAG -архітектури.

На основі наявних описів базових компонент складемо описи всіх необхідних регістрів JTAG-архітектури, спроектуємо пристрій управління у вигляді TAP-контролера і впровадимо апаратуру граничного сканування в проектний опис цифрового пристрою.

### 4.4.1. Синтез регістра граничного сканування

Почнемо проектування з регістра граничного сканування (Boundary-Scan Register) BSR, опис якого побудуємо на основі наявного VHDL-опису елемента пам'яті BSC (дод. 3, лістинг 4.1). Регістр BSR є зсувним регістром, елементи пам'яті якого з'єднані наскрізною лінією перенесення, що з'єднує вихід SO послідовних даних одного елемента зі входом SI послідовних даних наступного елемента. Крім того, регістр BSR має вхідну шину паралельних даних PI і вихідну шину паралельних даних PO. Сигнальні лінії цих шин підключені до відповідних входів PI і виходів PO кожного елемента пам'яті BSC. До всіх елементів пам'яті підключені лінії управління станом регістра BSR. Лістинг 4.3 (дод. 3) містить структурний опис регістра граничного сканування BSR, складений на основі VHDL-опису компоненти BSC.

На рис. 4.11 і 4.12 подано результати синтезу одного елемента пам'яті BSC і всього регістра граничного сканування для значення налаштованого параметра *length* = 3. Результати були отримані за допомогою програмної системи логічного синтезу Synplify Pro [94].

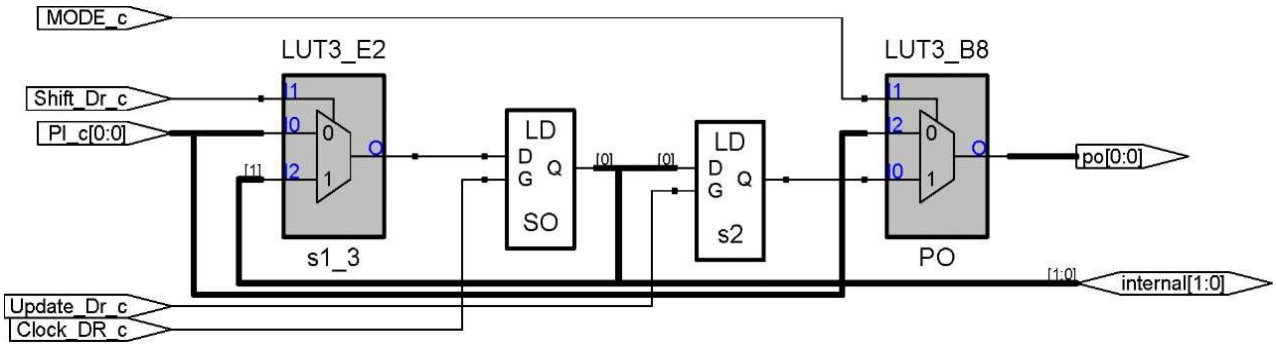



Рис. 4.11. Результат синтезу елемента BSC

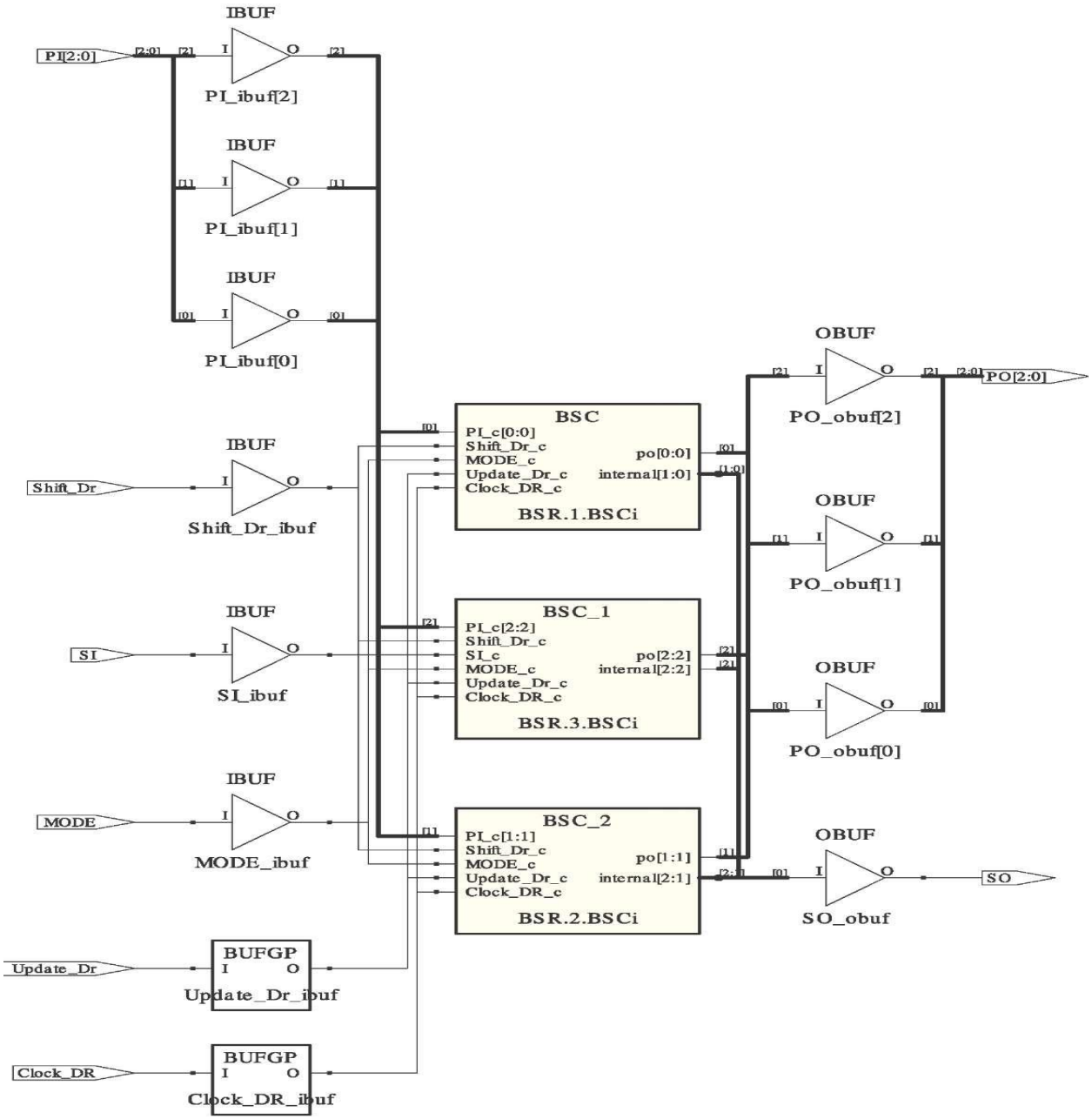



Рис. 4.12. Результат синтезу регістра BSR

### 4.4.2. Синтез регістра інструкцій

Аналогічно спроектуємо регістр інструкцій IR на основі наявного VHDL-опису компоненти IRC з дод. 3 (лістинг 4.2). Нехай регістр IR складається з чотирьох елементів IRC (цієї кількості вистачить для кодування чотирьох стандартних JTAG-інструкцій), при цьому на входи PI двох молодших елементів подається фіксоване значення «01», а при ініціалізації тіньовий регістр IR набуває значення «1111» (дод. 3, лістинг 4.4).

На рис. 4.13 показано результат RTL-синтезу 4-розрядного регістра IR і синтезовану структуру одного елемента пам'яті IRC.

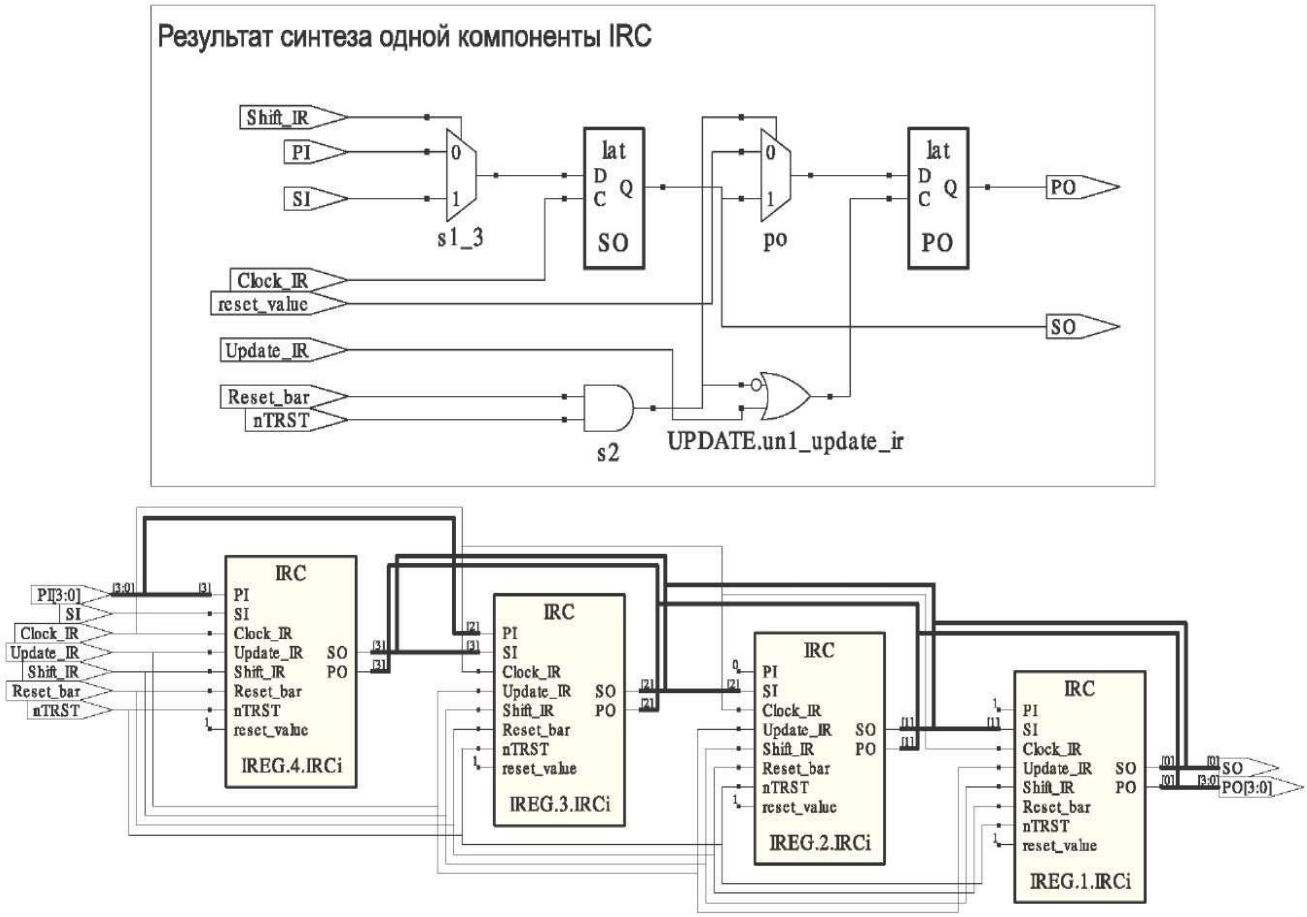



Рис. 4.13. Результати синтезу елемента IRC і регістра IR

### 4.4.3. Синтез TAP-контролера

Після того як всі регістри спроектовано (регістр обходу BR був описаний у попередньому підрозділі), можна приступати до розроблення їхнього пристрою управління. Почнемо зі схеми

декодера JTAG-інструкцій, який залежно від записаної команди в регістр IR виробляє відповідні кожному регістру керуючі і синхронізуючі сигнали. Зумовимо використання тільки обов'язкових JTAG-інструкцій: EXTEST, SAMPLE_PRELOAD, IDCODE і BYPASS .

У дод. 3 (лістинг 4.5) подано поведінковий VHDL-опис декодера JTAG-інструкцій.

Основним ядром пристрою управління JTAG-регістрами є цифровий кінцевий TAP-автомат, який функціонує відповідно до діаграми станів, поданої раніше на рис. 4.6. Спроектуємо TAP-автомат за допомогою двох компонент: синхронного пристрою зміни поточного стану і пристрою, який буде виробляти керуючі сигнали TAP-автомата. У дод. 3 (лістинг 4.6) подано поведінковий VHDL-опис алгоритму зміни станів TAP-автомата.

Разом з описом компоненти TAP_FSM подано код VHDL-пакета JP, що містить визначення користувальницького типу TAP_STATE, який у подальшому буде застосований для опису інших компонент.

Наступна проектована компонента TAP_OUT (дод. 3, лістинг 4.7**)** є частиною TAP-автомата, яка відповідає за вироблення керуючих сигналів залежно від поточного стану компоненти TAP_FSM.

Наведені вище описи компонент дозволяють скласти VHDL-опис всього TAP-контролера, який включає до своєї структури TAP-автомат, регістр інструкцій IR, декодер інструкцій і регістр обходу BR. У дод. 3 (лістинг 4.8) наведено змішаний VHDL-опис компоненти TAP-контролера.

### 4.4.4. Впровадження елементів граничного сканування в проект цифрового пристрою

Коли всі компоненти JTAG-архітектури спроектовані, їх можна впроваджувати в готовий проект цифрової компоненти. Для простоти розуміння виберемо цифрову компоненту ASIC, що має чотири зовнішніх порти (дод. 3, лістинг 4.9).

Для впровадження елементів архітектури JTAG у вихідний VHDL-опис компоненти ASIC спроектуємо нову компоненту, яка буде включати безпосередньо вихідний пристрій ASIC, модуль

JTAG-контролера TAP_CTRL і власне регістр граничного сканування BSR (рис. 4.14).

Лістинг 4.10 (дод. 3) містить структурний VHDL-опис JTAG-сумісного пристрою згідно зі схемою, наведеною на рис. 4.14, а на рис. 4.15 подано результат RTL-синтезу описаного пристрою.

Завдяки наявності параметрів, що настроюються в наведених VHDL-описах, побудова JTAG-сумісного пристрою може бути виконана для довільної кількості зовнішніх портів вихідної компоненти.

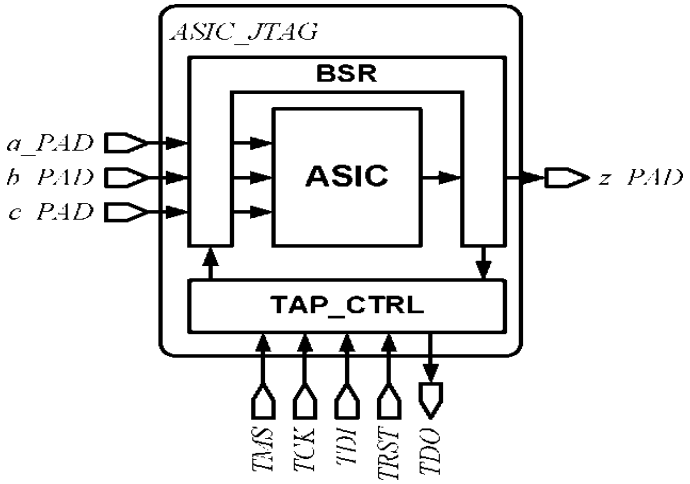



Рис. 4.14. Структура JTAG-сумісного пристрою

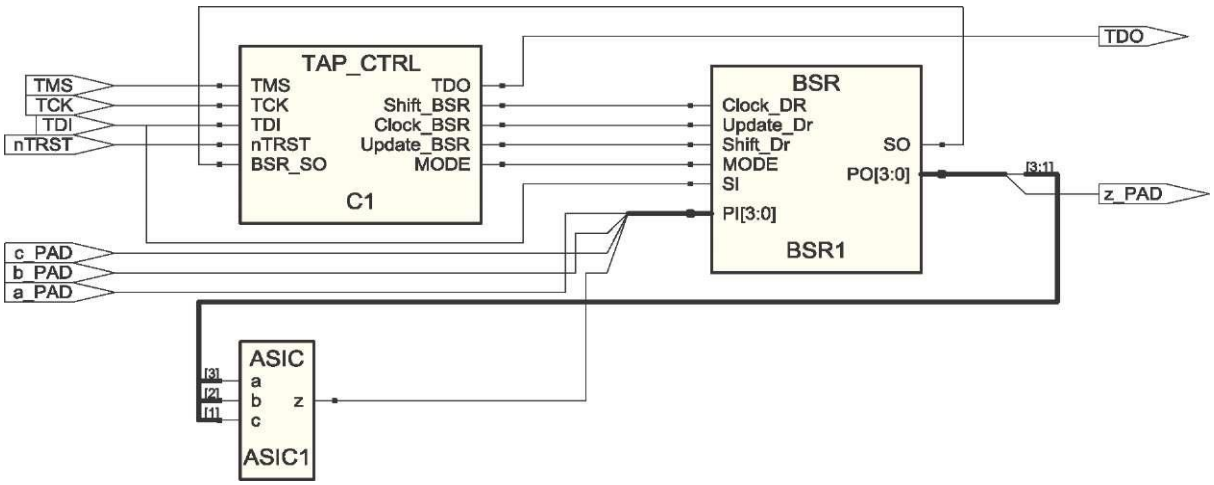



Рис. 4.15. Результат синтезу JTAG-сумісного пристрою

**Контрольні питання**

1. За якими ознаками класіфікуються запам'ятовуючі пристрої?
2. Які типи пам'яті належать до адресних?
3. Які типи пам'яті належать до послідовніх?
4. Які типи пам'яті належать до асоціативніх?
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 маскових 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.

## 5. Етапи проектування вбудованих систем

Крім визначення властивостей надійного функціонування цифрових пристроїв за наявності фізичних дефектів і зовнішніх несприятливих факторів, важливим є захист цифрових проектів і реалізованих пристроїв від несанкціонованих дій людини. До подібного роду дій можна віднести, наприклад, нелегальне використання програмних IP-компонент, поданих у вигляді HDL-описів.

При реалізації цифрових пристроїв на базі ПЛІС проблема захисту від несанкціонованих дій людини стає масштабнішою. Виражається це в першу чергу в тому, що і HDL-коди і BIT-образи конфігурації ПЛІС передаються і поширюються у відкритому вигляді.

На рис. 5.1 подана загальна схема взаємодії виробника і користувача цифрової системи з зазначенням об'єктів користування. Проектований цифровий пристрій на боці розробника знаходиться в довірчому оточенні, до якого входять система автоматизованого проектування, що включає програмні засоби створення та редагування проектних HDL-описів, програмні засоби синтезу (генерування BIT-образу), цифрова система на базі ПЛІС і пам'ять конфігурації PROM.

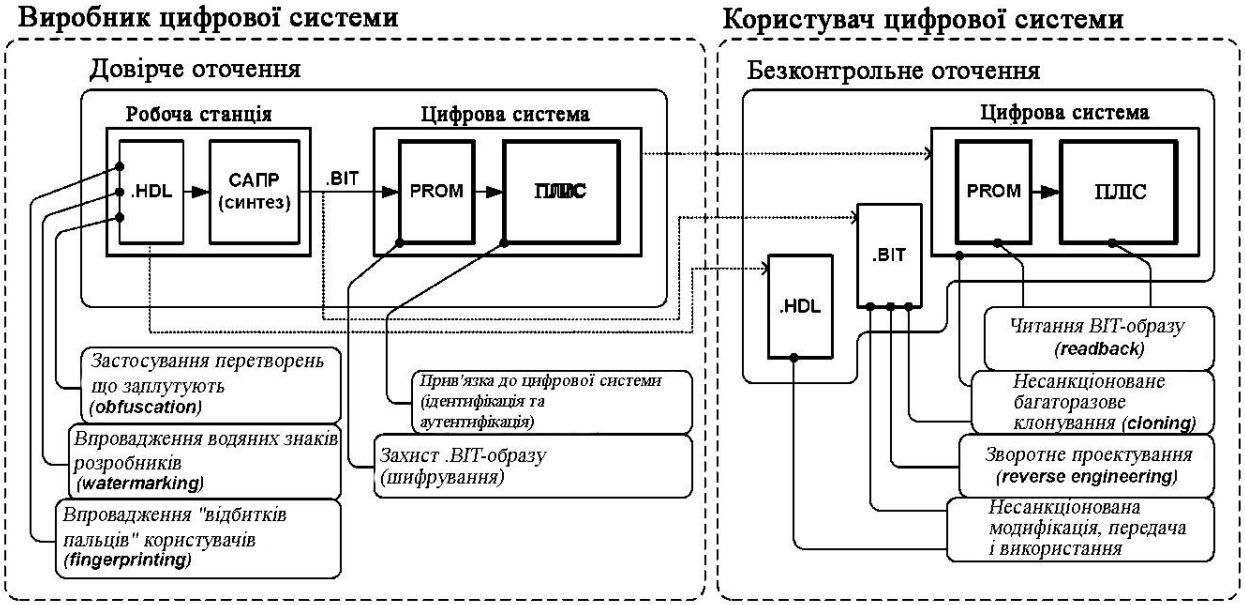



Рис. 5.1. Схема взаємодії виробника і користувача

У довірчому оточенні створюються об'єкти інтелектуальної власності, до яких належать файли проектних HDL-описів, файл BIT-образу конфігурації ПЛІС, конфігурована цифрова система (BIT-образ записаний у PROM). Перераховані об'єкти можуть передаватися користувачу (разом або окремо) залежно від угоди з виробником.

Будучи переданими користувачеві, об'єкти інтелектуальної власності знаходяться в безконтрольному оточенні, що означає наявність можливості несанкціонованих дій відносно них.

До несанкціонованих дій можна віднести такі:

- несанкціонована модифікація, передача або використання HDL-описів (IP-компонент), які потягли за собою порушення авторських прав виробника цифрової системи;
- зворотне проектування BIT-образу (reverse engineering), при якому зловмисник здійснює перетворення файла конфігурації ПЛІС в netlist-опис і/або інший опис, який сприяє розумінню внутрішньої структури і особливостей функціонування цифрової системи;
- багаторазове несанкціоноване використання BIT-образу (cloning), що припускає тиражування цифрової системи в обхід угод з виробником (наприклад виготовлення більш дешевої цифрової системи з ідентичною ПЛІС і використання оригінального BIT-образу для її конфігурації);
- читання BIT-образу (readback) безпосередньо з ПЛІС або PROM конфігурованої цифрової системи для проведення зворотного проектування.

Для запобігання несанкціонованим діям з боку користувача виробник застосовує різні методики. Розглянемо докладніше їхнє призначення і застосування.

## 5.1. Застосування заплутуючих перетворень HDL-описів

Заплутуючі перетворення, або обфускація, – це приведення вихідного тексту або виконуваного коду програми до вигляду, що зберігає її функціональність, але ускладнює аналіз, розуміння алгоритмів роботи і модифікацію при декомпіляції [95]. З точки зору HDL-описів, під обфускацією можна розуміти такі два типи перетворень: лексичну обфускацію і функціональну обфускацію.

Припустимо, що є вихідний HDL-опис V, який шляхом застосування до нього всіх послідовностей проектних дій DD перетворюється в цифрову схему S: $DD(V) \to S$. Застосувавши обфусцувальні перетворення O до V, отримуємо опис $V' : O(V) \to V'$, який у результаті перетворюється в ідентичну схему $S : DD(V') \to S$, при цьому $V'$ має меншу наочність і є більш складним для читання і розуміння навіть професійним проектувальником.

Подібний вид перетворення є лексичною обфускацією і ґрунтується на зміні форматування вихідного HDL-опису, що сприяє ускладненню його зорового сприйняття. Однак по суті всі мовні конструкції при цьому не зазнають модифікацій, що і спричиняє отримання ідентичної схеми S.

До можливих змін формату вихідного опису V можна віднести такі:

– видалення всіх коментарів;

– видалення/додавання символів-роздільників (delimits) між лексемами HDL-опису;

– заміна імен користувальницьких ідентифікаторів на ідентифікатори з довільними/нечитабельними іменами.

Так, видалення коментарів само по собі є першою перешкодою для швидкого розуміння вихідного HDL-опису, а форматування шляхом видалення таких роздільників, як кінець рядка (EOL), робить опис несприйнятливим до читання. При складанні HDL-описів проектувальники часто використовують імена ідентифікаторів, які несуть деяке смислове навантаження з точки зору їх призначення. Наприклад, сигнали синхронізації часто оголошуються як CLK, clock, sync тощо, що вже є достатньою інформацією для розуміння їх призначення. Заміна вихідних імен ідентифікаторів на менш читабельні ще більше ускладнює як сприйняття, так і розуміння HDL-опису. На рис. 5.2 подано два варіанти VHDL-опису: V – вихідний VHDL-опис і V ', отриманий шляхом обфусцувальних перетворень.

У наведеному прикладі всі призначені для користувача ідентифікатори замінено на кодові слова однакової довжини з застосуванням ASCII-символів, схожих за відображенням ('O' – 0x4F; '0' – 0x30; '1' – 0x31; 'l' – 0x6c).

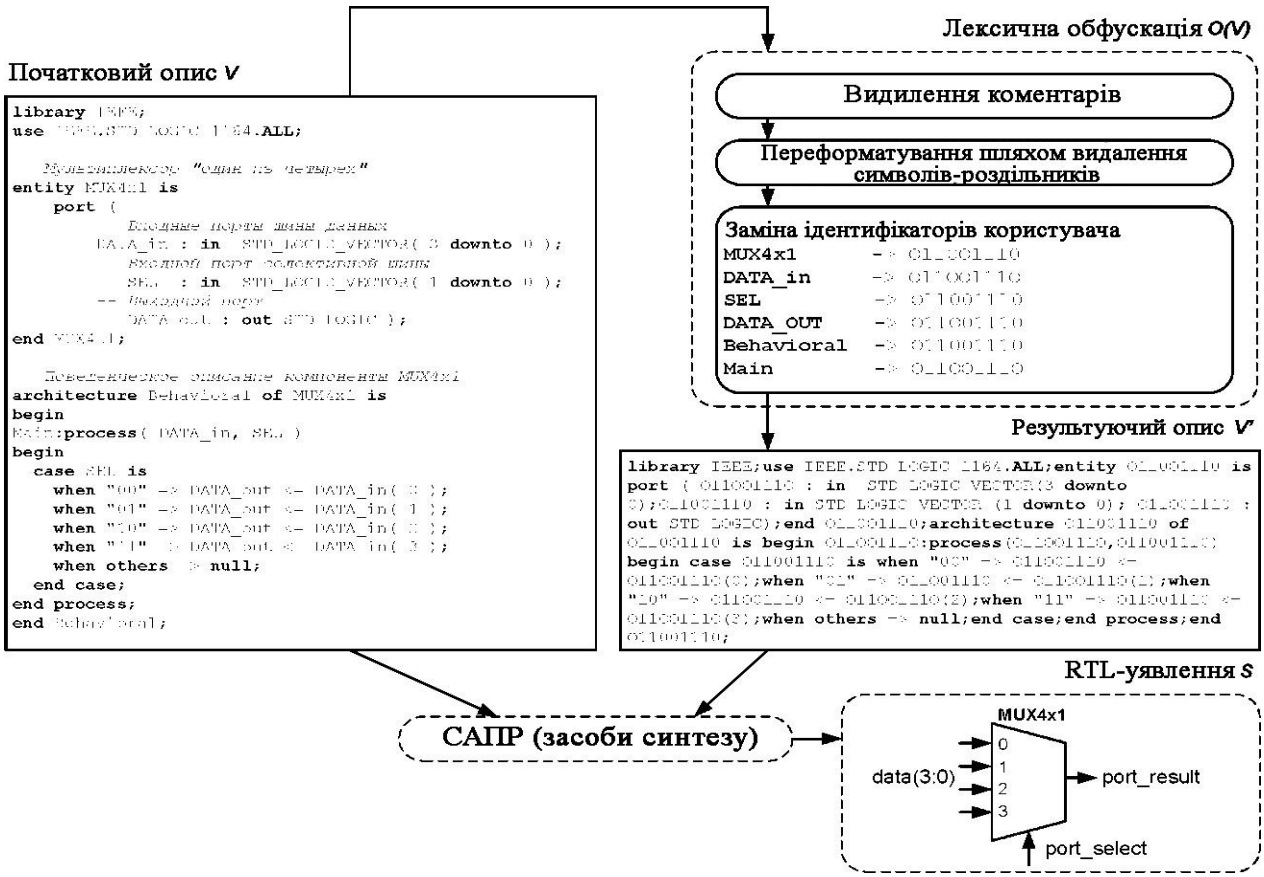



Рис. 5.2. Результат лексичної обфускації

Крім лексичної обфускації, до HDL-описів можна застосовувати обфускацію потоку виконання [95], яка для мов програмування означає перетворення графа передачі управління, що досягається за рахунок локальних перетворень коду або еквівалентних перетворень алгоритму. Наприклад, для мови VHDL запис паралельного оператора присвоювання може бути замінений на структурний опис сторонньої компоненти або використання функції [96]. А властивість паралельності операторів мови VHDL, поданих у тілі блока **architecture**, може бути використано для зміни порядку їх слідування у вихідному описі.

Читання HDL-опису, підданого лексичній обфускації, є досить складним. Однак яким б не був ступінь складності подібних перетворень, у результаті HDL-опис залишається синтезованим, що відкриває можливість зрозуміти структуру і функціональні особливості пристрою за результатами синтезу.

У зв'язку з цим інтерес викликає функціональна обфускація, під якою можна розуміти процес отримання еквівалентної цифрової схеми. Нехай дано вихідний опис V, який регламентує

функціонування і/або структуру цифрового пристрою $S{:}DD(V) \to S$. Перетворимо V таким чином, щоб отриманий після синтезу пристрій S' став функціональним еквівалентом пристрою $S{:}\ DD(O(V)) \to S'$.

Припустимо, що схема цифрового пристрою S описується таким виразом:

$$S = \{IP_0, \dots, IP_{i-1}; OP_0, \dots, OP_{j-1}; B_0, \dots, B_{k-1}; L_0, \dots, L_{m-1}\}, \quad (5.1)$$

де *IP* – множина вхідних портів;

*OP* – множина вихідних портів;

*B* – множина функціональних блоків, з яких складається схема пристрою;

*L* – множина провідних ліній, що з'єднують внутрішні блоки і порти.

Визначимо функціональність пристрою S як залежність значень на вихідних портах *OP* від значень на вхідних портах *IP*, внутрішніх блоків *B* і провідних ліній *L*:

$$OP = F_S(IP, B, L). \quad (5.2)$$

Припустимо, що існує інший цифровий пристрій S', схема якого описується як

$$S' = \{IP_0, \dots, IP_{i-1}; OP_0, \dots, OP_{j-1}; B'_0, \dots, B'_{k-1}; L'_0, \dots, L'_{m-1}\}, \quad (5.3)$$

при цьому множина блоків B' і множина ліній L' може не збігатися з аналогічними множинами пристрою S.

Пристрої S і S' будуть функціонально еквівалентними, якщо виконується така рівність:

$$F_S(IP, B, L) = F_{S'}(IP, B', L'). \quad (5.4)$$

Розглянемо найпростіший приклад функціонально еквівалентних схем. Нехай є схема S логічного елемента NOT (інвертор), яка описується множинами $S = \{IP_0; OP_0; B_0; L_0, L_1\}$ (рис. 5.3).

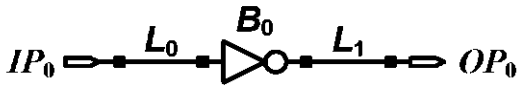



Рис. 5.3. Внутрішня структура схеми S

Функціонування поданої схеми залежить від значення на вхідному порту *IP0*, блока *B0* і ліній *L0* і *L1*: $OP_0 = F_S(IP_0, B_0, L_0, L_1)$.

Схемою, що є функціонально еквівалентною схемі S, може бути схема, що складається з логічного елемента XOR2 і трьох ліній L0, L1 і $L'_2$: $S' = \{IP_0; OP_0; B'_0; L'_0, L'_1, L'_0\}$, за умови що рівень сигналу на L0 постійно набуває значення 1 (рис. 5.4).

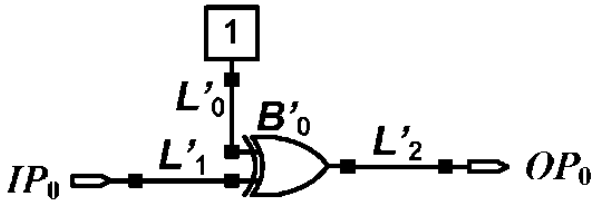



Рис. 5.4. Внутрішня структура схеми S'

Таким чином, існує можливість обфускації вихідного опису V, у результаті застосування якої отримують такий опис V', що $DD(V') \to S', F_{S'} = F_S$.

Розглянемо приклад функціональної обфускації VHDL-опису схеми повного суматора FADDER. Нехай порти A0 і B0 є входами сигналів розрядів двох доданків, порт P0 – входом сигналу перенесення від молодшого розряду, S1 – вихідний порт сигналу суми, P1 – вихідний порт сигналу перенесення. При використанні поведінкового VHDL-опису можна визначити значення вихідних портів у вигляді логічних виразів ДНФ функцій суми і перенесення (дод. 4, лістинг 5.1).

Для функціональної обфускації поданого опису розглянемо реалізацію функцій S1 і P1 у вигляді таблиці істинності і двох еквівалентних перетворень (дод. 4, лістинг 5.2).

Результатом RTL-синтезу обфусцованого опису буде схема, що складається з трьох компонент: ПЗП ємністю 8 х 3 біт і два елементи OR2 (рис. 5.5).

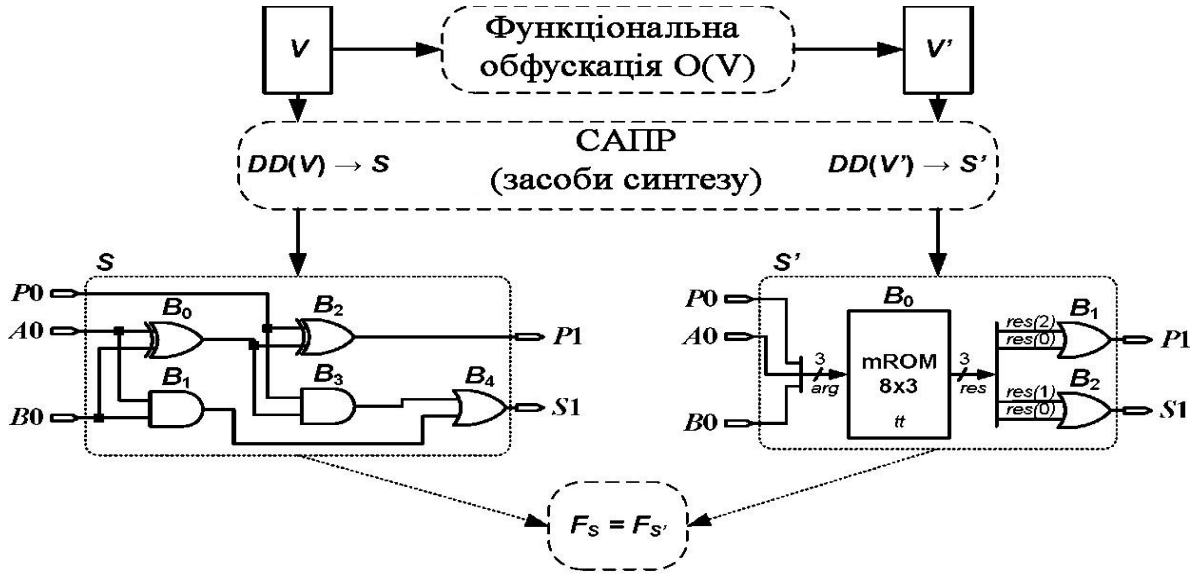



Рис. 5.5. Результат функціональної обфускації

З наведеного прикладу видно, що функціональна обфускація є «нечитабельною» навіть для засобів синтезу. Так, логічні елементи OR2 є функціонально зайвими з огляду на те, що на другі входи цих елементів (лінія res (0)) завжди подається сигнал із значенням логічного нуля, забезпечуючи тим самим трансляцію значень функцій на вихідні порти S1 і P1. Для розуміння суті поданого опису необхідний аналіз змісту таблиці істинності tt або аналіз змісту синтезованого ПЗП (для FPGA – аналіз LUT-блоків).

Завдання обфускації вихідних HDL-описів зводиться до ускладнення розуміння структури та функціональних особливостей розробленої схеми. Якщо до розглянутого прикладу функціональної обфускації застосувати лексичну обфускацію, то для розуміння вихідного VHDL-коду і синтезованої схеми зловмисникові необхідно докласти зусиль, еквівалентних витраченим проектувальником при створенні проектного опису.

Однак застосування обфусцувальних перетворень не є перешкодою для нелегального поширення вихідного HDL-опису з подальшою реалізацією на ПЛІС, тим самим порушуються права інтелектуальної власності проектувальника. Для захисту прав до проектних описів можуть бути застосовані технології «водяних знаків» і «відбитків пальців».

## 5.2. Впровадження «водяних знаків» і «відбитків пальців» у HDL-описі

Технологія «водяних знаків» (watermark) поширена для захисту прав інтелектуальної власності на різні об'єкти цифрових технологій, у тому числі і на програмне забезпечення [95]. У загальному випадку «водяний знак» впроваджується розробником в об'єкт його інтелектуальної власності для можливості в подальшому довести своє право інтелектуальної власності на даний об'єкт.

«Відбиток пальця» (fingerprint) по суті також є «водяним знаком», що несе інформацію не тільки про правовласників, а й користувача, якому легально була надана можливість користування даною копією об'єкта інтелектуальної власності. Розглянемо як можна здійснювати впровадження «водяних знаків» у проектні HDL-описи.

Нехай дано вихідний HDL-опис V цифрового пристрою S. Нехай K є «водяним знаком», а WM - функцією впровадження K в V:

$$WM(V,K) \rightarrow V_K. \tag{5.5}$$

HDL-опис, що передається, з впровадженим «водяним знаком» VK має ряд властивостей, найбільш значущими з яких є такі:

– K не повинно бути присутнім у $V_K$ явно;

– синтезована схема з $V_K$ також має містити K: $DD(V_K) \rightarrow S_K$;

– впроваджений K не має істотно впливати на функціонування, апаратні витрати і часові характеристики цифрового пристрою;

– K має бути підібраний таким чином, щоб виключити випадкову появу його копії в $V_K$ та в $S_K$;

– проектувальник, який здійснив впровадження K, повинен мати можливість довести, що саме він, і ніхто інший, здійснив впровадження $WM(V,K) \rightarrow V_K$.

Перша властивість необхідна для виключення можливості знаходження і видалення «водяного знака» з тексту HDL-опису. Якщо K все ж таки знайдено, то його видалення або заміна має позначитися на функціонуванні пристрою. Спроба видалення або

заміни водяного знака має бути рівнозначна перепроектуванню цифрового пристрою. Впроваджений «водяний знак» має бути прозорим для засобів синтезу для виключення можливості його видалення шляхом зворотного проектування RE, наприклад при спробі перетворити BIT-образ у netlist-опис і далі в HDL-опис з подальшим синтезом: $RE(V_K) \rightarrow S_K$.

Для можливості доведення авторства впровадженого «водяного знака» проектувальник може використовувати значення K як результат шифрування своїх даних (наприклад копірайт-рядка) за допомогою секретного ключа, значення якого відомо лише йому. На рис. 5.6 подана узагальнена схема впровадження «водяного знака» і процесу доведення авторства.

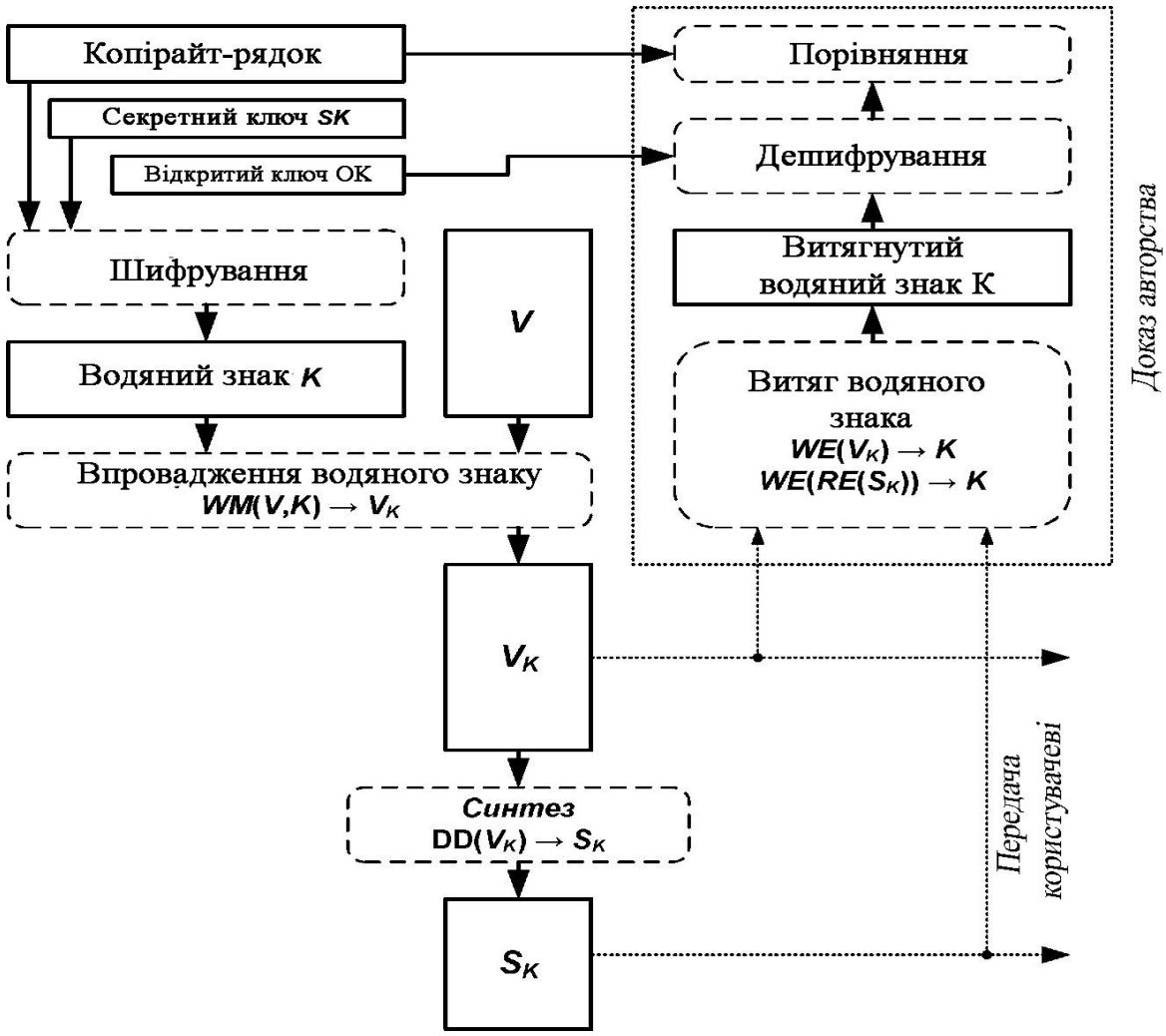



Рис. 5.6. Схема впровадження «водяного знака»

Багато розроблених методів і алгоритмів впровадження «водяних знаків» для програмного забезпечення можуть бути застосовані і до HDL-описів [95].

Розглянемо існуючі методики впровадження «водяних знаків» до проектних описів цифрових пристроїв для ПЛІС [97-99].

Для реалізації комбінаційних схем на FPGA використовуються LUT-блоки, які за своєю суттю є апаратною реалізацією таблиць істинності відповідних логічних функцій. У складі CLB-блоків FPGA типу SPARTAN-3E є можливість використовувати LUT-блоки з різною кількістю входів (від 1 до 4). Так, реалізація логічної функції з кількістю аргументів менше, ніж кількість входів обраного LUT-блока, залишає частково незатребуваним його ресурс, який можна використовувати для зберігання «водяного знака».

Розглянемо приклад реалізації повного суматора FADDER (дод. 4, лістинг 5.1) для FPGA SPARTAN-3E з використанням чотиривходових LUT-блоків (LUT4).

У САПР Xilinx ISE WebPack існує можливість задавання вмісту LUT-блоків безпосередньо в VHDL-описі шляхом використання компоненти LUT4 з пакета технологічних примітивів vcomponents бібліотеки UNISIM.

З огляду на те, що логічні функції S1 і P1 мають три аргументи, для їх реалізації потрібні два блоки LUT4, при цьому в кожному з них будуть використані лише вісім елементів пам'яті з 16 можливих.

На рис. 5.7 наведено структурний VHDL-опис блоків LUT4, що реалізують схему повного суматора, з впровадженим «водяним знаком» 0x1973 (0001100101110011) і результат синтезу цього опису.

Як показано на рисунку, старший байт «водяного знака» 0x73 впроваджений у блок LUT4_P1, який реалізує функцію перенесення P1.

Через те що адресний вхід 11 блока LUT4_P1 постійно набуває значення 0, значення функції P1 записано за адресами {0,1,4,5,8,9,12,13}. Запам'ятовуючі комірки з іншими адресами зберігають старший байт «водяного знака».

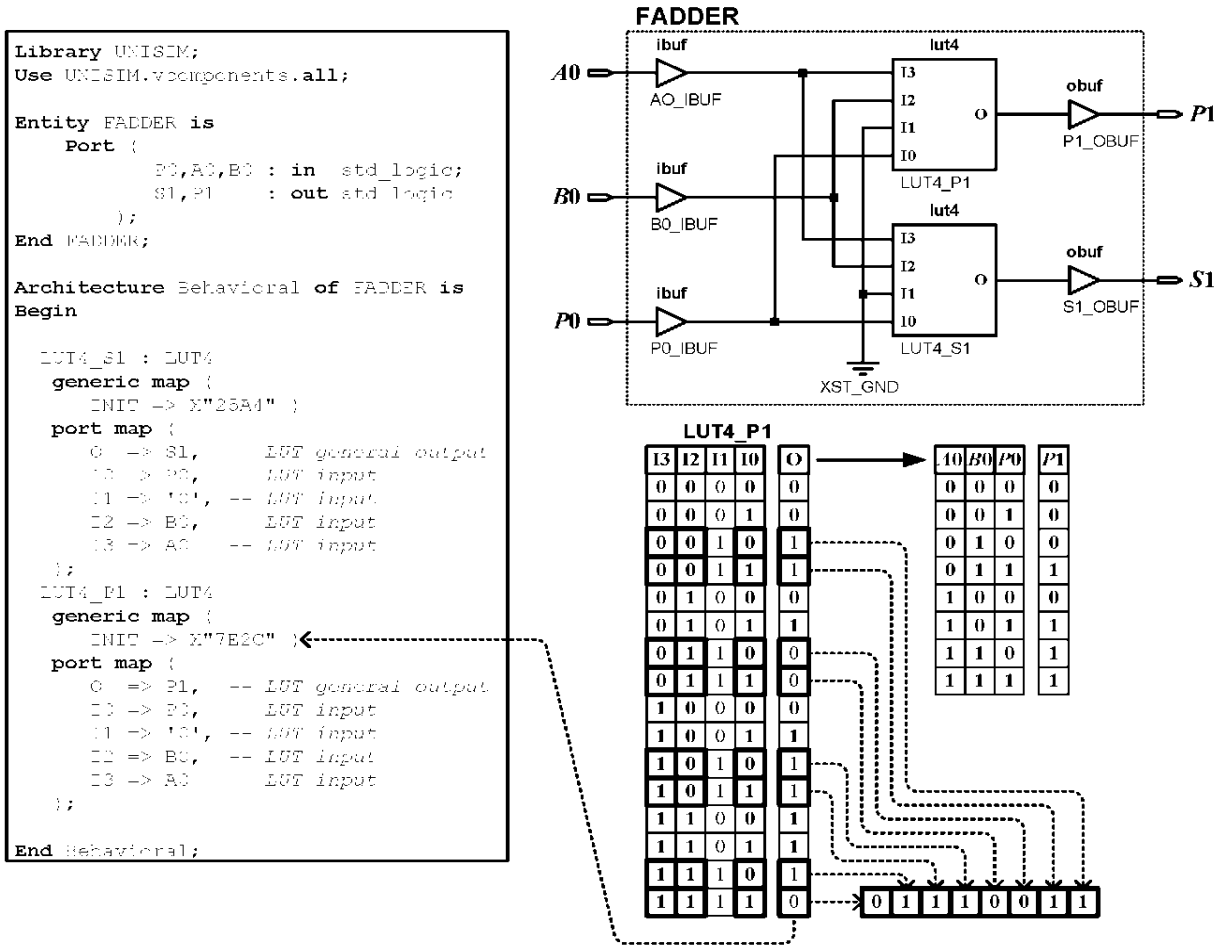



Рис. 5.7. Приклад впровадження «водяного знака» в LUT-блоки

Таким чином, значення «водяного знака» в явному вигляді не подано у вихідному VHDL-описі і воно зберігається в синтезованій схемі пристрою. Крім того, наявність знака не впливає на функціонування пристрою, а для його виявлення і видалення потрібен процес зворотного проектування з аналізом ініціюючих значень обох LUT-блоків.

Для наведеного прикладу існує можливість послідовного вилучення значення «водяного знака» в процесі моделювання HDL-опису та функціонування реалізованого пристрою. Для цього на адресні входи 11 двох LUT-блоків необхідно подати сигнал з рівнем 1, а на входи A0, B0 і P0 – двійковий номер розряду старшого і молодшого байта «водяного знака», значення яких будуть транслюватися на виходи S1 і P1 відповідно.

Розглянемо ще один варіант впровадження «водяного знака» до цифрової схеми кінцевого автомата на ПЗП. На рис. 5.8 подана

діаграма станів кінцевого автомата, що має чотири стани S0, S1, S2 і S3, вхідний сигнал Din і вихідний сигнал Dout.

Нехай стани автомата кодуються так: S0 = {00}, S 1 = {01}, S2 = {10}, S3 = {11}. Тоді для реалізації автомата потрібно два модулі пам'яті: ПЗП ємністю 8х2 біт для вироблення наступного стану (ROMFSM) і ПЗП ємністю 4х1 біт для вироблення значення Dout (ROMOUT) (рис. 5.9).

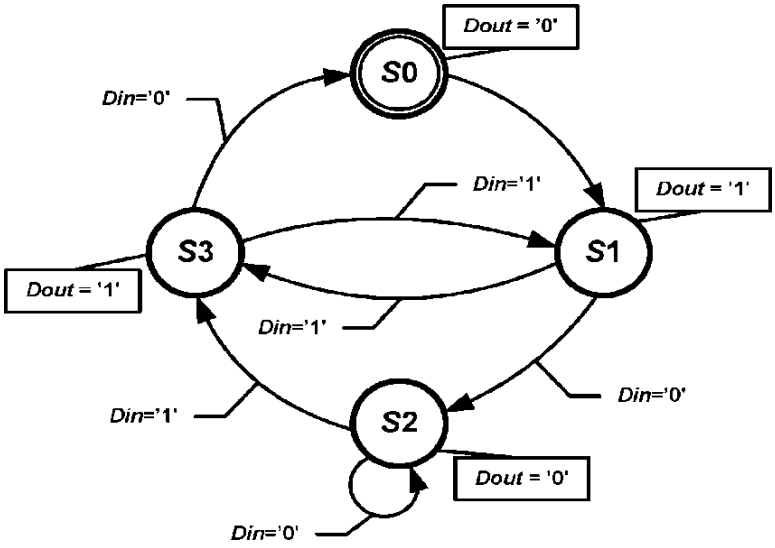



Рис. 5.8. Діаграма станів автомата

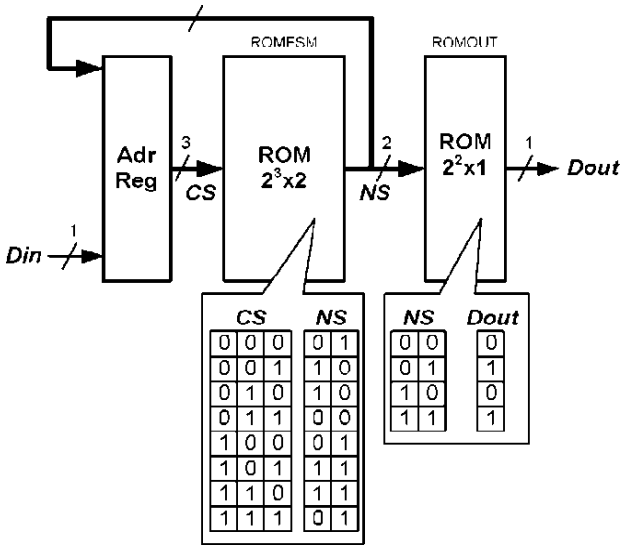



Рис. 5.9. Схема реалізації автомата на ПЗП

На адресні входи запам'ятовуючого пристрою ROMFSM надходить значення поточного стану автомата CS, сформоване зі

значення сигналу входу Din і наступного стану NS: CS = {Din, NS}. Вибране значення з ROMFSM також надходить на адресні входи запам'ятовуючого пристрою ROMOUT, на виході якого виробляється значення сигналу Dout.

При реалізації розглянутого цифрового автомата на FPGA Xilinx SPARNAT-3E можна скористатися технологічними примітивами ПЗП, мінімальна конфігурація яких становить 16x1 (компонента ROM16x1 з пакета vcomponents бібліотеки UNISIM). Для реалізації запам'ятовуючого пристрою ROMFSM потрібно використання двох компонент ROM16x1, що дає можливість впровадити 16-бітний «водяний знак» (рис. 5.10).

Ще більше можливостей для впровадження «водяних знаків» існує при створенні проектних описів кінцевих автоматів з мікропрограмним управлінням. Наприклад «водяний знак» може бути впроваджений при кодуванні виконуваних інструкцій, а для впровадження в тіло мікропрограми можуть бути застосовані методики, що знайшли своє втілення у сфері програмного забезпечення.

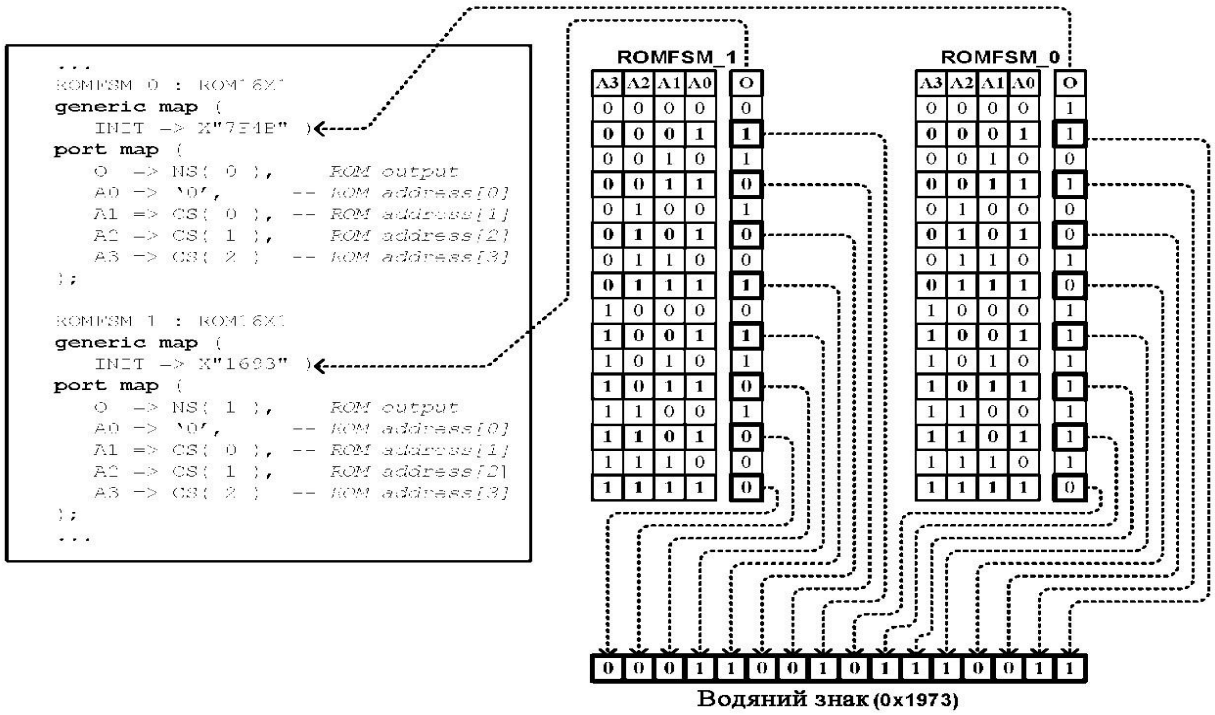



Рис. 5.10. Впровадження «водяного знака» в ПЗП автомата

## 5.3. Методи аутентифікації та ідентифікації цифрових пристроїв, реалізованих на ПЛІС

Багато виробників інтегральних схем вирішують завдання ідентифікації НВІС шляхом реалізації спеціальних регістрів зберігання унікальних ідентифікаторів (серійних номерів), значення яких, як правило, один раз задається на етапі виробництва і в подальшому використанні доступні тільки для читання.

Наявність такого роду ідентифікаторів дозволяє проектувальникам цифрових систем ефективно вирішувати багато завдань: адресація НВІС, підключених до єдиної інформаційної магістралі; використання ідентифікаторів як відкритого ключа при реалізації алгоритмів шифрування; реалізація методів і алгоритмів захисту від несанкціонованого використання тощо.

Однак більшість ПЛІС, що серійно випускаються, не містять регістрів унікальних ідентифікаторів, що ускладнює вирішення перерахованих вище завдань розробниками цифрових систем. У свою чергу призначена для користувача реалізація відповідних регістрів ресурсами ПЛІС не захищена від клонування (несанкціонованого повторення і використання) як під час створення проектних описів, наприклад вихідних HDL-кодів, так і після реалізації в апаратурі.

Одним з найпоширеніших видів атак на цифрові пристрої, реалізованих на ПЛІС типу FPGA, є клонування BIT-образу. Через те що вихідні проектні описи (netlist, VHDL-коди) і самі BIT-образи конфігурації FPGA подаються у відкритому форматі, зловмисник може без особливих технічних зусиль нелегально використовувати їх і тиражувати. Крім того, існуючий у зловмисника BIT-образ може бути перепроектований для отримання netlist і розуміння функціонування цифрової системи. Але частіше BIT-образ клонується (наприклад для реалізації на більш дешевих FPGA) як «чорна скринька» без аналізу і зміни функціонування.

Одним з варіантів захисту цифрових систем для FPGA є застосування елементів апаратної криптографії, зокрема механізму аутентифікації [100]. На рис. 5.11 подана узагальнена схема аутентифікації вигляду «запит-відповідь» цифрового пристрою, реалізованого на FPGA.

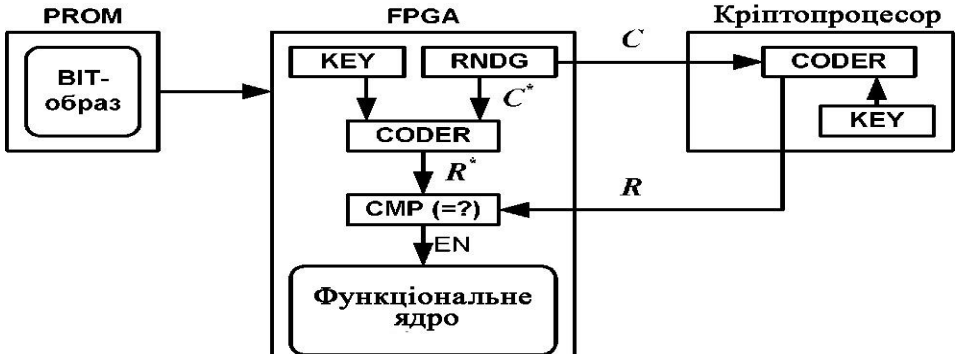



Рис. 5.11. Схема аутентифікації цифрового пристрою

Проектований цифровий пристрій, крім функціонального ядра, містить апаратні блоки схеми аутентифікації: регістр секретного ключа KEY, генератор випадкових чисел RNDG, шифратор CODER і схему порівняння на рівність CMP. До складу всієї цифрової системи, крім FPGA, включені НВІС енергонезалежної пам'яті PROM, яка зберігає BIT-образ реалізованого пристрою, і криптопроцесор, що забезпечує реалізацію криптографічних алгоритмів, у тому числі і алгоритму аутентифікації. Необхідною умовою реалізації аутентифікації є наявність копії секретного ключа KEY і схеми шифрування CODER на боці криптопроцесора. У подібних системах найбільш вразливим є сам секретний ключ KEY. З точки зору зловмисника найкраще виглядає процес зворотного проектування BIT-образу FPGA для визначення значення KEY. У цьому випадку бажано обфускувати не тільки вихідні HDL-описи цифрового пристрою, але і апаратну структуру схем аутентифікації (зокрема регістра секретного ключа).

При ініціалізації системи BIT-образ, що зберігається у відкритому вигляді, передається по відкритому каналу для конфігурації FPGA. У початковий момент часу функціонування реалізованого пристрою генератор RNDG передає криптопроцесору по відкритому каналу запит C. На боці FPGA копія згенерованого запиту C* використовується для шифрування секретного ключа KEY і отримання значення R*. У цей же час криптопроцесор здійснює аналогічні дії, оперуючи вихідним значенням C і копією ключа KEY. Отриманий результат R передається по відкритому каналу на бік FPGA для порівняння зі значенням R*. При рівності значень (R = R*) приймається

рішення про дозвіл функціонування основного ядра цифрового пристрою. В іншому випадку функціонування системи блокується. До недоліків такого підходу можна віднести наявність додаткової апаратури у вигляді криптопроцесора, зберігання секретного ключа в початкових проектних описах і BIT-образі, наявність додаткової апаратури, яка блокує функціонування пристрою, що захищається. Більш надійною є методика, заснована на шифруванні BIT-образу FPGA (рис. 5.12).

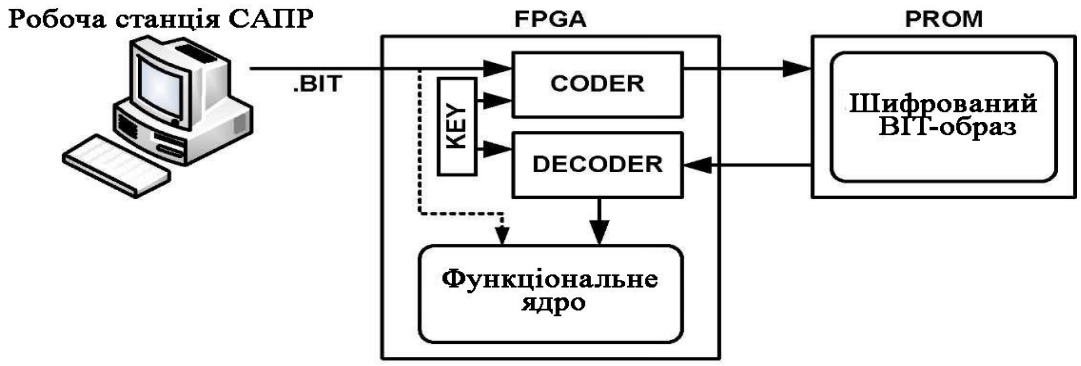



Рис. 5.12. Узагальнена схема шифрування BIT-образу FPGA

Ця методика заснована на використанні апаратного шифрування і дешифрування BIT-образу на боці HBIC FPGA, при цьому секретний ключ шифрування також знаходиться у FPGA і може являти собою унікальний ідентифікатор, значення якого одноразово програмується при виготовленні або може бути перевизначено користувачем.

Отриманий за допомогою засобів автоматизованого проектування BIT-образ передається по відкритому каналу від робочої станції проектувальника безпосередньо у FPGA. Для передачі сконфігурованої цифрової системи стороннім користувачам проектувальник ініціалізує PROM BIT-образом, який шифрується за допомогою блока CODER і секретного ключа KEY. У початковий момент використання системи шифрований BIT-образ по відкритому каналу передається на бік FPGA, де попередньо дешифрується за допомогою блока DECODER і ключа KEY, і тільки потім використовується для конфігурації FPGA.

Проблема захисту від клонування цифрових пристроїв на FPGA, що не містять описаних вище апаратних ресурсів, залишається актуальною. Для вирішення завдання ідентифікації цифрового пристрою проектувальники використовують унікальні ідентифікатори інших інтегральних схем, що входять разом з ПЛІС до складу цифрової системи. Наприклад, цифрова система NEXYS-2 [45] містить НВІС Flash-пам'яті Nymonyx StrataFlash JS28F128 [121], яка має унікальний 64-бітний ідентифікатор, значення якого задається на етапі виробництва.

Проектувальник, використовуючи значення унікального ідентифікатора Flash-пам'яті, отримує його hash-сигнатуру, значення якої впроваджує в проектний опис. Проектувальник перед передачею користувачеві сконфігурованої цифрової системи обчислює значення сигнатури ідентифікатора Flash-пам'яті, наприклад за допомогою використання hash-функцій: HASH (ID)^IDH. Отримане значення впроваджується до проектного опису цифрового пристрою, наприклад як «відбиток пальця». До складу реалізованого пристрою входять такі функціональні блоки: блок читання унікального ідентифікатора ID, блок вилучення «відбитка пальця» IDH, блок апаратної реалізації hash-функції і блок порівняння на рівність (рис. 5.13).

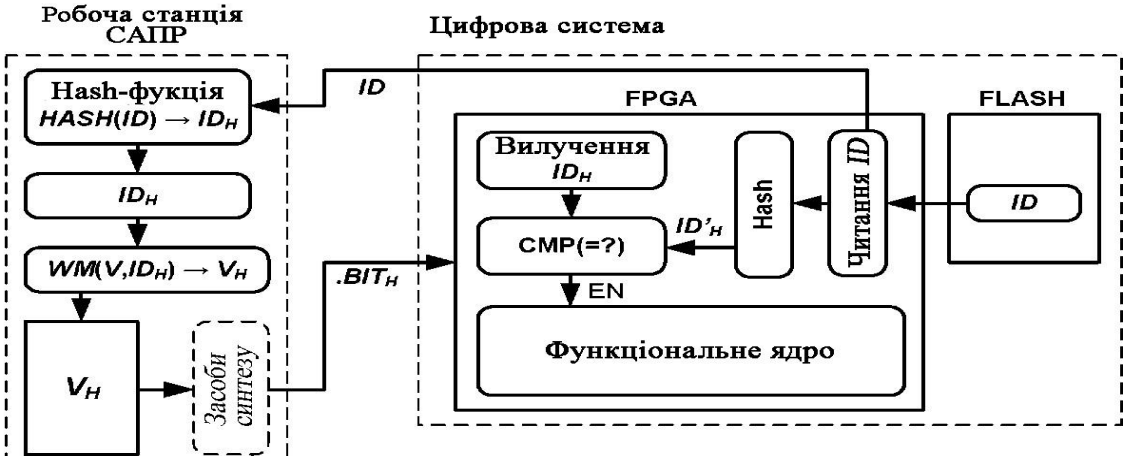



Рис. 5.13. Ідентифікація FPGA за допомогою Flash-пам'яті

Проектувальник, використовуючи значення унікального ідентифікатора Flash-пам'яті, отримує його HASH-сигнатуру, значення якої впроваджує в проектний опис: $WM(V, ID_H) \rightarrow V_H$.

Далі опис $V_H$ піддається синтезу з генеруванням способу конфігурації FPGA $BIT_H$.

При ініціалізації цифрової системи спочатку здійснюється читання значення ідентифікатора ID з Flash-пам'яті і витягування IDH з ресурсів реалізованого цифрового пристрою. Потім блок апаратної реалізації hash-функції обчислює значення IDH, яке порівнюється з витягнутим IDH. У випадку рівності двох значень приймається рішення про дозвіл функціонування основного ядра цифрового пристрою.

Таким чином, ідентифікація цифрового пристрою, реалізованого на FPGA, здійснюється за допомогою унікального ідентифікатора Flash-пам'яті, що ускладнює можливість несанкціонованого клонування як BIT-образу, так і всієї цифрової системи в цілому.

## 5.4. Фізично неклоновані функції

Фізична криптографія, заснована на структурній складності електронних систем, знаходить своє застосування не тільки в методах захисту конфіденційної інформації, але все частіше застосовується для забезпечення захисту цифрових систем від нелегального використання. Вперше ідею застосування фізично неклонованих функцій PUF (Physical Unclonable Function) для цифрових систем запропонував Р. Папп (R. Pappu) у своїх роботах [101, 102], а сучасне тлумачення і формальне визначення PUF було запропоновано П. Туілсом (P. Tuyls) у 2007 році [103]. За визначенням Туілса, фізично неклонованою функцією є характеристика фізичної (цифрової) системи, яка не підлягає клонуванню (копіюванню, відтворенню) на інших системах. Цифрові системи складаються, як правило, з множини компонент, фізичні параметри яких на стадії виробництва набувають випадкових значень. Подібна властивість отримала назву фізичної варіації технологічного процесу. У процесі створення цифрових систем принципово неможливо управляти величинами фізичних параметрів їхніх компонент, визначаючи для них конкретні значення. Таким чином, наявність подібних випадкових параметрів робить кожну цифрову систему унікальною і фізично невідтворною (неклонованою). Ідея

вилучення унікальних параметрів з цифрових систем лежить в основі апаратних PUF.

Формально PUF описується значеннями пар вхідних і відповідних їм вихідних параметрів, які для апаратних PUF є відповідно значеннями вхідних сигналів запиту C (Challenge) і вихідних сигналів відповіді R (Response). Сама ж PUF є функцією перетворення множини запитів Ci в множину відповідей Ri:

$$R_i = PUF(C_i). \quad (5.6)$$

У 2009 році У. Рурмаір (U. Ruhrmair) [104] дав загальне визначення PUF як систем із надвеликим обсягом інформації (SHIC, Super High Information Content).

За визначенням У. Рурмаіра, фізично неклоновані функції являють собою складні некеровані фізичні системи з надвеликим об'ємом структурної інформації, які задовольняють наступні властивості.

Інформація з таких систем може бути неодноразово залучена з високим ступенем надійності шляхом подачі різних запитів Ci і отримання множини відповідей Ri.

Кількість можливих запитів Ci має бути досить великою, щоб всі можливі значення відповідних відповідей Ri не могли бути отримані шляхом повного перебору всіх можливих запитів Ci за прийнятний відрізок часу.

Володіючи інформацією про пару запит-відповідь $\{C_i, R_i\}$, нема ніякої можливості розрахувати, змоделювати або якимось іншим математичним способом передбачити значення пари $\{C_i, R_j\}, i \neq j$, або іншу множину таких пар.

Для фізичної системи з надвеликих об'ємом структурної інформації має бути надзвичайно складним (практично неможливим) її фізичне відтворення або клонування як аналогічної системи, що має ідентичну множину пар {Ci,Ri}.

Для цифрових пристроїв ідея створення PUF заснована на використанні фізичних варіацій технологічного процесу створення інтегральних схем. Такі варіації носять випадковий характер і не можуть бути передбачені, а тим більше клоновані. Незначні відхилення фізичних параметрів при виготовленні ідентичних за функціональністю інтегральних схем у першу

чергу виражаються в розходженні їхніх параметричних характеристик, наприклад у затримках поширення сигналів [105].

Таким чином, завдання реалізації апаратних PUF може бути сформульована як створення цифрового пристрою, що дозволяє приймати за своїми вхідними портами множину запитів Ci і виробляти на своїх вихідних портах множину відповідей Ri таким чином, щоб пари {Ci, Ri} були унікальними, непередбачуваними і неклонованими на інших ідентичних за функціональністю і топологією інтегральних схемах.

Перші приклади реалізації апаратних PUF були засновані на вимірюванні затримки поширення сигналів у реконфігурованих шляхах цифрових пристроїв. Під шляхом цифрового пристрою розуміють множину послідовно підключених логічних елементів, які дозволяють транслювати логічне значення вибраного тону. Кожен елемент у шляху характеризується затримкою d поширення сигналу.

У свою чергу величина параметра d визначається двома складовими:

$$d = d_s + d_r, d_s \gg d_r, \tag{5.7}$$

де *ds* – величина статичної затримки елемента;

*dr* – випадкова динамічна компонента, що залежить від фізичних варіацій технологічного процесу виготовлення цифрового пристрою.

Кількість елементів шляху визначає його довжину і значення сумарної затримки поширення сигналу від початку шляху до виходу останнього елемента. Чим довше шлях, тим більше результуюча випадкова динамічна складова dr, яку можна зареєструвати (виміряти) за допомогою цифрової логіки. Значення результуючої складової dr в принципі не може бути однаковим на різних кристалах інтегральних схем. Одним з найбільш відомих методів реалізації апаратних PUF, заснованих на вимірюванні затримок поширення сигналів, є PUF типу арбітр.

### 5.4.1. PUF типу арбітр

Основна ідея реалізації PUF типу арбітр лежить у побудові двох топологічно і функціонально ідентичних шляхів на одному

кристалі інтегральної схеми. Такі шляхи називаються симетричними шляхами і мають дуже близькі за значеннями величини часу поширення сигналів по них. Однак фізично такі шляхи є принципово різними завдяки відмінності результуючих складових dr для кожного з них. Вимірювання відмінностей у часі поширення сигналів по симетричних шляхах (вимірювання довжин шляхів) може бути здійснено за допомогою одночасної подачі на входи обох шляхів фронту сигналу і визначення на виходах, який шлях виявився довшим. Шляхи проектуються у вигляді конфігурованих симетричних з'єднань за допомогою мультиплексорів, кількість яких визначає не тільки довжину симетричних шляхів, а й потужність множини шляхів, з якої можлива вибірка двох шляхів для подальшого порівняння. По суті керуючі входи мультиплексорів приймають множину значень {0,1}, які можуть сприйматися як запит Ci, значення якого визначає конкретну пару шляхів. Однією з найпростіших схем порівняння (арбітр) симетричних шляхів є синхронний D-тригер, на вхід даних якого подається сигнал з виходу першого шляху, а на вхід синхронізації - сигнал з виходу другого шляху. Таким чином, формоване значення на виході тригера {0,1} є результатом порівняння двох обраних шляхів, а з точки зору неклонованої функції на виході арбітра формується значення відповіді Ri. Найбільш поширена схема, яка реалізує PUF типу арбітр, наведена на рис. 5.14 [106].

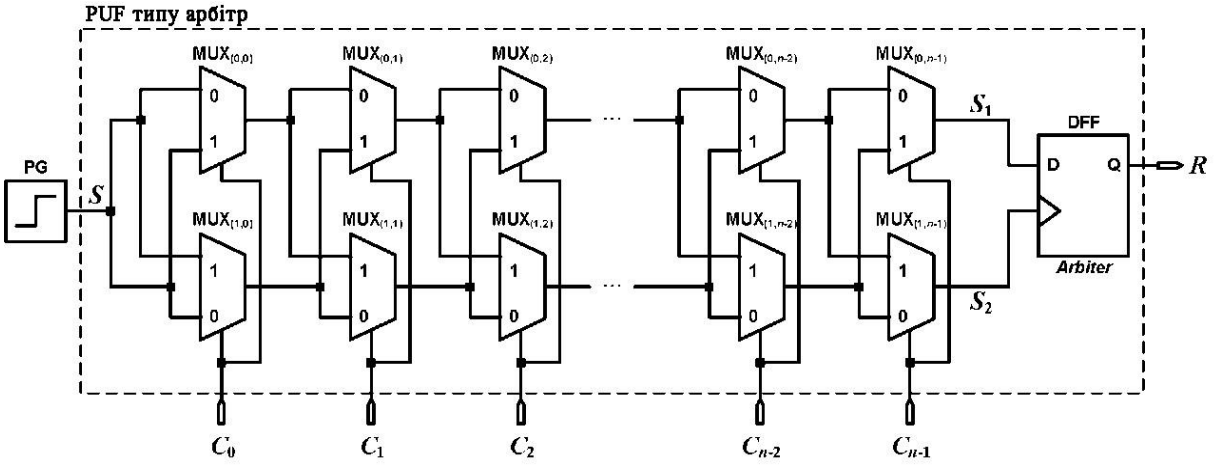



Рис. 5.14. Функціональна схема PUF типу арбітр

Схема має один вхідний порт цифрового імпульсу S, що виробляється генератором PG, n вхідних портів сигналів запиту Ci (Co, Ci, ..., Cn_i) і один вихідний порт відповіді R. До складу схеми включені 2n двовходових мультиплексорів MUX(i;j) (i G {0,1},j G {0, ...,n – 1}) і синхронний D-тригер DFF. Множина мультиплексорів формує конфігуровані цифрові шляхи, ланкою яких є пара двох суміжних мультиплексорів MUX (0j) і MUX1j). Виходи всіх ланок, за винятком останнього, перехресно підключені до нульових і одиничних входів мультиплексорів наступних ланок. При нульовому значенні сигналу Ci = 0 мультиплексор MUX (ij) комутує сигнал зі свого нульового входу на нульовий вхід мультиплексора наступної ланки MUX(i,j+1). При Ci = 1 відбувається перехресна комутація сигналів, при якій сигнали з одиничних входів мультиплексорів MUX (ij) подаються на одиничні входи мультиплексорів MUX(ij+1).

Таким чином, для даної схеми існує можливість сформувати 2n різних пар симетричних шляхів проходження сигналу S.

Сигнали S1 і S2, значення яких формуються на виходах останньої пари мультиплексорів MUX (0, n-1) і MUX (1n-1), є виходами двох симетричних шляхів, які надходять на вхід даних D і вхід синхронізації арбітра DFF. У разі якщо шлях, який пройшов сигнал S1, є довшим за шлях, яким пройшов сигнал S2, арбітр сформує на своєму виході Q значення 0. В іншому випадку на виході Q сформується значення 1 (рис. 5.15). Вихід арбітра безпосередньо підключений до виходу відповідей R поданої схеми PUF.

Перед проведенням порівняння двох обраних шляхів тригер DFF слід встановити в нульовий стан (R = 0). Через симетрію обираних пар шляхів на етапі виготовлення інтегральної схеми неможливо передбачити, який із шляхів конкретної пари має меншу затримку (є більш коротким). У загальному випадку подана схема PUF типу арбітр дозволяє отримувати однобітну відповідь R на n-бітний запит C.

Розглянемо один з можливих варіантів апаратної реалізації PUF типу арбітр. Використовуємо мову VHDL для проектування поведінкового і структурного опису апаратної схеми PUF типу арбітра. Наведемо на поведінковій підмножині VHDL такі компоненти: мультиплексор MUX2 x 1 і арбітр як синхронний D-тригер з можливістю його асинхронного скидання в нульовий стан (дод. 4, лістинг 5.3).

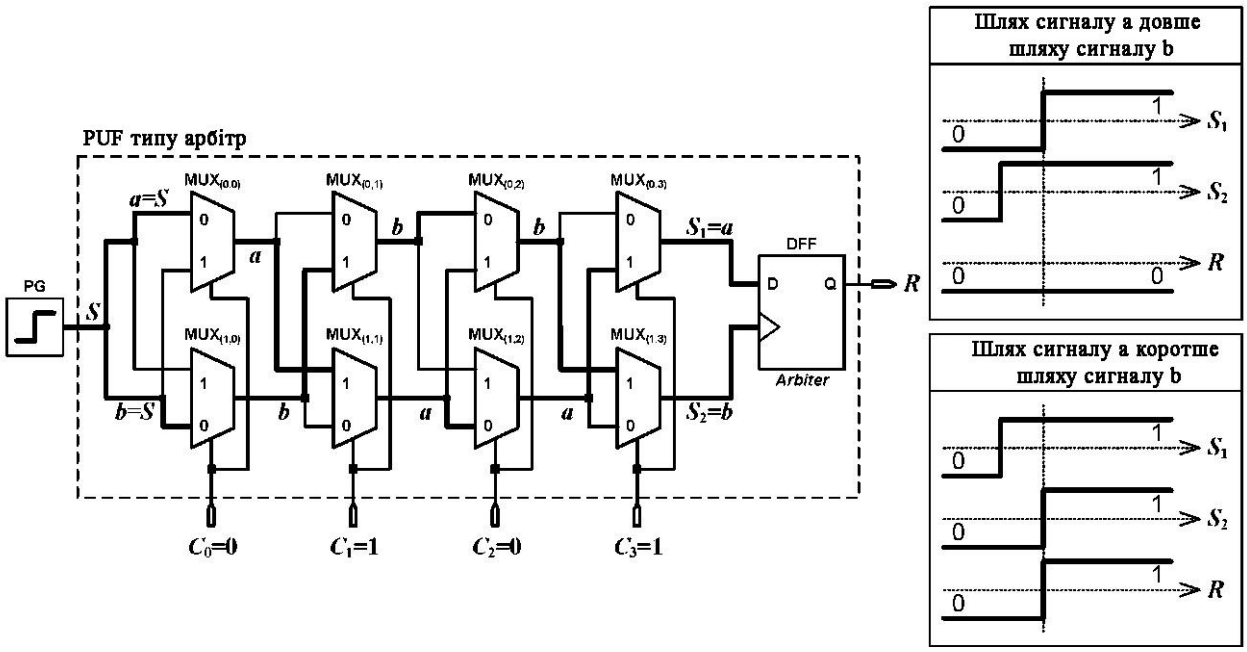



Рис. 5.15. Схема PUF типу арбітр для випадку n = 4

Далі спроектуємо компоненту LINK, яка подаватиме одну конфігуровану ланку для формування різних пар шляхів, що містить два суміжних мультиплексори MUX (0j-) і MUX (1j) (рис. 5.16). У лістингу 5.4 (дод. 4) наведено вихідний код структурного VHDL-опису компоненти LINK, складеного на базі поведінкового опису мультиплексора MUX2 x1.

На підставі спроектованої компоненти LINK складемо VHDL-опис, що налагоджується, конфігурованих пар шляхів довжини N (дод. 4, лістинг 5.5).

Нехай генератор PG формує єдиний імпульс, тривалість якого визначатиметься зовнішнім джерелом синхроімпульсів.

У лістингу 5.6 (дод. 4) подано поведінковий опис генератора PG.

На рис. 5.17 зображена часова діаграма функціонування генератора PG.

При використанні системного генератора, який виробляє синхроімпульси з частотою FSYS, генератор буде формувати єдиний імпульс на своїх вихідних портах протяжністю P = 1 / FSYS. Для вироблення імпульсу необхідно спочатку здійснити ініціалізацію генератора шляхом установлення вхідного сигналу RST = «1». Потім при RST = "0" дозволити процес генерування, подавши на вхідний порт GO значення «1».

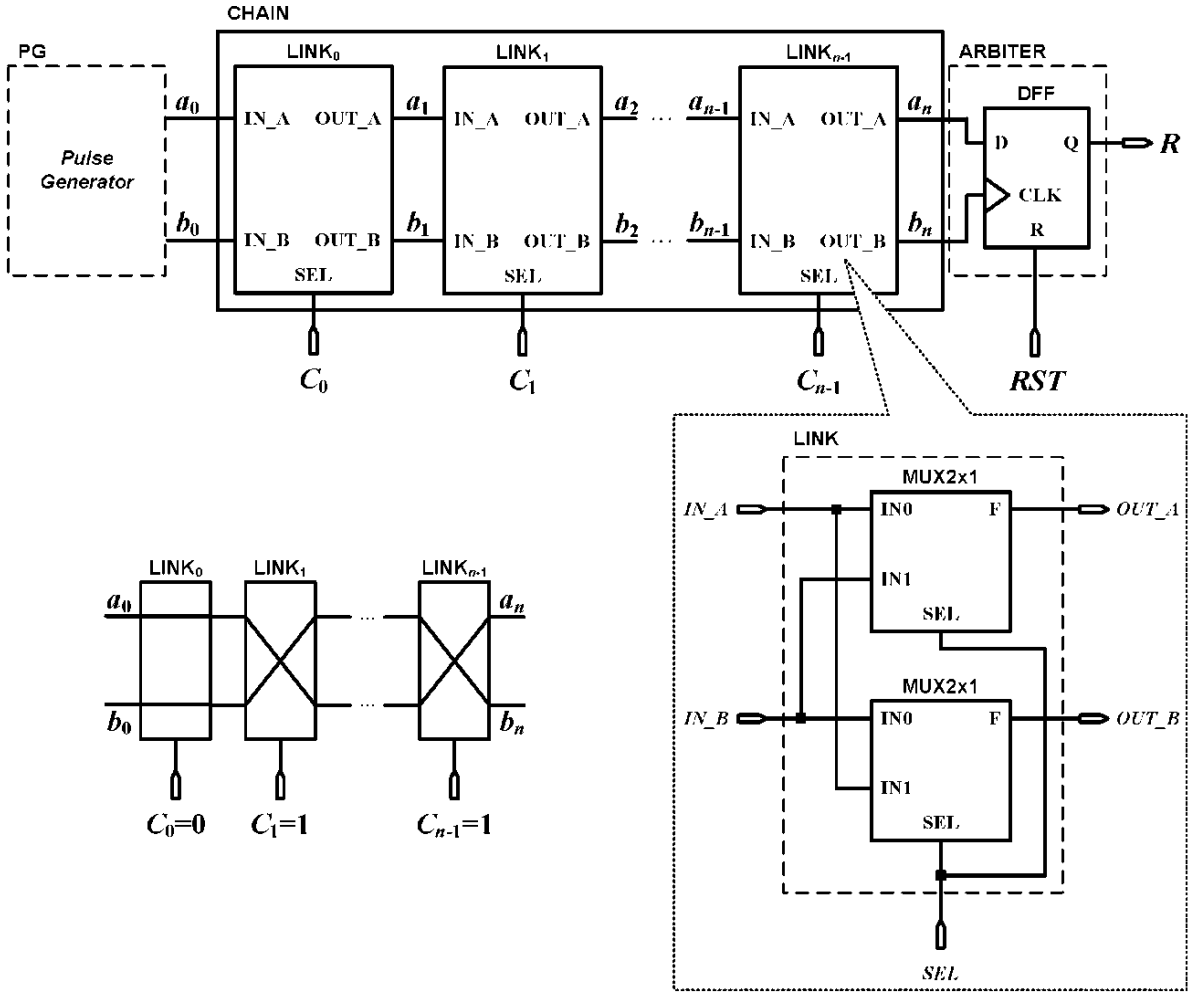



Рис. 5.16. Структура конфігурованих шляхів, що складається з компонент LINK

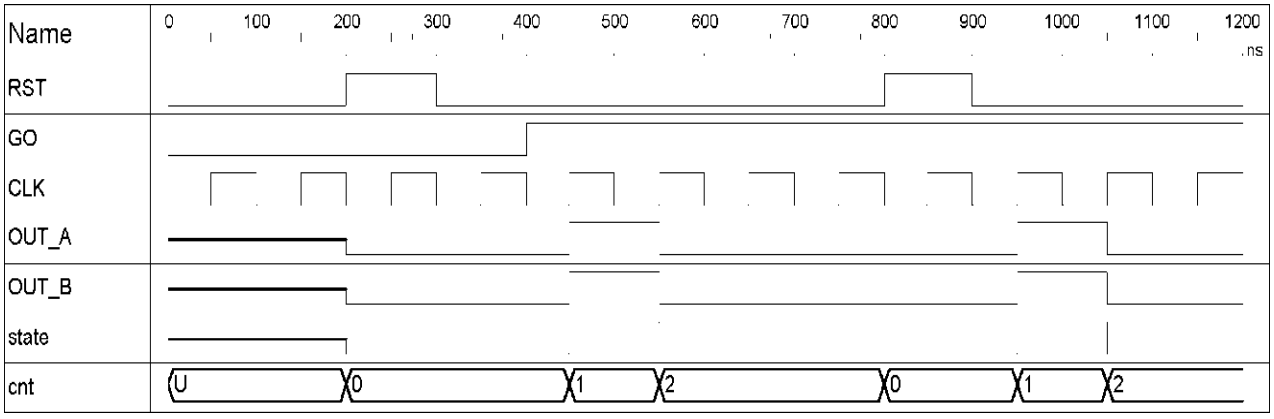



Рис. 5.17. Часова діаграма функціонального моделювання генератора

Схемна реалізація генератора для ПЛІС Xilinx Spartan-3E XC3S500E-5fg320, отримана в САПР Xilinx ISE, подана на рис. 5.18.

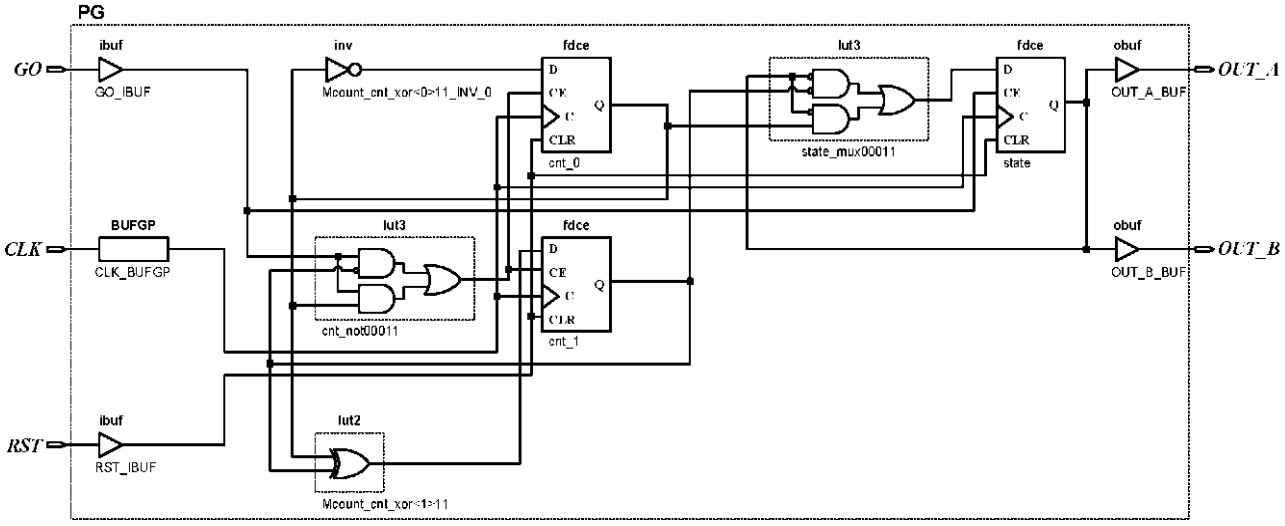



Рис. 5.18. Результат технологічного синтезу генератора

Збирання всіх вищеописаних компонент проведемо за допомогою структурного VHDL-опису (дод. 4, лістинг 5.7).

Подана апаратна реалізація PUF типу арбітр може налагоджуватися шляхом зміни цілочисельного параметра N, що визначає протяжність конфігурованих шляхів і розмірність вхідних шини запиту C. Наприклад, для значення N = 64 результатом синтезу поданого VHDL-опису буде компонента CHAIN, що складається з 64 компонент LINK, що забезпечують конфігурацію пари симетричних шляхів з 264 можливих. При цьому апаратні витрати на реалізацію всієї компоненти APUF складають менше одного відсотка від програмованих ресурсів FPGA XC3S500E-5fg320 (табл. 5.1).

Таблиця 5.1

Апаратні витрати для реалізації PUF типу арбітр (N = 64)

| Назва ресурсу | Кількість використаних блоків | Кількість наявних блоків | Частка витрат, % |
|---|---|---|---|
| Slices | 66 | 4656 | 1,41 |
| Slice Flip-Flops | 4 | 9312 | 0,04 |
| 4-input LUTs | 132 | 9312 | 1,41 |

Існує кілька варіантів отримання багаторозрядної відповіді R на n-розрядний запит C з використанням поданої схеми

апаратної реалізації PUF типу арбітр. Одне з простих рішень полягає в застосуванні генератора псевдовипадкової послідовності (ГПВП), який виробляє n-розрядні двійкові числа. Запит C використовується як початковий стан ГПВП. Потім послідовно виробляються генератором n-розрядні числа, що подаються як запити на входи схеми PUF типу арбітр, на кожен з яких генерується однобітна відповідь R. У підсумку за m тактів функціонування ГПВП виробляється m-розрядна відповідь R, акумульована на виході арбітра (рис. 5.19).

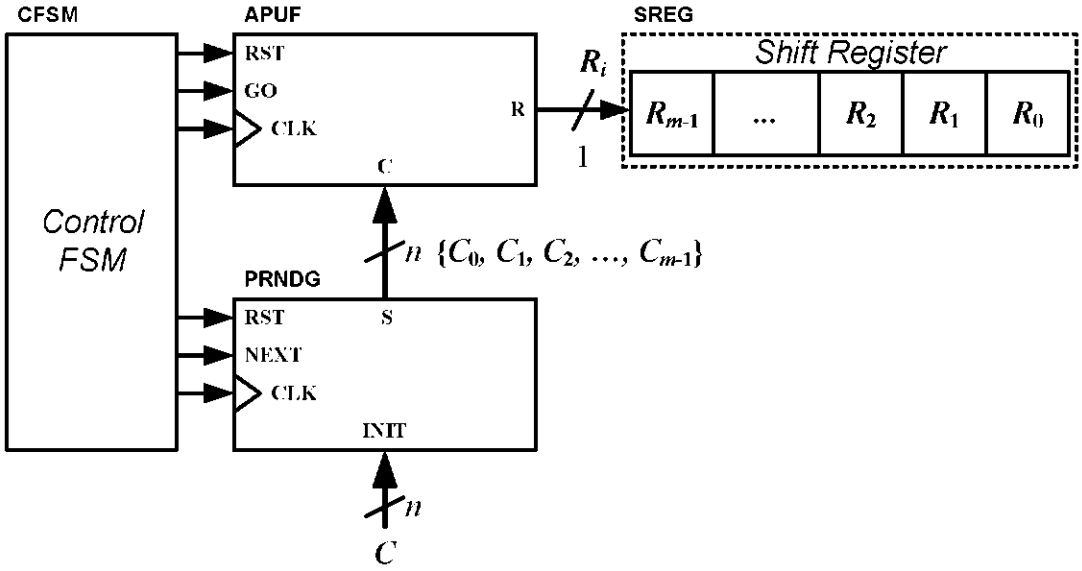



Рис. 5.19. Застосування ГПВП для PUF типу арбітр

Для побудови такої схеми можна за основу взяти спроектовану компоненту APUF, компоненту генератора псевдовипадкових послідовностей PRNDG з можливістю задавання початкового стану і m-розрядний зсувний регістр SREG, що виконує роль перетворювача послідовного коду в паралельний. Крім того, необхідним буде наявність керуючої компоненти CFSM, що забезпечує узгоджене функціонування генератора, регістра зсуву і компоненти APUF.

При використанні великих значень n і m подане рішення є дуже витратним за часом. Як альтернативну схему можна використовувати багаторазове дублювання компоненти APUF, при якому кожен вихід Ri братиме участь у формуванні m-розрядної відповіді (рис. 5.20).

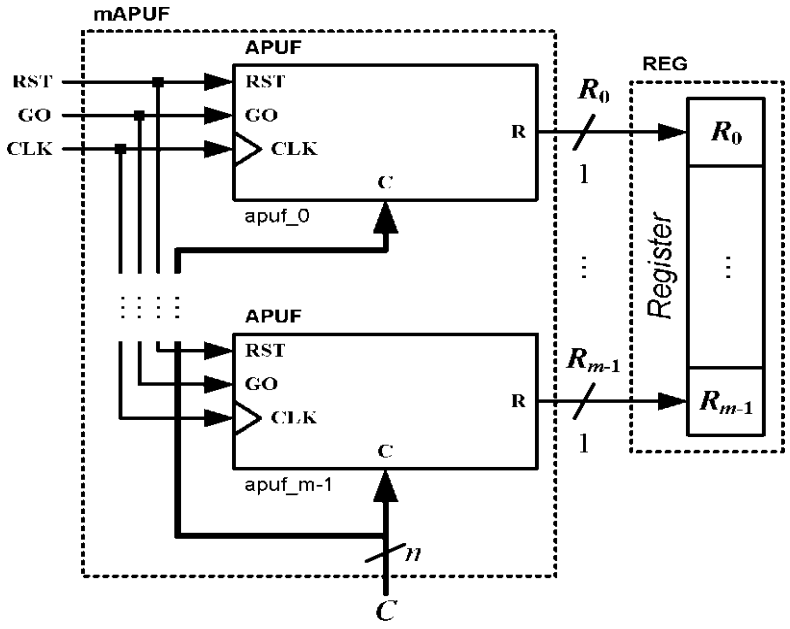



Рис. 5.20. Багаторазове використання компоненти APUF

При використанні другого варіанта час отримання m-розрядної відповіді зменшується порівняно з часом отримання однобітної відповіді, проте апаратна складність PUF типу арбітр збільшується в m раз, що призведе до такого самого пропорційного збільшення площі кристала і зростання потужності, споживаної всією схемою.

Як альтернативна схема PUF типу арбітр у роботі [107] була запропонована побудова двох симетричних шляхів на основі тристабільних буферів, що виконують роль елементів затримки в шляхах проходження сигналів. На рис. 5.21 подана загальна схема PUF типу арбітр з використанням тристабільних буферів.

Схема складається з двох конфігураційних шляхів проходження сигналу S, що містять n елементів TC (ij) (i e {0,1}, j e {0, ..., n / 2 - 1}). При цьому верхній шлях формується з елементів TC (0j), а нижній – з елементів TC (1; j-). Сигнал S1 з виходу верхнього шляху надходить на D-вхід арбітра, а сигнал S2 з виходу нижнього шляху - на вхід синхронізації. Кожен елемент TC (i, j) складається з пари тристабільних буферів TBUF і TBUFo з парафазним управлінням. Так, буфер TBUF перемикається в третій стан шляхом подачі значення логічного нуля на його керуючий вхід, а буфер TBUFo – шляхом подачі логічної одиниці. Входи і виходи буферів об'єднані відповідними портами

Si і So. Конфігурація кожного елемента TC^j) здійснюється шляхом подачі керуючого сигналу на вхід C. За умови C = 0 сигнал з входу Si транслюватиметься на вихід So за допомогою тристабільного буфера TBUFo, в іншому випадку - за допомогою буфера TBUF, утворюючи тим самим пару симетричних шляхів.

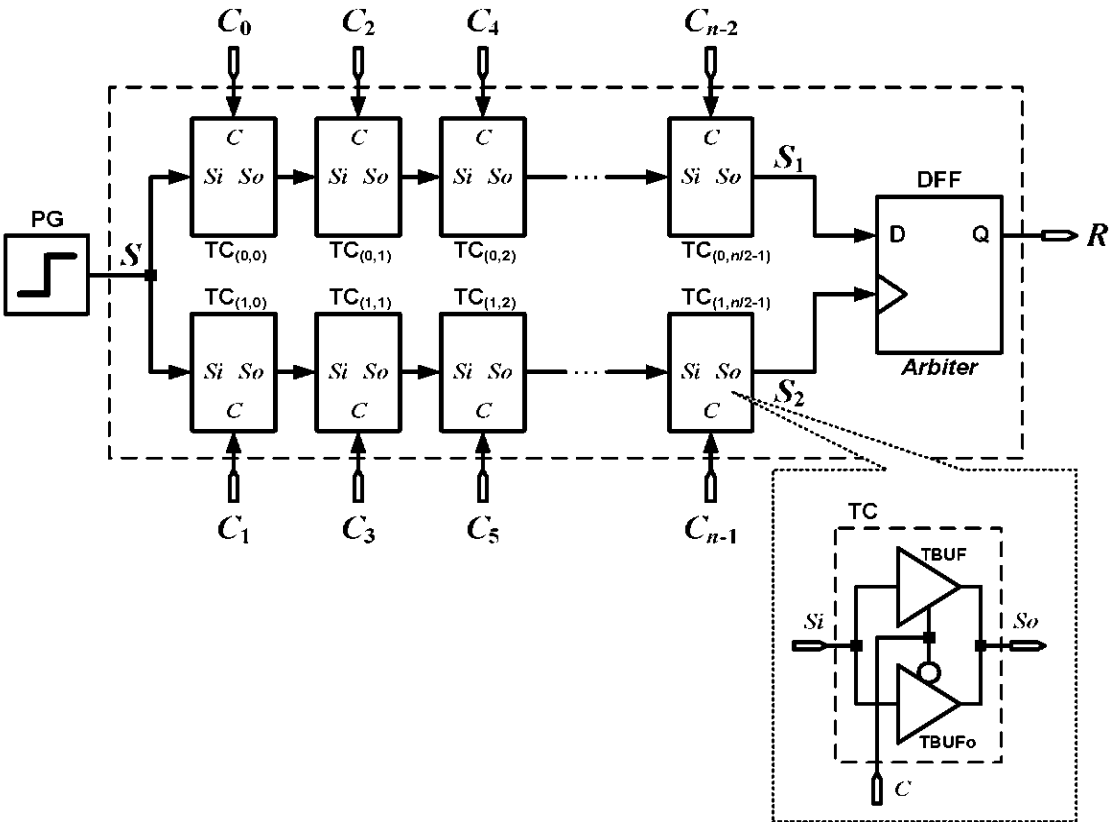



Рис. 5.21. Використання тристабільних буферів для побудови конфігураційних шляхів

Як показали проведені дослідження, схема, побудована на тристабільних буферах, споживає майже на 20 % менше енергії і займана площа кристала на 30 % менше порівняно зі схемою на мультиплексорах [107].

До основних недоліків PUF типу арбітр можна віднести її невисоку надійність через наявність систематичної асиметрії у своїй топології, одержуваної під час неконтрольованого автоматизованого синтезу, при якому практично неможливо управляти ідентичністю всіх елементів, довжиною сполучних провідників, рівністю затримок поширення сигналів і т. д. Для інтегральних схем типу ПЛІС застосування PUF типу арбітр є ще більш недостовірним через явну асиметричність програмованих з'єднань.

### 5.4.2. PUF на основі кільцевих генераторів (RO-PUF)

Значно більшу надійність мають PUF на основі кільцевих генераторів (RO-PUF, Ring Oscillator PUF) [108, 109]. Цей тип фізично неклонованих функцій заснований на використанні частотних характеристик кільцевих генераторів RO, що являють собою схеми з послідовно підключених інверторів з негативним зворотним зв'язком. Кількість інверторів має бути непарною, що забезпечує формування вихідного сигналу у вигляді меандра, частота якого визначається величиною затримки поширення сигналу по ланцюгу зворотного зв'язку. Для керованості схем RO в ланцюг зворотного зв'язку додають двовходовй логічний елемент І, що дозволяє утримувати генератор у стабільному стані. Через варіації затримок сигналу на елементах генератора два ідентичних за топологією і функціональністю RO мають принципово різні частоти формування вихідного сигналу. Результат порівняння частот двох генераторів є основою для формування однобітної відповіді для схеми RO-PUF. Таким чином, пара генераторів RO, реалізована на одній або двох інтегральних схемах, матиме довільне співвідношення частот і буде унікально характеризувати цю пару або інтегральну схему.

На рис. 5.22 подана узагальнена функціональна схема RO-PUF.

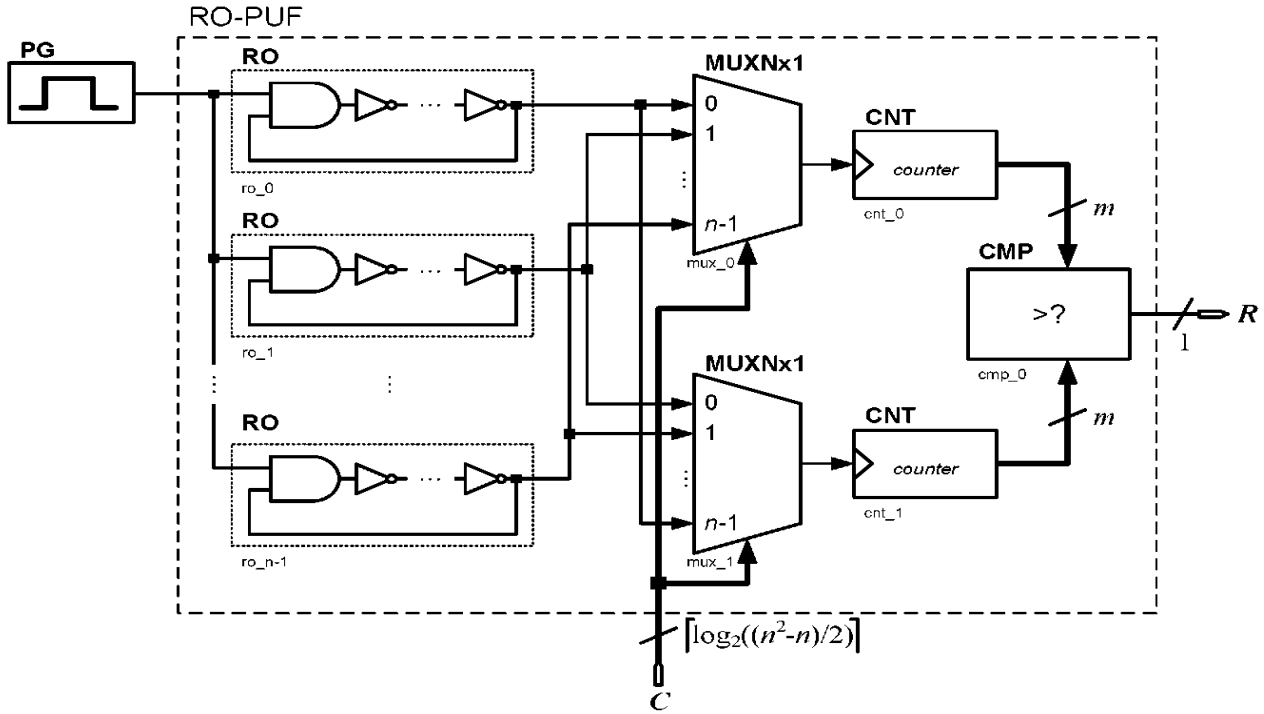



Рис. 5.22. Функціональна схема RO-PUF

Схема будується на n кільцевих генераторах ro_i, управління якими здійснюється зовнішнім джерелом одиночного імпульсу PG. Вироблені генераторами імпульси надходять на входи мультиплексорів mux_0 і mux_1, кожен з яких комутує один з n сигналів, що надійшли, на свій вихідний порт.

Вибір комутованого імпульсу здійснюється за допомогою єдиного селективного [log2((n2 — n)/2)]-розрядного запиту C. Виходи мультиплексорів підключені до вхідних портів сигналу синхронізації двійкових m-розрядних лічильників cnt_0 і cnt_1, які підраховують кількість згенерованих імпульсів. Після закінчення підрахунку два m-розрядних значення лічильників порівнюються на компараторі cmp_0. Сигнал, що формується компаратором, є однобітною відповіддю R на запит C всієї схеми RO-PUF.

Часовий інтервал вимірювання кількості згенерованих імпульсів задається тривалістю утримання вихідного сигналу генератора PG в значенні 1, а біт відповіді R сигналізує про нерівність частот двох обраних генераторів RO.

Часовий інтервал вимірювання кількості згенерованих імпульсів задається тривалістю утримання вихідного сигналу генератора PG в значенні 1, а біт відповіді R сигналізує про нерівність частот двох обраних генераторів RO.

Розглянемо приклади реалізації компоненти RO і двох мультиплексорів MUXNx1 мовою VHDL. Почнемо з компоненти кільцевого генератора RO. Слід зазначити, що для запобігання логічній оптимізації схеми послідовно ввімкнених інверторів будемо використовувати структурний VHDL-опис зі встановленим параметром програмного засобу синтезу, який зберігає ієрархію внутрішніх компонент описуваної схеми. Наприклад, для САПР Xilinx ISE даний параметр називається Keep Hierarchy і розташований у закладці Synthesize :: Process Properties : Synthesis Options : Keep Hierarchy [83].

У лістингу 5.8 (дод. 4) подано структурний VHDL-опис масштабованої компоненти кільцевого генератора RO, заснованої на використанні сторонніх компонент інвертора (INV1) і двохвходового логічного елемента I (AND21).

На рис. 5.23 подано результат технологічного синтезу компоненти RO для випадку N = 3 (N – коефіцієнт масштабування (дод. 4, лістинг 5.8)).

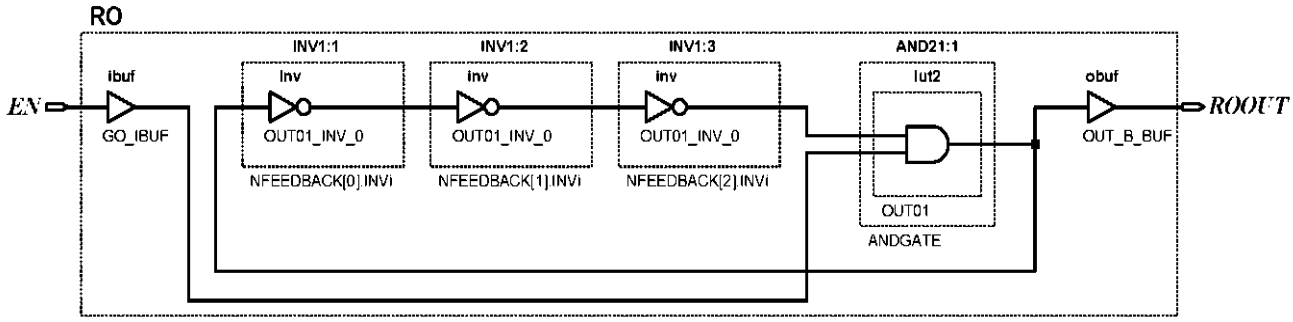



Рис. 5.23. Результат технологічного синтезу компоненти RO

Слід зазначити, що формування негативного зворотного зв'язку в структурі генератора можливе тільки при непарних значеннях N. Для оцінювання частоти імпульсів, що виробляються генератором RO, можна скористатися його параметричної моделлю, створеною автоматизованою системою синтезу після успішних етапів розміщення і трасування. Така модель забезпечує спостережуваність сигналів у внутрішніх точках синтезованої схеми з урахуванням усереднених параметрів затримок поширення сигналів обраної НВІС. На рис. 5.24 подана часова діаграма параметричного моделювання компоненти RO для випадку її реалізації на НВІС XC3S500E-5fg320. Часова діаграма отримана за допомогою компоненти ISIm, що входить до складу САПР Xilinx ISE.

Відповідно до проведеного моделювання значення половини періоду сигналу, який формується на виході генератора, становить (16,602 - 13, 280) = 3,322 нс, а відповідно частота сигналу приблизна дорівнює 150,51 МГц (1/(2 • 3,322 • 10-9) « 150,51 • 106).

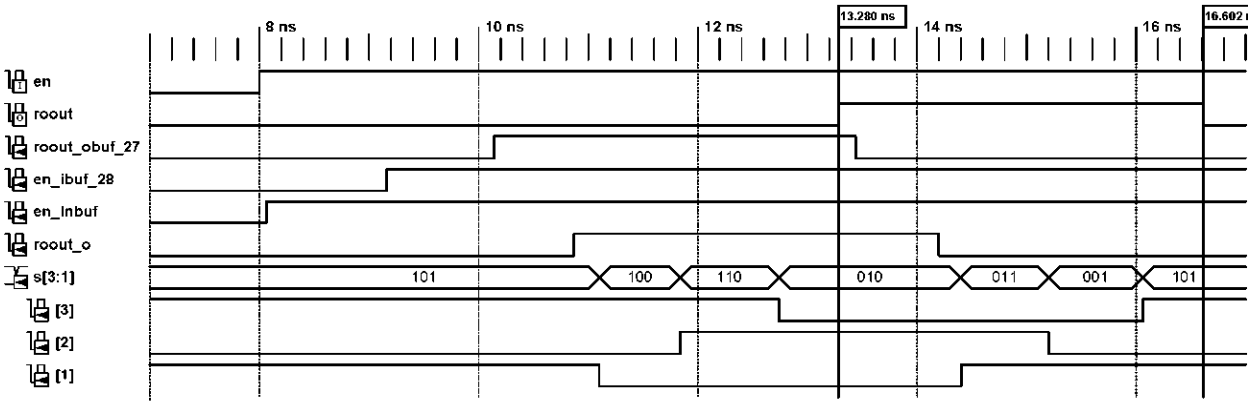



Рис. 5.24. Результат параметричного моделювання схеми генератора RO

Отримане значення частоти цілком залежить від часу затримки поширення сигналу по всіх елементах схеми генератора включаючи конфігураційні міжз'єднання. Наприклад, затримка поширення сигналу від виходу першого інвертора (сигнал s (1)) до виходу другого інвертора (сигнал s (2)) становить 0,763 нс, у той час як подібна затримка між другим і третім інвертором складає 0,885 нс. Різниця між затримками в 0,122 нс обумовлена територіальною віддаленістю інверторів на кристалі і різною протяжністю конфігурованих з'єднань.

Таким чином, реалізований генератор RO здатний формувати вихідний сигнал з частотою, обумовленою технологічними особливостями елементів, з яких сформована схема генератора.

У табл. 5.2 подано значення частот вихідних імпульсів генератора RO, отримані при параметричному моделюванні для різних значень масштабованого коефіцієнта N.

Таблиця 5.2

Значення частоти FRO вихідних імпульсів

| N | Частота FRO, МГц |
|---|---|
| 1 | 375,65 |
| 3 | 150,51 |
| 5 | 108,24 |
| 7 | 82, 27 |
| 9 | 63, 36 |
| 11 | 50, 84 |

Перейдемо до проектування двох компонент мультиплексорів MUXNx1 для можливості формування пари сигналів від двох різних кільцевих генераторів. Оберемо варіант реалізації мультиплексорів з кількістю вхідних інформаційних сигналів N = 2m і m-розрядною вхідною шиною селективних сигналів SEL.

У лістингу 5.9 (дод. 4) подано поведінковий опис масштабованої компоненти мультиплексора MUXNx1 і структурний опис компоненти MUXNx2, що дозволяє вибирати пару сигналів, що генеруються двома незалежними генераторами RO (рис. 5.22).

При реалізації наведених компонент мультиплексорів для ПЛІС типу FPGA можна забезпечити оптимізацію синтезованого результату за рахунок використання не тільки LUT-блоків, а й мультиплексорів, вбудованих у SLICE-блоки.

Для забезпечення цієї оптимізації необхідно вказати такі параметри синтезатора: Synthesize :: Process Properties :: HDL Options :: Mux Extraction (YES).

Наприклад, при ввімкненій опції Mux Extraction схемна реалізація мультиплексора MUXNx2 з коефіцієнтом масштабування m = 6 вимагає 34 SLICE-блоків і 66 LUT-блоків, а при вимкненій опції - 46 SLICE-блоків і 90 LUT-блоків.

У загальному випадку схемна реалізація RO-PUF вимагає великих витрат апаратури на реалізацію порівняно з PUF типу арбітр. Пов'язано це в першу чергу з апаратною складністю схем мультиплексорів, що також негативно позначається і на кількості можливих формованих пар сигналів для порівняння. Так, для коефіцієнта масштабування m = 10 схемна реалізація компоненти MUXNx2 в принципі неможлива для НВІС XC3S500E-5fg320 через недостатність наявних апаратних ресурсів. Однак, на відміну від PUF типу арбітр, RO-PUF має досить високий ступінь надійності при реалізації для ПЛІС [108].

### 5.4.3. PUF на основі статичного ОЗП (SRAM PUF)

Статичні оперативні запам'ятовуючі пристрої (СОЗП) широко застосовуються в обчислювальній техніці для зберігання даних. Елемент СОЗП, що дозволяє зберігати один біт інформації, як правило, будується на основі чотирьох транзисторів, які реалізують два інвертори з перехресними зворотними зв'язками (елемент з двома стійкими станами) [111]. У цифровій схемотехніці подібним елементом може бути RS-засувка, проектована на логічних елементах 2I-HI (рис. 5.25).

Одночасна подача значення логічної одиниці на входи R і S дозволяє фіксувати поточний стан засувки, який було змінено останньою операцією запису. Припустимо, що такий стан вхідних портів має місце при ввімкненні напруги живлення. Через симетрію RS-засувки апріорно невідомо, яке конкретне значення буде зафіксовано на виході Q (nQ). Це значення є випадковим і визначається багатьма факторами [112].

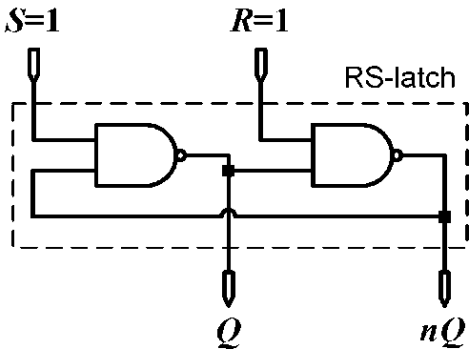



Рис. 5.25. SRAM PUF, реалізована у вигляді RS-засувки

Експериментально підтверджено, що більшість комірок СОЗП при ввімкненні напруги живлення переважно переходять в один з двох можливих станів. Відбувається це тому, що кожен запам'ятовуючий елемент СОЗП, що є RS-засувкою, має багато неоднорідних елементів (фізично різна довжина провідних ліній, неоднорідність фізичних і хімічних властивостей окремих ділянок кремнієвого напівпровідника і т. п.).

Подібний дисбаланс є причиною того, що деякі запам'ятовуючі елементи СОЗП частіше перемикаються в один конкретний стан, ніж у протилежний. Однак передбачити такий стан практично неможливо. І тільки менша частина запам'ятовуючих елементів СОЗП при ввімкненні напруги живлення перемикається в істинно випадковий стан (наближений до рівномірного розподілу), а велика частина елементів стійко перемикається в стан 0 або 1.

У роботі [113] експериментально було підтверджено, що кількість випадкових станів запам'ятовуючих елементів СОЗП є обмеженою. Для дослідження були обрані три ідентичних НВІС FPGA Virtex II Pro. Під час експерименту аналізувався стан 4096 запам'ятовуючих елементів пам'яті конфігурації після ввімкнення напруги живлення. Пам'ять конфігурації являє собою двовимірну матрицю запам'ятовуючих елементів розмірністю 32 х 128. У результаті багаточисельних експериментів було виявлено, що більше 90 % досліджуваних елементів пам'яті завжди встановлювалися в стан 0, а менше 10 % – у стан 1. І тільки менше 1 % елементів встановлювалися рівноймовірно в стан 0 або 1.

Отримані результати дозволяють зробити висновок, що SRAM PUF є дуже ненадійними, а особливо стосовно

програмованих логічних пристроїв. Пояснюється це регулярністю топології FPGA: у будь-якому функціонально симетричному елементі, наприклад в RS-засувці, реалізованій на FPGA, практично завжди присутня прогнозована асиметрія. Для вирішення цієї проблеми була запропонована нова концепція на базі SRAM PUF, що отримала назву PUF типу «метелик» (Butterfly PUF).

### 5.4.4. PUF типу «метелик» (Butterfly PUF)

Вперше фізично неклонована функція типу «метелик» була описана в роботі [114]. Основна ідея реалізації даної PUF ґрунтується на ефекті SRAM PUF при ввімкненні живлення з такою зміною: один запам'ятовуючий елемент будується на двох D-засувках з перехресними зворотними зв'язками, що дозволяють перемикати всю схему в стан функціонування кільцевого генератора. Спроектуємо подібну схему з урахуванням технологічних особливостей FGPA Xilinx SPARTAN-3E і проведемо параметричне моделювання результату розміщення і трасування (place & route).

До складу LUT-блоків обраної HBIC FPGA входять налагоджувані елементи пам'яті, які можуть бути налагоджені як D-засувки з можливістю асинхронного установлення в одиничний стан і асинхронного скидання в нульовий стан. У бібліотеці технологічних примітивів подібний функціональний елемент має назву ldcp, а в бібліотеці параметричних моделей – X_LATCHE. Складемо поведінковий опис такої засувки і на його основі спроектуємо схему PUF типу «метелик» (дод. 4, лістинг 5.10).

На рис. 5.26 подано результат технологічного синтезу VHDL-опису компоненти BPUF для FPGA XC3S500E-5fg320.

Розглянемо функціонування наведеної схеми BPUF. Припустимо, що при ввімкненні живлення сигнал дозволу на вхідному порту E і сигнал управління на вхідному порту FLY встановлені в стан логічної одиниці. Таким чином, засувка DLAT1 виробляє на своєму вихідному порту Q сигнал у стані логічної одиниці, а засувка DLAT0 – сигнал у стані логічного нуля, який у свою чергу транслюється на вихідний порт Q всієї схеми. При цьому вся схема знаходиться в стабільному стані. При

подачі сигналу з рівнем логічного нуля на вхідний порт FLY вся схема переходить у нестабільний стан, у якому забезпечується генерування цифрових імпульсів на виході Q. Період імпульсів визначається виключно часом спрацьовування двох засувок і часом поширення сигналу по лініях s0 і s1. Таке функціонування гіпотетично можливе при абсолютно ідентичних параметрах двох D-засувок і провідних ліній, що забезпечують їх з'єднання.

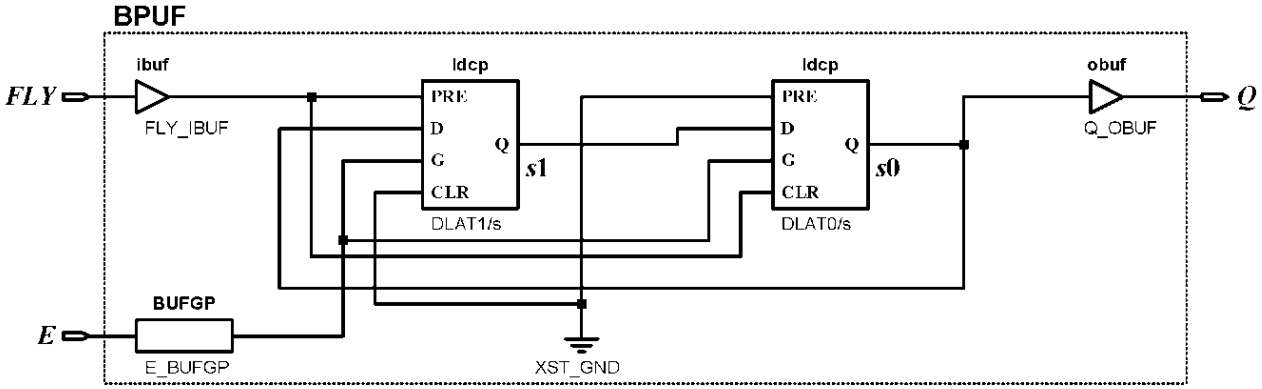



Рис. 5.26. Схемна реалізація PUF типу «метелик»

Однак на практиці, з огляду на заздалегідь технологічну асиметрію, одна з засувок спрацює швидше за іншу, що призведе до встановлення конкретних значень на виходах обох елементів ldcp. Потім через певний час на спільний вхід дозволу E подається сигнал з низьким рівнем, що змушує обидві засувки припинити своє функціонування зі збереженням поточного стану. Таким чином, кінцевий стан схеми (рівень сигналу на виході Q) буде обумовлено виключно випадковою різницею затримок на лініях s0 і s1, що утворюють зворотний зв'язок, і на лінії сигналу FLY.

Для реалізації концепції фізично неклонованої функції на основі наведеної схеми можна використовувати кілька варіантів. Наприклад, формований запит C може являти собою тривалість утримання сигналів E і FLY в активному і неактивному стані щодо кількості імпульсів генератора системної частоти. Таким чином, з використанням різних за тривалістю керуючих сигналів E і FLY на виході схеми буде формуватися однобітна відповідь R схеми PUF типу «метелик».

За необхідності вироблення m-розрядної відповіді R можна використовувати багаторазову реалізацію схеми, наведену на рис. 5.26, із загальними керуючими сигналами E і FLY і паралельними виходами Q.

### 5.4.5. Комбіновані фізично неклоновані функції

Зацікавленість розробників цифрової апаратури і дослідників у сфері апаратної криптографії змушує шукати більш досконалі і надійні способи реалізації фізично неклонованих функцій. Розглянемо деякі підходи, базовані на описаних вище схемних реалізаціях PUF. Так, у роботі [115] було запропоновано удосконалення PUF на основі статичного ОЗП. Як було показано вище, основою схемної реалізації SRAM PUF є RS-засувка, що дозволяє комбінаційній схемі зі зворотним зв'язком стабільно зберігати потрібне установлення логічного 0 або 1 (рис. 5.25). Функціонування такої схеми може бути подано у вигляді таблиці переходів, яка однозначно визначає функціонування RS-засувки. Сигнали на входах R і S схеми можуть набувати довільних значень (R, S) е {(0,0); (0,1); (1,0); (1,1)}, у той же час вихідні порти можуть формувати тільки три комбінації сигналів (Q,nQ) е {(0,1); (1,0); (1,1)}.

У табл. 5.3 подані всі можливі комбінації вхідних сигналів і станів RS-засувки. Символами (X, X) е {(0,0); (0,1); (1,0); (1,1)} позначені байдужі значення вхідних сигналів на портах R і S.

Таблиця 5.3

Таблиця переходів RS-засувки

| Поточні значення на входах (R, S) | Наступні значення на входах (R, S) | Поточний стан (Q, nQ) | Наступний стан (Q, nQ) |
|---|---|---|---|
| = (X,X) | (0, 0) | (Q,nQ) | (1,1) |
| = (X,X) | (0,1) | (Q,nQ) | (0,1) |
| = (X,X) | (1,0) | (Q,nQ) | (1, 0) |
| = (0,0) | (1,1) | (Q,nQ) | (Q,nQ) |
| = (0,0) | (1,1) | (Q,nQ) | (?, ?) |

У разі стандартного застосування RS-засувки як пам'ять великого елемента забороняється послідовна подача двох наборів (0,0) і (1,1) на керуючі входи (R, S). Так, при подачі вхідного набору (0,0) RS-засувка перейде в стан (1,1). При подальшій подачі набору (1,1) на входи (R, S) і при абсолютній симетрії провідних ліній, які формують зворотний зв'язок, на виходах Q і nQ можна буде спостерігати послідовне перемикання значень сигналів з 0 в 1. Однак на практиці RS-засувка в такому випадку набуде одого з двох стабільних станів: (Q, nQ) = (1,0) або (Q, nQ) = (0,1). Вибір конкретного стану буде залежати від внесеної асиметрії при виготовленні даного елемента пам'яті – асиметрії двох логічних елементів 2I-HI та з'єднувальних ліній.

Розглянутий заборонений перехід на керуючих входах RS-засувки може бути розглянутий як емуляція підключення напруги живлення на запам'ятовуючий елемент СОЗП і відповідно може бути використаний як схемна реалізація фізично неклонованої функції. Для забезпечення такої реалізації в роботі [115] була запропонована схема RS-засувки з об'єднаними керуючими входами R і S (рис. 5.27).

Розглянемо параметричну модель поданої схеми PUF на основі RS-засувки, яка дозволяє моделювати затримки поширення сигналів у провідних лініях, що беруть участь у з'єднанні двох елементів 2I-HI (дод. 4, лістинг 5.11). Припустимо, що для поданої моделі затримки поширення сигналів по наступних провідних лініях рівні: від вхідного порту RS до відповідних входів логічних елементів; від виходів елементів до вихідних портів Q і nQ. Домовимося, що тимчасові інтервали спрацьовування двох елементів 2I-HI є рівними.

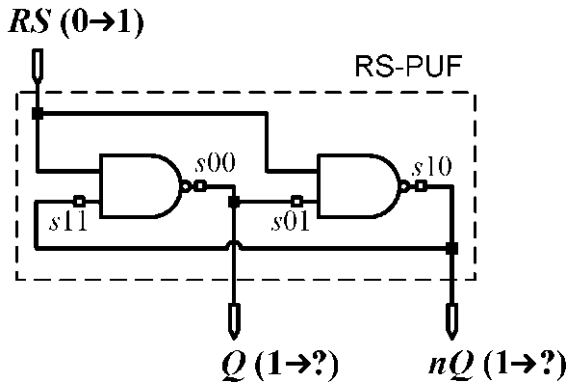



Рис. 5.27. PUF на основі RS-засувки

Припустимо, що час поширення сигналу по провідній лінії, що з'єднує вихід елемента NAND20 зі входом елемента NAND21 (вузли s00, s01), менше часу поширення сигналу по провідній лінії, що з'єднує вихід NAND21 зі входом NAND20 (вузли s10, s11). У цьому випадку після здійснення переходу сигналу зі стану 0 в 1 на вході RS, на виході Q встановиться значення 0, а на виході nQ - значення 1. При зворотній умові нерівності затримок поширення сигналів на виходах встановляться протилежні значення (Q, nQ) = (1,0). У гіпотетичному випадку рівності розглянутих затримок на виходах схеми будуть вироблятися імпульси з періодом, рівним сумарній величині затримок на двох провідних лініях.

Для збільшення діапазону зміни випадкових значень затримок і відповідно, стабільності і надійності фізично неклонованих функцій у роботі [115] була запропонована комбінована реалізація на основі PUF типу арбітр і PUF RS-засувки. У результаті комбінована схема набуває такого вигляду, як на рис. 5.28.

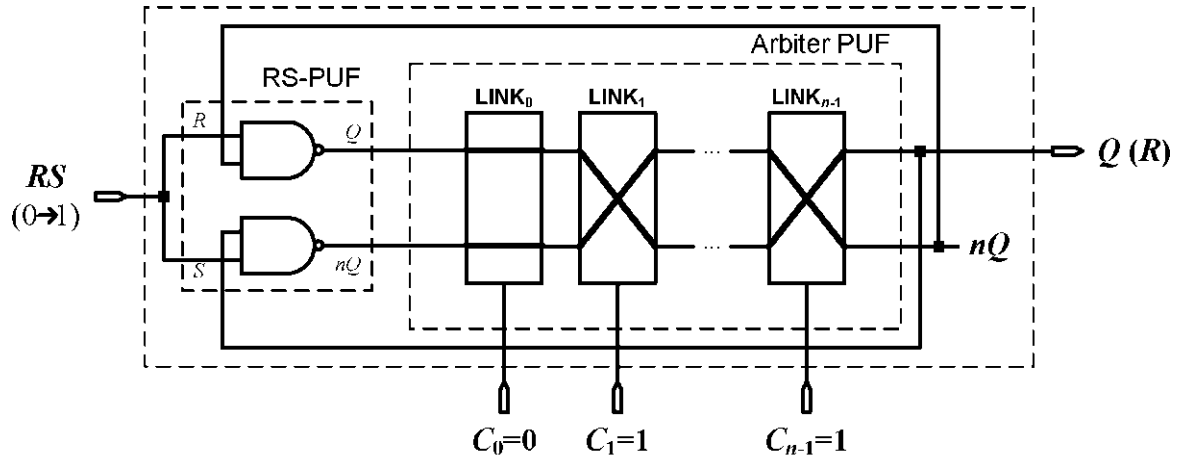



Рис. 5.28. Схема комбінованої PUF

Основною ідеєю в запропонованій комбінованій схемі є збільшення шляху між виходом одного з двох елементів 2I-HI RS-засувки і входом іншого елемента 2I-HI.

Довжина шляхів при цьому збільшується за рахунок послідовно включених n елементів LINK (рис. 5.16). При такій схемній реалізації затримка поширення сигналів залежить не тільки від технологічних варіацій під час виробництва елементів

схеми, а й від значення n-розрядного запиту C, який визначає конфігурацію шляхів проходження сигналів. Значення відповіді R на певний запит C формується на виході компоненти LINKn-1.

Ще одним рішенням, запропонованим у роботі [115], є комбінована PUF на основі кільцевих генераторів і PUF типу арбітр (рис. 5.29).

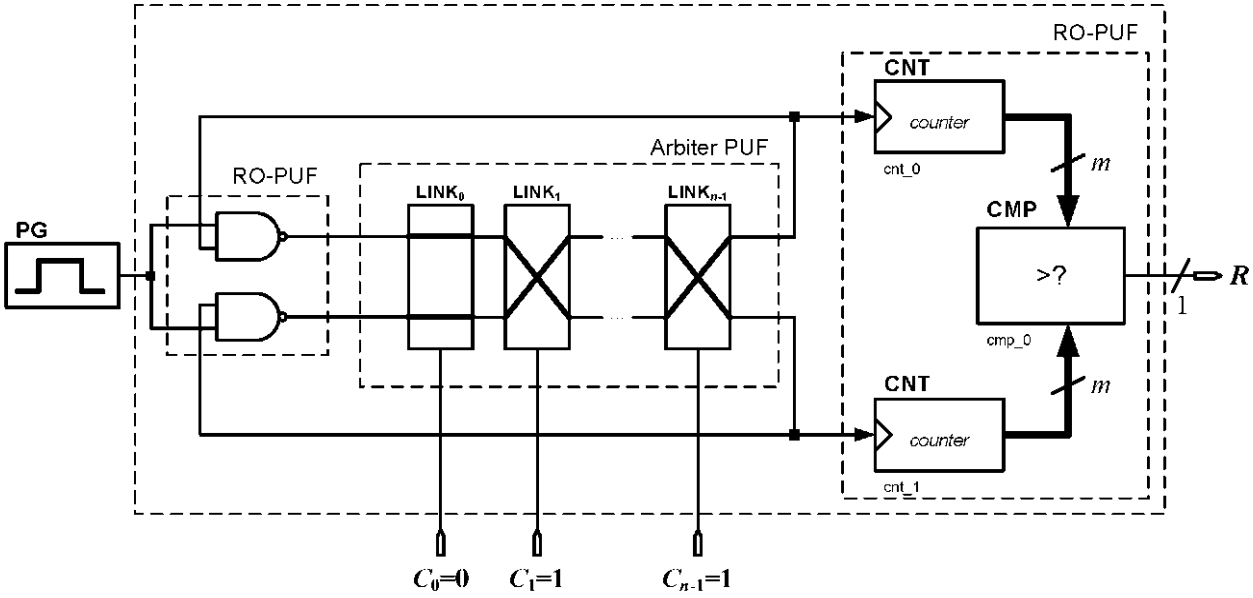



Рис. 5.29. Схема комбінованої PUF на основі кільцевих генераторів і PUF типу арбітр

Розглянемо функціонування наведеної схеми. Для заданого n-розрядного запиту C конфігуруються два шляхи, що утворюють негативні зворотні зв'язки для двох кільцевих генераторів, побудованих на елементах 2I-HI. Сформована таким чином унікальна пара генераторів виробляє високочастотні імпульси, які подаються на входи синхронізації двох лічильників cnt_0 і cnt_1. Кінцеві значення лічильників порівнюються, і на основі отриманого результату порівняння виробляється однобітна відповідь R. Часове вікно вимірювання кількості згенерованих імпульсів визначається тривалістю одиничного сигналу, який формується зовнішнім генератором PG. Слід зазначити, що схемні реалізації модифікованих PUF, наведені на рис. 5.28 і 5.29, можуть бути спроектовані із застосуванням тристабільних буферів (рис. 5.21).

Стандартна схемна реалізація PUF типу арбітр дозволяє судити про асиметричності двох сконфігурованих шляхів за допомогою аналізу переднього фронту сигналу. У загальному випадку затримка сигналу d на логічному елементі визначається за формулою d = (tr + tf) / 2, де tr – тривалість переднього фронту вихідного сигналу, а tf - заднього фронту. Тривалість фронтів tr і tf цілком залежить від опору елемента і вихідної паразитної ємності, які у свою чергу визначаються геометричними розмірами транзисторів, використовуваних для його виготовлення. Відповідно невеликі відхилення в геометрії транзисторів призводять до збільшення або зменшення затримки сигналу d і зміни його шпаруватості.

Для збільшення певності PUF типу арбітр у роботі [116] була запропонована схемна модифікація арбітра, що дозволяє визначати не тільки нерівність шляхів, а й нерівність шпаруватості двох сигналів. На рис. 5.30 подана схема модифікованого арбітра.

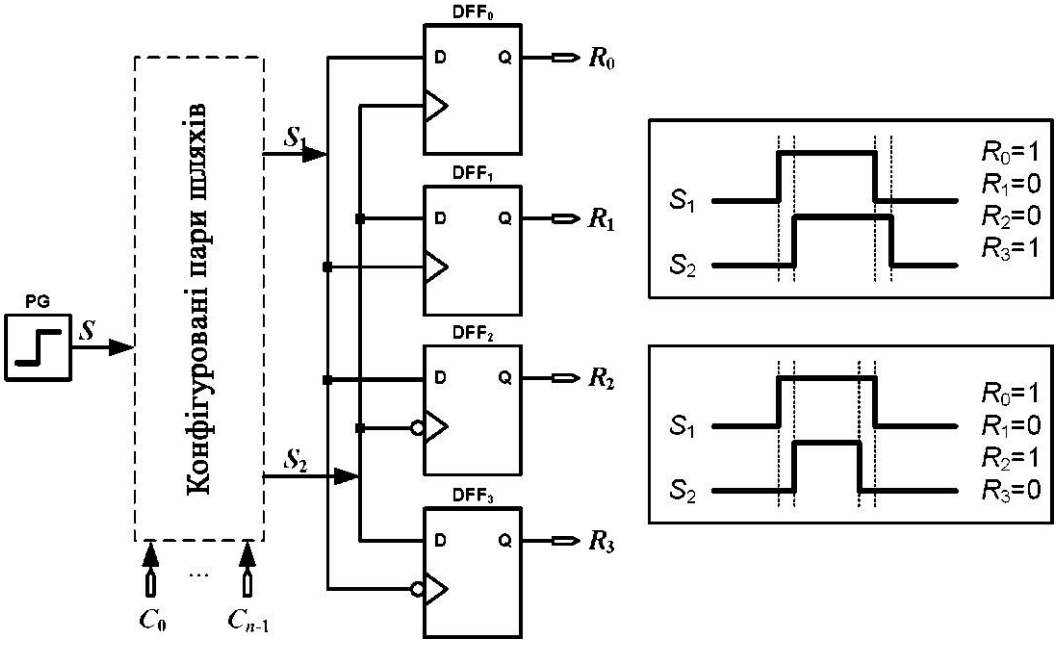



Рис. 5.30. Схема PUF з модифікованим арбітром

Наведена схема еквівалентна описаній вище схемі PUF (рис. 5.14), за винятком реалізації арбітра. Схема модифікованого арбітра складається з чотирьох D-тригерів (DFF0 - DFF3), два з яких стробуються заднім фронтом сигналу синхронізації (DFF2 і

DFF3). Два сигнали Si і S2, що пройшли пару шляхів сконфігуровані запитом C, надходять на інформаційні входи D і входи синхронізації чотирьох тригерів так, як це показано на рис. 5.30. На виходах тригерів формується чотирирозрядна відповідь R = (R3, R2, Ri, Ro).

Припустимо, що сигнал Si пройшов більший шлях порівняно з сигналом S2, при цьому сигнали мали однакову шпаруватість. У цьому випадку арбітром буде сформована відповідь R = (1,0,0,1).

Якщо шпаруватість сигналу S2 зменшилася (рис. 5.30), то арбітр згенерує відповідь R = (1,0,1,0). Слід зазначити, що при використанні стандартної схеми арбітра (один D-тригер) в обох розглянутих випадках сформується одна і та ж відповідь R = 1.

Розглянуті варіанти схемних реалізацій фізично неклонованих функцій можуть бути застосовані для вирішення різних завдань: генерування «відбитків пальців» (digital fingerprinting) і «водяних знаків» (watermarking) [117] цифрових пристроїв захисту від нелегального використання (у разі незаконного клонування bit-образу для ПЛІС) [118], виявлення шкідливих апаратних закладок у цифрових пристроях [119], генерування істинно випадкових чисел (true random numbers) [120] тощо.

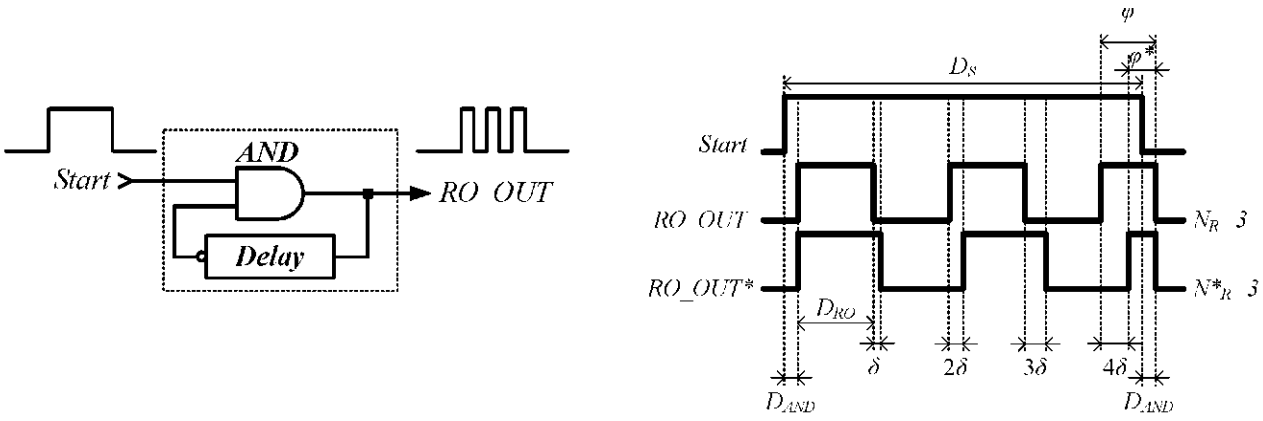
Однією з технічних проблем, з якими стикаються розробники цифрових пристроїв на базі ПЛІС, є проектування регістрів унікальних ідентифікаторів. У наступному підрозділі розглянемо деякі варіанти реалізації таких регістрів.

## 5.5. Ідентифікація ПЛІС

Одним з альтернативних методів реалізації унікальних ідентифікаторів FPGA (у тому числі і секретних ключів) є застосування фізично неклонованих функцій. Так, у роботі [122] була запропонована методика побудови цифрових ідентифікаторів ПЛІС, заснована на реалізації модифікованого методу RO-PUF (Ring Oscillator PUF), що полягає в порівнянні кількості цифрових імпульсів, що виробляються двома функціонально ідентичними кільцевими генераторами.

### 5.5.1. Аналіз кільцевого генератора цифрових імпульсів

Як було описано вище, функціональна схема кільцевого генератора являє собою непарну кількість послідовно з'єднаних інверторів, при цьому вихід останнього інвертора, який є виходом самого генератора, з'єднаний зі входом першого, утворюючи ланцюг зворотного зв'язку (кільце). Для визначення конкретного початкового стану генератора і забезпечення його керованості можна додати в ланцюг зворотного зв'язку двовходовий логічний елемент І, перший вхід якого є входом дозволу функціонування генератора, а другий вхід бере участь у формуванні кільця. На рис. 5.31, а подана узагальнена функціональна модель цифрового кільцевого генератора імпульсів.




а б

Рис. 5.31. Кільцевій генератор імпульсів: а – узагальнена функціональна модель; б – основні часові параметри

Зображений генератор має вхідну лінію сигналу дозволу функціонування Start і вихідну лінію цифрових імпульсів RO_OUT. При нульовому значенні сигналу Start генератор знаходиться в стані очікування, при цьому сигнал на вихідній лінії набуває також нульового значення. Функціонування генератора починається при перемиканні рівня сигналу Start з значення 0 в значення 1 і триває весь період утримання в значенні 1. Кількість вироблюваних імпульсів на виході RO_OUT знаходиться в прямій залежності від тривалості утримання сигналу на вході Start в значенні 1 і затримки поширення вихідного сигналу по ланцюгу зворотного зв'язку, що включає

логічний вентиль AND і елемент затримки Delay (множина інверторів). Позначимо часовий інтервал утримання сигналу дозволу як DS, а затримку поширення сигналу в ланцюзі зворотного зв'язку генератора як Dro = DaND + DDelay.

Припустимо, що підрахунок імпульсів, що виробляються генератором, проводиться цифровим двійковим лічильником, стробований фронтом вихідного сигналу RO_OUT. Якщо DRO> DS, генератор буде виробляти один імпульс тривалістю DS. У цьому випадку реєстроване значення на лічильнику дорівнюватиме NR = 1. За умови, що 2DRO <DS, лічильник буде реєструвати значення NR> 1. Виразимо значення DS через часові параметри генератора:

$$D_S = D_{AND} + 2D_{RO}(N_R - 1) + \varphi - D_{AND} = 2D_{RO}(N_R - 1) + \varphi, \quad (5.8)$$

де значення $\varphi$ задовольняє такі нерівності:

$$D_{AND} < \varphi < (2D_{RO} - D_{AND}).$$

Будемо розглядати значення DS як тривалість часового вікна вимірювання кількості зареєстрованих імпульсів NR.

Для двох порівнюваних генераторів, реалізованих на різних ПЛІС, часове вікно вимірювань має бути ідентичним і легко масштабованим. Припустимо, що сигнал Start виробляється для кільцевого генератора на основі зовнішнього стабільного і технологічно незалежного джерела синхронізації, яким, наприклад, може служити кварцовий осцилятор.

Домовимося, що осцилятор генерує сигнал синхронізації у формі меандра з частою FCLKG. Тоді мінімальне значення DS часу утримання сигналу Start можна прийняти рівним 1 / FCLKG.

У цьому випадку для збільшення значення DS можна використовувати апаратні подільники частоти FCLKG, серед яких найбільш переважними виглядають подільники з коефіцієнтом ділення k = 2, реалізовані на довічних лічильниках. При цьому часове вікно вимірювання кількості імпульсів може бути лінійно масштабованим відносно значення Ds.

Наприклад, для збільшення часового вікна вимірювання в чотири рази (k = 4) можна використовувати дворозрядний двійковий лічильник, на вхід синхронізації якого надходить сигал

з частотою FCLKG = 1 / DS, а на виході старшого розряду лічильника формується сигнал Start тривалістю 4DS.

Припустимо, що вимірювання кількості імпульсів проводиться в часовому інтервалі, кратному значенню DS. У цьому випадку вираз (5.8) буде виглядати як

$$kD_S = 2D_{RO}(N_R(k) - 1) + \varphi(k), \tag{5.9}$$

де $k$ – натуральне число, що визначає коефіцієнт масштабування DS;

$NR(k)$ — кількість зареєстрованих імпульсів;

$\varphi(k)$ — часовий інтервал, вимірюваний від фронту останнього імпульсу і до моменту закінчення вікна вимірювання.

Покажемо, що значення k безпосередньо впливає не тільки на кількість реєстрованих імпульсів кільцевих генераторів, а й на достовірність ідентифікації мікросхем ПЛІС.

Виразимо кількість реєстрованих імпульсів у вікні вимірювання тривалістю kDS:

$$N_R(k) = \left[\frac{kD_S - \varphi(k)}{2D_{RO}}\right] + 1. \tag{5.10}$$

Припустимо, що два генератори цифрових імпульсів, реалізованих на різних ПЛІС, мають незначні відмінності, які можна виразити збільшенням затримки поширення сигналу по ланцюгу зворотного зв'язку DRO на деяку малу величину 6 C DRO, DRO = DRO + 6 (рис. 5.31). Тоді відповідно до виразу (5.10) кількість реєстрованих імпульсів другого генератора виражається як

$$N_R^*(k) = \left[\frac{kD_S - \varphi^*(k)}{2(D_{RO}+\delta)}\right] + 1. \tag{5.11}$$

При цьому абсолютне значення різниці кількостей реєстрованих імпульсів двох генераторів оцінюється виразом

$$\Delta N_R(k) = |N_R(k) - N_R^*(k)| == \left|\left[\frac{kD_S - \varphi(k)}{2D_{RO}}\right] - \left[\frac{kD_S - \varphi^*(k)}{2(D_{RO}+\delta)}\right]\right| \approx$$

$$\approx \left|\left[\frac{\varphi^*(k) - \varphi(k)}{2D_{RO}}\right]\right| \tag{5.12}$$

за умови, що $\delta \ll D_{RO}$. Різницю значень $\varphi^*(k) - \varphi(k)$ можна подати як $2\delta(N_R(k) - 1)$. З урахуванням, що $N_R(k) \approx \left[\frac{kD_S}{2D_{RO}}\right] + 1$, вираз (5.12) набуде вигляду

$$\Delta N_R(k) \approx \left|\left[\frac{\varphi^*(k)-\varphi(k)}{2D_{RO}}\right]\right| \approx k\left[\frac{\delta D_S}{2D^2{}_{RO}}\right]. \qquad (5.13)$$

Очевидно, що при малих значеннях k різниця з виразу (5.13) буде набувати нульових значень. Встановимо експериментально, при яких значеннях k вона набуватиме стабільних ненульових значень.

### 5.5.2. Проектування конфігурованого генератора імпульсів

Для проведення експерименту були обрані ідентичні цифрові системи Digilent Nexys-2 [45], до складу яких входять ПЛІС Xilinx SPARTAN-3E (XC3s500e-5FG320) [31], виготовлені за 90 нм CMOS технологічним процесом. Для можливості реалізації генераторів імпульсів з різними значеннями DRO була запропонована функціональна цифрова модель, структурна схема якої наведена на рис. 5.32.

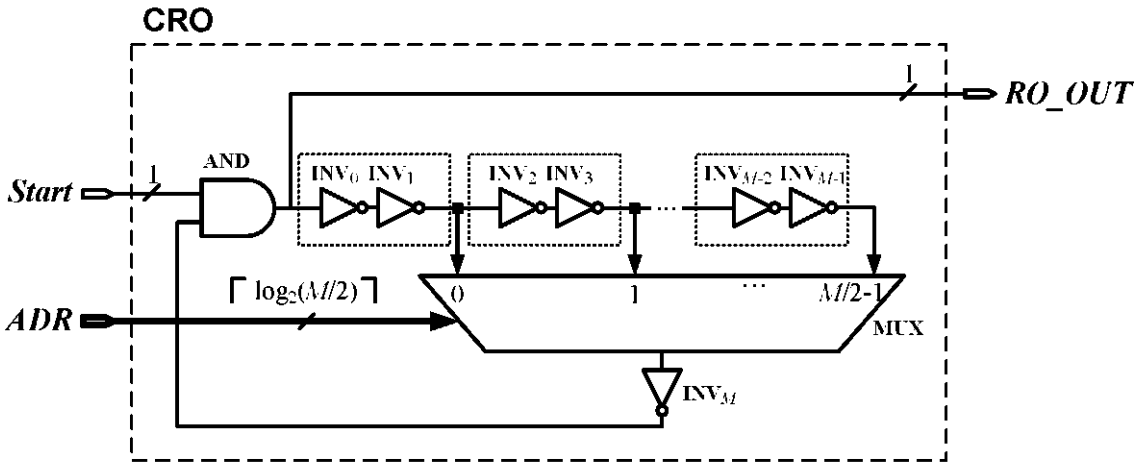



Рис. 5.32. Структура конфігурованого кільцевого генератора

Ланцюг зворотного зв'язку генератора містить M інверторів, які попарно згруповані. Вихід кожної пари підключений до

відповідного входу мультиплексора MUX. Комутація M/2 входів мультиплексора з його вихідним портом визначається двійковим значенням, що подається на log2(M/2)]-розрядну адресну шину ADR. Вихід мультиплексора з'єднаний з інверсним входом логічного вентиля AND, вихід якого є вихідним портом генератора RO_OUT. Таким чином, подана схема реалізує M/2 кільцевих генераторів з різними фіксованими значеннями DRO.

У роботі [122] було запропоновано використовувати схему конфігурованого генератора імпульсів як схему, що реалізує фізично неклоновану функцію, аргументами якої є тривалість імпульсу на вході Start і значення на вхідній шині ADR. Значенням фізично неклонованої функції є кількість зареєстрованих імпульсів на вихідному порту RO_OUT

$$PUF_{CRO}(kD_S, ADR) = N_R(k). \quad (5.14)$$

Для апаратної реалізації поданої структури конфігурованого генератора імпульсів було спроектовано цифровий пристрій, що дозволяє встановлювати різні фіксовані значення kDS і ADR і з можливістю реєстрації вироблюваних значень NR(k). Пристрій містить такі основні блоки: генератор стартового імпульсу SPG, конфігурований генератор цифрових імпульсів CRO, лічильник імпульсів CNT, контролер світлодіодних індикаторів SLC. Пристрій, що реалізується на ПЛІС FPGA XC3s500e-5FG320, управляється генератором системних імпульсів CLKG (Fclkg = 50 MHz), апаратними кнопками BTN1 (джерело асинхронного сигналу ініціалізації) і BTN2 (управління режимом відображення результату), трьома перемикачами SW (2: 0), що входять до складу цифрової системи Digilent Nexys-2 (рис. 5.33).

Наведена структура конфігурованого генератора була спроектована і описана мовою VHDL з урахуванням конструктивних особливостей системи Digilent Nexys-2 за допомогою САПР Xilinx ISE WebPack 13.1 [83]. За результатами синтезу VHDL-опису, апаратура генератора імпульсів займає 97 Slice-блоків кристала Xilinx SPARTAN-3E, що становить близько 2 % усіх ресурсів даної ПЛІС (46 Slice-блоків при цьому було використано для реалізації контролера семисегментних індикаторів SLC).

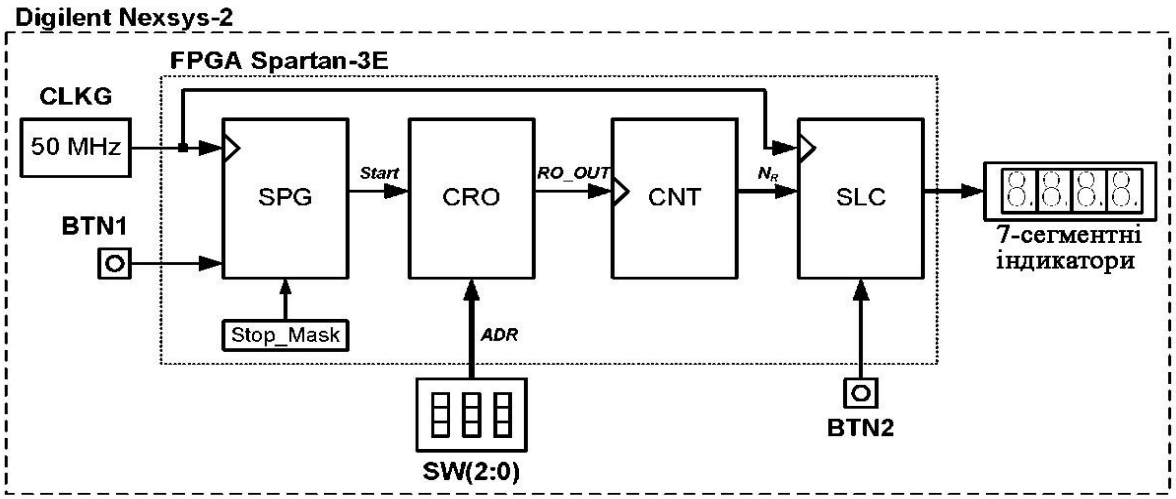



Рис. 5.33. Структура реалізованого цифрового пристрою в системі Digilent Nexys-2

### 5.5.3. Постановка експерименту і аналіз результатів

Для проведення експерименту були взяті дві ідентичні системи В і В* (Digilent Nexys-2 з ПЛІС FPGA XC3s500e-5FG320), для яких були визначені такі обмеження.

Живлення обох систем (+3,3 В) здійснювалося від одного джерела живлення системного блока ПЕОМ класу Pentium Dual-Core за допомогою телекомунікаційних каналів USB (+5,0 В) та ідентичних регуляторів напруги LTC1765, що входять до складу систем В і В*.

Обидві ПЛІС були сконфігуровані однією бітовою послідовністю, згенерованою САПР ISE WebPack на основі загального VHDL-проекту при абсолютно однакових параметрах синтезу, розміщення, трасування з'єднань і конфігурації.

Процедура конфігурації ПЛІС здійснювалася за допомогою програмного забезпечення Digilent Adept 2.5.

Вироблення системної частоти, на підставі якої здійснювалося генерування сигналу Start, проводилось ідентичними генераторами AGXO 75IL (50 МГц).

Процеси ініціалізації і функціонування конфігурованих генераторів імпульсів здійснювалися одночасно.

Узагальнюючи вищесказане, можна зробити припущення про рівні зовнішні умови проведення експерименту для двох функціонально ідентичних цифрових систем. Експеримент

проводився в кілька етапів. На першому етапі для обраних значень k = 2r (i = 0 ... 10) здійснювався моніторинг кількості згенерованих імпульсів NR (k) двома генераторами, що конфігуруються. На рис. 5.34 подані значення різниці кількості зареєстрованих імпульсів залежно від параметра k і всіх можливих восьми значень ADR.

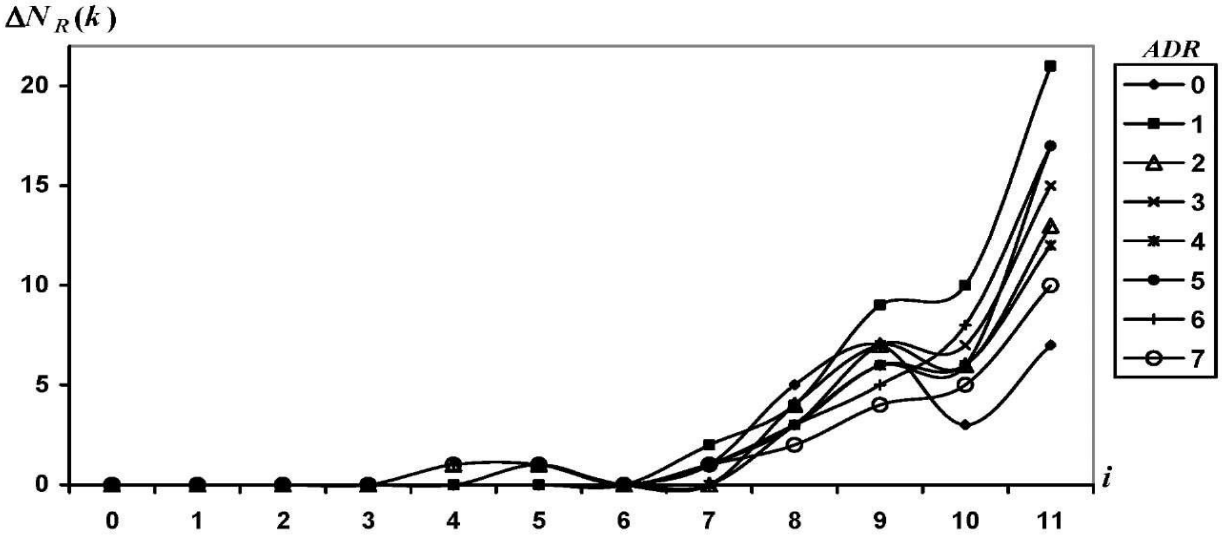



Рис. 5.34. Значення різниці ANR (k) для k = 2r і всіх можливих значень ADR

Як видно з наведеного графіка, першого ненульового значення різниця імпульсів набуває для k = 4 і ADR = {2, 7}. При цьому кількість імпульсів для значення ADR = 2 одно NR (16) = 23 і NR (16) = 22.

Зі збільшенням значення k спостерігається лінійне збільшення різниці реєстрованих імпульсів для всіх значень ADR.

У табл. 5.4 наведені експериментальні дані, отримані для значення k = 210 (kDS = 210 * 20 = 20480 нс). Тут і далі по тексту значення реєстрованих імпульсів подано в шістнадцятковому форматі.

Значення зареєстрованих імпульсів NR (k) і NR (k) відповідають першій B і другій B* цифровій системі. Значення арифметичної різниці імпульсів позначено як DA(k), а значення відстані Хеммінга – як HA(k). Для даного експерименту середні значення арифметичної різниці і відстані Хеммінга дорівнює відповідно [DA(k)J =7 та [HZ(k)J = 2.

Таблиця 5.4

| Параметр | ADR | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Nn(k) | 7FC | 671 | 55D | 4BB | 40F | 396 | 33D | 2F2 |
| NR (k) | 7F8 | 666 | 557 | 4B3 | 408 | 38F | 334 | 2ED |
| DA(k) | 4 | 11 | 6 | 8 | 7 | 7 | 9 | 5 |
| HA(k) | 1 | 4 | 2 | 1 | 3 | 3 | 2 | 5 |

Експериментальні дані для kDS = 20480 нс.

Далі для кожного значення ADR і вибраного kDS = 20480 нс були проведені 100 послідовних експериментів для виявлення відхилень у реєстрованих значеннях NR (k) і NR (k). Відхилення в реєстрованих значеннях можливі через наявність технологічних варіацій і вироблення генераторами імпульсів з частотою, що перевершує робочу частоту функціонування ПЛІС [132, 133]. Так, для значень ADR = 0 і k = 210 реєструється близько 2040 імпульсів, що відповідає частоті 100 МГц, що вдвічі перевищує частоту функціонування XC3s500e-5FG320.

Як еталонне реєстроване значення кількості імпульсів були обрані значення, що найбільш зустрічаються при 100 послідовних експериментах. Слід зазначити, що для всіх можливих значень ADR абсолютна величина відхилень не перевищувала 2 (рис. 5.35).

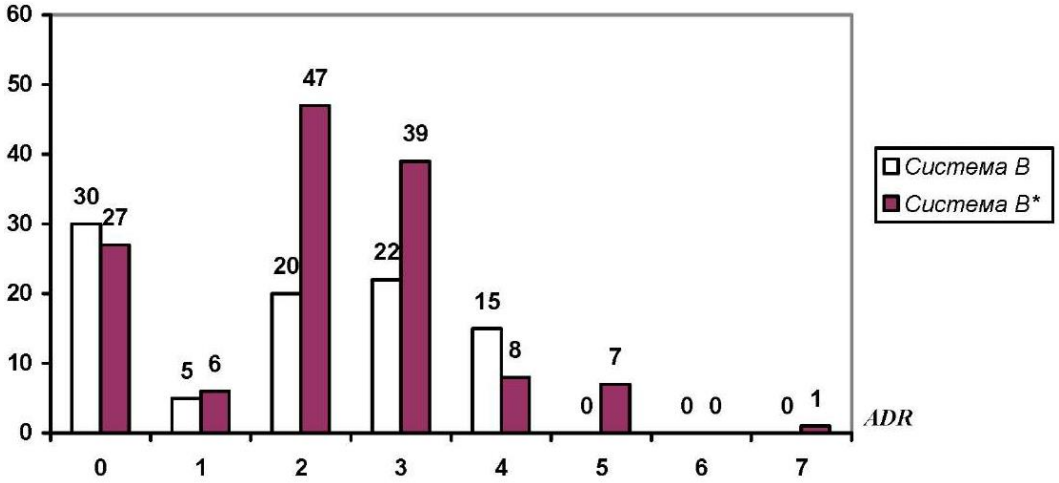



Рис. 5.35. Розподіл кількості відхилень від еталонного значення реєстрованих імпульсів

Наприклад, для значення ADR = 4 генератор імпульсів, реалізований на системі B, має такий розподіл значень: 410 (85 %), 40F (10 %), 411 (5 %), а генератор, реалізований на системі B*, - 409 (92 %), 40A (7 %), 408 (1 %). З урахуванням отриманих даних сформуємо карти ймовірностей появи символу 1 на 11 значущих розрядах двійкових уявлень чисел NR (k) і NR (k) (табл. 5.5).

Таблиця 5.5

Карти ймовірностей появи символу 1

| Параметр | Номер двійкового розряду | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Nr (k) | 1,0 | 0 | 0 | 0 | 0 | 0 | 0,9 | 0,1 | 0,1 | 0,1 | 0,15 |
| NR (k) | 1,0 | 0 | 0 | 0 | 0 | 0 | 0 | 1,0 | 0 | 0,07 | 0,92 |

Значення зареєстрованих імпульсів, у тому числі і значення ймовірностей появи символу 1 на різних позиціях, можуть бути використані для побудови унікального ідентифікатора ПЛІС, що реалізує конфігурований генератор цифрових імпульсів. Наприклад, наведену карту ймовірностей перетворимо у відповідності з мажоритарним принципом, при якому всі значення ймовірності менше 0,5 подамо як символ 0, а інші - символ 1. При цьому карти ймовірностей трансформуються в двійкові ідентифікатори (табл. 5.6).

Таблиця 5.6

Двійкові ідентифікатори цифрових систем

| Ідентифікатор | Номер двійкового розряду | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ID(B) | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| ID(B*) | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |

Як видно з табл. 5.6, значення ідентифікаторів ID (B) і ID (B*) помітні на трьох розрядах, що корелює з експериментальними даними, наведеними в табл. 5.4. Реалізація подібного алгоритму обчислення двійкових ідентифікаторів може бути не завжди виправдана з точки зору як апаратурних, так і часових ресурсів.

Як альтернативу можна використовувати алгоритм, заснований на збільшенні значення DS і розрядності лічильника CNT (рис. 5.33), при яких можна виділити фіксовані виконавчі розряди чисел NR (k) і NR (k) з одиничними значеннями ймовірності появи символів 1 і 0.

Експериментально було встановлено, що для значення $kD_S = 2^{25} * 20 \simeq 0{,}671$ с 26-розрядного лічильника CNT і різних значень ADR десять старших двійкових розрядів чисел NR (k) і NR (k) мають одиничні ймовірності появи символів 0 і 1 на фіксованих позиціях (табл. 5.7).

Для даного експерименту середні значення арифметичної різниці і відстані Хеммінга відповідно $\lfloor \overline{D_\Delta}(k) \rfloor = 13$ та $\lfloor \overline{H_\Delta}(k) \rfloor = 3$.

Таблиця 5.7

Експериментальні дані для $kD_S \simeq 0{,}671c$ $(k = 2^{25})$

| Параметр | ADR | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Nr (k) | 3D9 | 34E | 2CE | 26E | 232 | 1E5 | 1A9 | 187 |
| NR (k) | 3C8 | 340 | 2C1 | 260 | 224 | 1D9 | 19E | 17Б |
| Da (k) | 17 | 14 | 13 | 14 | 14 | 12 | 11 | 12 |
| Яд (k) | 2 | 3 | 4 | 3 | 3 | 4 | 5 | 6 |

Проведені експерименти наочно демонструють можливість застосування конфігурованих генераторів цифрових імпульсів для побудови унікальних ідентифікаторів ПЛІС. Для наближеного оцінювання ступеня відмінності ідентифікаторів для всіх функціонально ідентичних ПЛІС з класу XC3s500e-5FG320 було здійснено тестове моделювання конфігурованого генератора цифрових імпульсів з урахуванням усереднених технологічних параметрів у симуляторі ISim, що входить до складу САПР Xilinx ISE. Для цього після успішно проведеного процесу розміщення і трасування (Place and Route) у меню Design була обрана опція View-Simulate, а як параметр цієї опції було зазначено Post-Route. Симулятор ISim здійснює параметричне моделювання цифрового проекту ПЛІС з можливістю спостереження значень сигналів в основних вузлах розміщеної схеми.

На рис. 5.36 подана гістограма розподілу кількості згенерованих імпульсів на реальних цифрових системах Б / Б * і зареєстрованих у симуляторі ISim за таких умов: ADR = 7, k = 2 %,i = 0...10.

Відповідно до отриманих експериментальних даних для значень $i > 7$ $(i = \lfloor log_2 k \rfloor)$ спостерігається стабільне збільшення арифметичної відстані $D^*_\Delta(k)$ між кількістю імпульсів, зареєстрованих у процесі моделювання, і кількістю імпульсів, зареєстрованих на реальних цифрових системах. Так, для значення $k = 2^7$ значення різниці $D^*_\Delta(k) = 25$, $D^*_\Delta(2^8) \approx 50, D^*_\Delta(2^9) \approx 100, D^*_\Delta(2^{10}) \approx 200$ і т. д.

Можна припустити, що зі збільшенням Δ до значення $D^*_\Delta(k)$ буде змінюватися відповідно до виразу $D^*_\Delta(k) \approx 25 \cdot 2^{\lfloor log_2 k \rfloor - 7} \approx \frac{25}{128} k, \forall k \geq 2^7$.

У свою чергу значення $D^*_\Delta(k)$ визначає кількість унікальних ідентифікаторів (кількість досліджуваних мікросхем ПЛІС) для обраного значення $k$. Наприклад, можна припустити, що для $k = 2^{11}$ близько 400 ПЛІС з класу XC3s500e-5FG320 можуть бути помітними. Таким чином, збільшення значення $k$ дозволяє підвищувати достовірність ідентифікації ПЛІС.

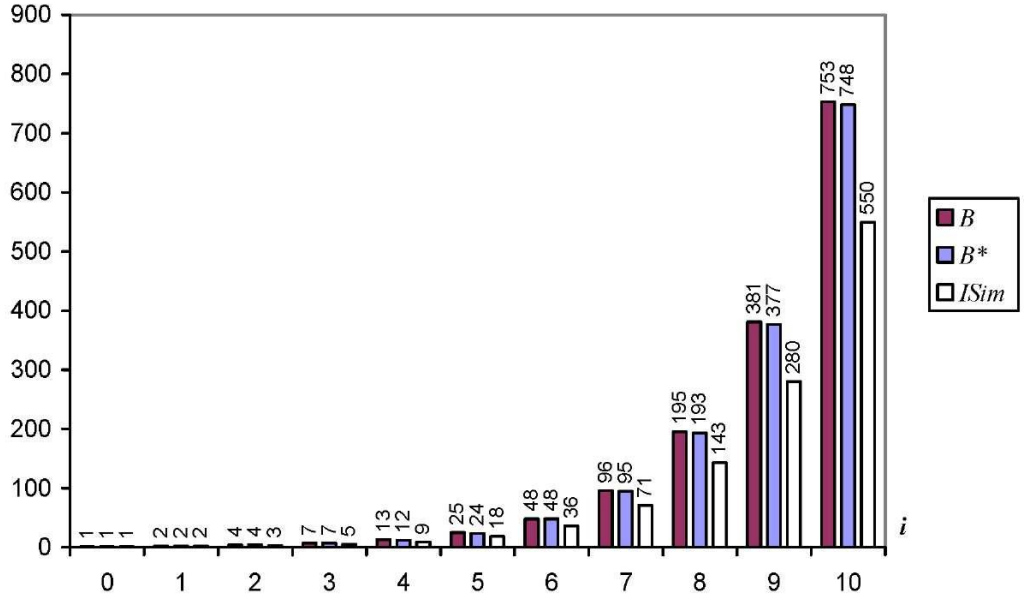



Рис. 5.36. Розподіл кількості реєстрованих імпульсів

**Контрольні питання**

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

## Висновки

У навчальному посібнику наведено огляд вбудованих систем, зокрема систем на кристалі. Особливу увагу приділено технологіям програмованих логічних інтегральних схем, що дозволяє реалізувати різні IP-компоненти цифрових пристроїв і систем.

Розглянуто особливості проектування та реалізації цифрових пристроїв на програмованих логічних інтегральних схемах типу FPGA з застосуванням високорівневої мови VHDL. Наведено методики використання різних стилів VHDL-описів для проектування базових компонент вбудованих цифрових пристроїв.

Розглянуто окремі питання синтезу цифрових пристроїв з мікропрограмного управління. Наведено основні аспекти проектування контролепридатних цифрових пристроїв, а саме методи проектування вбудованої апаратури самотестування і контролепридатного доступу. Подано проектні описи вбудованої апаратури граничного сканування і приклад їх впровадження до проектного опису цифрового пристрою.

Крім того, у посібнику наведено огляд сучасних методів захисту цифрових проектів і пристроїв від несанкціонованого використання і копіювання. Розглянуто методики заплутуючих перетворень проектних описів, впровадження «водяних знаків» і «відбитків пальців». Наведено огляд методів аутентифікації і ідентифікації цифрових пристроїв, реалізованих на ПЛІС. Показано, що перспективною технологією, що лежить в основі методів апаратної аутентифікації і ідентифікації цифрових пристроїв, є апаратна реалізація фізично неклоніруемих функцій. Розглянуто приклад використання фізично неклонованої функції кільцевого генератора для вирішення завдання ідентифікації FPGA.

У посібнику наведено велику кількість вихідних кодів VHDL-описів різних цифрових компонент і пристроїв, придатних для реалізації на програмованих логічних інтегральних схемах типу FPGA.

Подане все в посібнику є баченням авторів у частині теоретичних і практичних аспектів проектування вбудованих цифрових пристроїв і систем для програмованих логічних інтегральних схем. Посібник може бути корисним фахівцям у сфері проектування вбудованих цифрових систем.

## 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. Лістровий С. В., Мірошник М. А., Клименко Л. А. Теорія автоматичного керування, штучний інтелект і автоматизація процесу прийняття рішень: навч. посіб. Харьков: 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, Issue 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. Харьков: 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. A. 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. Мірошник М. А. Проектування діагностиної інфраструктури обчислювальних систем та пристроїв на ПЛІС: монографія. Харків: ХУПС, 2012. 188 с.

13. 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. Харьков: УкрДАЗТ, 2008. 119 с.

14. Мірошник М. А. Моделі і методи синтезу інтеллектуальної діагностичної інфраструктури розподілених компютерних систем: автореф. дис. … д-ра техн. наук. Харків, 2013. 26 с.

15. Методи і моделі планування ресурсів в GRID-системах: монографія / С. В. Лістровий, С. В. Мінухин, В. С. Пономаренко, С. В. Знахур. Харків: ІНЖЕК, 2008. 408 с.

16. Мірошник М. А., Лістровий С. В. Інформаційно-управляючі системи та організації паралельних обчислень: навч. посіб. Харків: УкрДУЗТ, 2015. 308 с.

17. Мікропроцесорні засоби в автоматизованих системах керування технологічними процесами / А. К. Бабіченко, І. Л. Красніков, Ю. А. Бабіченко та ін.; за ред. А. К. Бабіченка. Харків: Вид-во ТОВ «Водний Спектр Джі-Ем-Пі», 2016. 440 с.

18. Ситнік Б. Т. Інформаційні системи і технології на залізничному транспорті: навч. посіб. Харків: УкрДУЗТ, 2019. 168 с.

19. Каргин А. А. Введение в интелектуальные машины. Кн. 1. Регуляторы: монография. Донецк: Норд-Прес, ДонНУ, 2010. 526 с.

20. Проектирование цифровых устройств на ПЛИС: учебник / М. Л. Малиновский, И. А. Фурман, С. Я. Бовчалюк; под ред. И. А. Фурмана. Харьков : Факт, 2006. 162 с.

21. Иванюк А. А. Проектирование встроенных цифровых устройств и систем: монография. Минск: Бестпринт, 2012. 337 с.

22. Иванюк А. А. Использование физически неклонируемых функций типа «арбитр» для идентификации встроенных систем на базе ПЛИС. *Информационные технологии и системы 2011* (ИТС 2011): материалы междунар. науч. конф. Минск. 26 окт. 2011 / БГУИР: редкол.: Л. Ю. Шилин [и др.]. Минск, 2011. С. 270–271.

23. Иванюк А. А. Использование конфигурируемых генераторов импульсов для идентификации ПЛИС. *Информатика.* 2011. № 4. С. 113–123.

24. Хаханов В. І., Семенец В. В., Хаханова І. В. Проектування цифрових схем з використанням мови VHDL: навч. посіб. Харків: ХНУРЕ, 2003. 492 с.

25. Грушвицкий Р. И., Мурсаев А. Х., Угрюмов Е. П. Проектирование систем на микросхемах программируемой логики. Санкт-Петербург: БХВ–Петербург, 2002. 608 с.

26. Ivaniuk A. A. Optimal Memory Tests Coding for Programmable BIST Architecture. *Radioelectronics and Informatics.* 2008. № 4. P. 32–37.

27. Design automation of easy-tested digital finite state machines / M. A. Miroschnyk, A. S. Shkil, E. N. Kulak, D. Y. Kucherenko, Y. V. Pakhomov. *Radio Electronics, Computer Science, Control*: Zaporizhzhia National Technical University, 2018 № 2. С. 117-124.

28. Аналіз апаратурних витрат при тестопридатному проектуванні керуючих цифрових автоматів / М. А. Мірошник, Ю. В. Пахомов, О. С. Шкіль та ін. *Вісник СНУ ім. В. Даля.* 2018. № 6. С. 101–109.

29. Проведення діагностичних експериментів в керуючих автоматах з використанням синхронізуючих послідовностей / А. С. Шкіль, М. А. Мірошник, Ю. В. Пахомов, Д. Г. Караман. *Радиотехника и информатика.* 2018. № 3. С. 83-90.

30. Miroshnyk M. A., Pakhomov Yu. V. Model of automated hardware diagnostics of remote energy systems management points. *Світлотехніка та електроенергетика.* 2017. № 3. С. 3-9.

31. Мірошник М. А. Методи автоматизованого комп'ютерного проектування гетерогенних комп'ютерних мереж критичного застосування. *Інформаційно-керуючі системи на залізнич-*

*ному транспорті.* 2019. № 4. С. 3-8. URL: 10.18664/ ikszt.v0i4.178719.

32. Мірошник М. А., Корольова Я. Ю. Ітераційні алгоритми компонування в конструкціях мультимедіа. *Інформаційно-керуючі системи на залізничному транспорті.* 2019. № 2. С. 3-8. URL: 10.18664/ikszt.voi1.158795.

33. Мірошник М. А., Клименко Л. А. Методи автоматизованого комп'ютерного проектування цифрового пристрою локального управління. *Інформаційно-керуючі системи на залізничному транспорті.* 2019. № 1. С. 11-18. URL: 10.18664/ikszt.voi1.158795.

34. 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.

35. 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.

36. Listrovoy S. V., Minykhin S. V. General Approach to Solving Optimization Problems in Distributed Cjmputing Systems and Theory of Intelligence Systems Construction. *Journal of Automation and Information Science.* 2010. Vol. 42, N. 3. P. 30–45.

37. Listrovoy S. V., Butenko V. M. Algoritm of Sub Exponential Complexity for the SAT. *International Jornal of Computer and Information Technology* (ISSN: 2279–0764). September 2013. Vol. 2, Issue 05. P. 211–215.

38. 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.

39. 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.

40. Listrovoy S. V. Jn Cjrrelation of P and NP–Classes. *I.J. Modern Education and Computer Science.* 2012. № 3. P. 21–27.

41. 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.

42. Verification of automatic logic control systems on FPGA by analyzing the correctness of state diagrams of control finite state machines / M. A. Miroshnik, A. S. Shkil, E. N. Kulak, I. V. Filippenko, D. Y. Rahklis. *11th IEEE International Conference on Dependable Systems, Services and Technologies* (DESSERT'2020). May 14. 2020. P. 218.

43. Design timed FSM with VHDL Moore pattern / M. A. Miroshnik, O. S. Shkil, E. N. Kulak, I. V. Filippenko, D. Y. Rakhil, A. Miroshnyk. *Radioelectronics, Informatics, Management.* Zaporozhye National Technical University, 2020. № 2.

44. Timed logical control FSM in hardware / M. A. Miroshnik, O. S. S hkil, E. N. Kulak, I. V. Filippenko, D. Y. Rakhil, A. Miroshnyk. *IEEE CMIS'2020.* April 04. 2020.

45. Miroshnik M., Zaichenko O., Galkin P. Testbench for Kalman Filter on FPGA for Microwave Measurement. *TCSET'2020*. February 02. 2020.

46. Testable design of control digital automatic machines / M. Miroshnyk, O. Shkil, D. Rakhlis, I. Filippenko, E. Kulak, A. Miroshnyk. *2020 IEEE 15th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET), 25-29 Feb. 2020, Lviv-Slavske, Ukraine.* URL: https://ieeexplore.ieee.org/document/9088670. 07.05.2020 р. – Загол. з екрана.

47. Testable design of control digital automatic machines / M. A. Miroshnik, O. S. Shkil, E. N. Kulak, I. V. Filippenko, D. Y. Rakhil, A. Miroshnyk. *IEEE TCSET'2020.* February 02. 2020.

48. Miroshnik M., Zaichenko O., Galkin P. Model and algorithms for microwave multi-port receiver. *PICST-2019.*

49. Synchronizing Sequences For Verification Of Finite State Machines. / M. A. Miroshnik, O. S. Shkil, E. N. Kulak, I. V. Filippenko, D. Y. Rakhil, Yu. Pahomov, A. Miroshnyk. *Synchronizing Sequences For Verification Of Finite State Machines*. September 9-13, 2019, Odessa, Ukraine. 6 p.

50. Analysis of the state diagram correctness of automatic logic control systems on FPGA Paper / M. A. Miroshnik, O. S. Shkil, E. N. Kulak, I. V. Filippenko, D. Y. Rakhil. *29-th International Scientific Symposium «Metrology and Metrology Assurance».* Sozopol, Bulgaria, Sept. 10–14, 2019. 202 p. URL: https:// https://ieeexplore.ieee.org/document/8935958. 01.04.2020 р. Загол. з екрана. DOI: 10.1109/MMA.2019.8936027.

51. Miroshnik M. A., Korolova Ya. Y., Pakhomov Y. V. Graphic Representation of Component Algorithms in Multimedia Devices. *29-th International Scientific Symposium «Metrology and Metrology Assurance».* Sozopol, Bulgaria, Sept. 10–14, 2019. 6 p. URL: https://https://ieeexplore.ieee.org/document/8935958. 01.04.2020 р. Загол. з екрана.

52. The Microwave Oven Thermal Field Uniformity Increasing by Using Powermeter / O. B. Zaichenko, M. A. Miroshnyk, P. V. Galkin, L. V. Golovkina, Y. V. Pakhomov. *Joint Technical Program CAOL-2019:* 8-th International Conference on Advanced Optoelectronics and Lasers. Sozopol, Bulgaria, Sept. 6–8, 2019. 6 p. URL: https://ieeexplore.ieee.org/document/9019550. 01.04.2020 р. Загол. з екрана. Doi: 10.1109/ CAOL46282.2019.9019550.

53. Design of real-time system logic control on FPGA / M. A. Miroshnik, O. S. Shkil, E. N. Kulak, I. V. Filippenko, D. Y. Rakhil. *17th IEEE EAST-WEST DESIGN & TEST SYMPOSIUM* (IEEE EWDTS-2019). 2019 P. 1-4. DOI: 10.1109/EWDTS.2019.8884387/.

54. Miroshnik M., Zaichenko O., Galkin P. Signal Flow Graph for Optimizing of Mutual Sensors Reflection in the Multiprobe Microwave Multimeter. *IEEE UkrCON-2019*. July 2-6, 2019. Lviv, Ukraine. P. 200-203. doi: 10.1109/UKRCON.2019.8879925. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8879925&isnumber=8879768.

55. Mation of easy-tested digital finite state machines / M. A. Miroschnyk, Y. V. Pakhomov, A. S. Shkil, E. N. Kulak, D. Y. Kucherenko. *Radioelectronics, Informatics, Management.* Zaporozhye National Technical University, 2018. № 2. DOI: 10.15588/1607-3274-2018-2-13.

56. Design of Logical Control Units Based on Finite State Machines' Patterns / M. A. Miroshnyk, S. M. Poroshyn, A. S. Shkil, E. N. Kulak, I. V. Filippenko, D. Y. Kucherenko. *16th IEEE EAST-*

*WEST DESIGN & TEST SYMPOSIUM* (EWDTS-2018). 2018. № 9. P. 208-216.

57. Practical Methods for de Bruijn sequences Generation using Non-Linear Feedback Shift Registers / O. Demihev, M. Miroschnyk, D. Karaman, I..Filippenko, V. Krylova, T. Korytchinko. *14th IEEE International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering*. Lviv-Slavske, Ukraine. 2018. № 2. P. 35.

58. Miroshnik M. A., Zaichenko O. B., Galkin P. Analysis of mutual sensor reflection in the multiprobe microwave multimeter. *Radio Electronics & Info Communications* (UkrMiCo), International Conference/ IEEE UkrMiCo-2018. DOI: 10.1109/UkrMiCo43733.2018.

59. Synchronizing Sequences For Verification Of Finite State Machines / M. A. Miroshnik, A. S. Shkil, E. N. Kulak, I. V. Filippenko, D. Y. Kucherenko, A. S. Grebenyuk. *9th IEEE International Conference on Dependable Systems, Services and Technologies* (DESSERT'2018). May 24-27, 2018.

60. Miroschnyk M. A., Pakhomov U. V. Automation of hardware diagnostics of remote location for gas systems metering. *14th International Conference ICTERI 2018 ICT in Education, Research, and Industrial Applications*: Integration, Harmonization, and Knowledge Transfer. Kyiv – Ukraine. 2018. № 5. P. 14.

61. Design automation of testable finite state machines / М. А. Мирошник, А. С. Шкиль, Э. Н. Кулак, Д. Е. Кучеренко, Е. Э. Герман. *15th IEEE EAST-WEST DESIGN & TEST SYMPOSIUM* (EWDTS-2017). 2017. № 9. С. 203-208.

62. Miroshnik M. A., Krulov V. A. Communication channel statistical characteristics research methods. *Материалы конференции 2016 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET).* С. 562-565. IEEE. 2016 / 2 / 23.

63. Miroshnik M., Kotukh V., Kaptsova N. Information analysis of energy 7. performance and factors affecting the serviceability of products illustrated by the example in gas industry. *Материалы конференции 2016 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET).* С. 436-438. IEEE. 2016 / 2 / 23.

64. Miroshnik M. A., Tsekhmistro R. I., Demichev A. I. Solving of sat-problems of artificial intelligence with the help of local elimination algorithms. *Telecommunications and Radio Engineering.* 2016. T. 75, Vol. 75, Issue 7. P. 621–629. doi: 10.15587/1729-4061.2016.76103.

65. 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. / O. B. Zaichenko, I. I. Klyuchnik, M. A. Miroshnik, R. I. Tzekhmistro. *Telecommunications and Radio Engineering*. 2015. T. 74. № 1. P. 79-86.

66. Miroshnik 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. T. 27, № 10. P. 875-891.

67. Miroshnik M., Zagarij G., Derbunovich L. Design of a built-in diagnostic infrastructure for fault-tolerant telecommunication systems. *IEEE Modern Problems of Radio Engineering, Telecommunications and Computer Science - Proceedings of the 11th International Conference, TCSET'2012.*

68. Olsen S. The multicore SoC: Will 2010 be the turning point. *Design and Reuse*. 2010. URL: http://www.design–reuse.com/articles/ 23163/ multicore–soc.html. Date of access: 27.07.2011.

69. Ежов В. Coretx–A9 MPCore: продуктивность настольного компьютера, энергопотребление мобильного устройства. *Электронные компоненты*. 2009. № 10. С. 70–71.

70. Gierenz V., Panis C., Nurmi J. Parameterized MAC unit generation for a scalable embedded DSP core. *Microprocessors and Microsystems*. 2010. Vol. 34, Issue 5. P. 138–150.

71. Процессор арифметического декодера стандарта H.264 / А. В. Станкевич [та ін.]. *Информационные системы и технологии (Information Systems and Technologies (IST'2010): материали VI междунар. конф.* Минск, 24–25 нояб. 2010. Минск, 2010. С. 561–564.

72. Петровский Ал. А., Станкевич А. В., Петровский А. А. Быстрое проектирование систем мультимедиа от прототипа. Минск: Бестпринт, 2011. 412 с.

73. Rapid single–chip secure processor prototyping on the OpenSPARC FPGA platform / J. M. Szefer [et al.]. *Rapit System Prototyping (RSP'11)*: Proc. on *22nd* IEEE Int. Symp., Karlsruhe, Germany, 24–27 May, 2011. P. 38–44.

74. FPGA Based Symmetric Multi–core Processors for Optimized Performance of H.264 Encoder / M. E. Krishnan [et al.]. *Advances in Recent Technologies in Communication and Computing (ARTCom'10):* Proc. on IEEE Int. Conf., Kottayam, India, 16–17 Oct., 2010. P. 235–239.

75. Temple K. Intel Expands Customer Choice with First Configurable Intel Atom–based Processor. *Intel Newsroom*. 22 Nov. 2010. URL: http://newsroom.intel.com/community/intel_newsroom/blog/2010/11/22/intel–expands–customer–choice–with–first–configurable–intel–atom–based–processor. Date of access: 13.06.2011.

76. Iqbal M. A., Awan U. S. Reconfigurable Processor Architecture For High Speed Applications. *Advance Computing Conference (IACC'2009)*: Proc. on IEEE Int. Conf., Kottayam, India, 6–7 Mar., 2009. P. 624–629.

77. Akoglu A., Sreeramareddy A., Josiah J. G. FPGA based distributed self healing architecture for reusable systems. *Cluster Computing*. 2009. Vol. 12, Num. 3. P. 269–284.

78. Shashikumar R., Gouda L. C. S. Self–Healing Reconfigurable FPGA Based Fault Tolerant Security Model for Shared Internet Resources. *International Journal of Computer Science and Network Security*. 2009. Vol. 9, Num. 1. P. 281–285.

79. Self–Healing Approaches for FPGAs and Wiring Manifolds / S. Thompson [et al.]. *NASA Ames Research Center*. 2010. URL: http://ti.arc.nasa.gov/rn/pub–archive/1311h/1311%20(Thompson,%20S).pdf. – Date of access: 05.08.2011.

80. Tsang C.–C., Knok– Hay So H.Dynamic power reduction of FPGA–based configurable computers using precomputation. *ACM SIGARCH Computer Architecture News.* 2010. Vol. 38, Issue 4. P. 87–92.

81. Doboli A., Kane P., Van D.Ess Dynamic reconfiguration in a PSoC device. *Field–Programmable Technology (FPT'2009):* Proc. on Int. Conf., Sydney, Australia, 9–11 Dec., 2009. P. 361–363.

82. In–System Programming Design Guidelines for ispJTAG Devises: technical documentation. *Lattice Semiconductor Corporation*. 2002. URL: http://www.latticesemi.com/lit/docs/ispbs/ispguide.pdf. Date of access: 18.08.2011.

83. PicoBlaze 8–Bit Microcontroller for CPLD Devices: application note XAPP387. Xilinx Inc., 2003. URL: http://www.xilinx.com/support/documentation/application_ notes/xapp387.pdf. Date of access: 29.08.2011.

84. Weste H. E., Harris D. M. CMOS VLSI Design: A Circuits and Systems Perspective. NY: Addison Wesley, 2010. 838 p.

85. Zynq–7000 Extensible Processing Platform. Xilinx Inc., 2012. URL: http://www.xilinx.com/products/silicon–devices/epp/zynq–7000/ index.htm. Date of access: 17.04.2011.

86. MicroBlaze Soft Processor Core. Xilinx Inc., 2012. URL: http://www.xilinx.com/tools/ microblaze.htm. Date of access: 16.05.2012.

87. Programmable System–on–Chip. Cypress Semiconductor Corporation, 2012. URL: http://www.cypress.com. Date of access: 19.04.2011.

88. PSoC5: CY8C55 Family Datasheet. Cypress Semiconductor Corporation, 2012. URL: http://www.cypress.com/?docID=34815. Date of access: 19.04.2011.

89. Flynn M. J., Luk W.Computer System Design: System–On–Chip. New Jersey: John Wiley and Sons, 2011. 320 p.

90. Salminen E., Kulmala A., Hamalainen T. D. Survey of Network–on–chip Proposals. 2008. URL: http://www.ocpip.org/uploads/documents/OCP–IP_Survey_of_NoC_Proposals_White_ Paper_April_2008.pdf. Date of access: 08.05.2012.

91. A comparison of Network–on–Chip and Busses. Design and Reuse, 2005. URL: http://www.design–reuse.com/articles/10496/a–comparison–of–network–on–chip–and–busses.html. Date of access: 14.05.2012.

92. Choosing an Intellectual Property Core. MIPS Technologies, Inc., 2002. URL: http://www.mips.com/media/files/white–papers/Choosing_an_Intellectual_Property_Core.pdf. Date of access: 19.04.2011.

93. Intellectual Property and Cores Overview. Xilinx, Inc., 2008. URL: http://www.xilinx.com/itp/xilinx10/isehelp/ise_c_intellectual_property_cores.htm. Date of access: 19.04.2011.

94. The VSM Advantage. Labcenter Electronics, Ltd., 2012. URL: http://www.labcenter.com/ products/vsm_overview.cfm. Date of access: 20.04.2011.

95. Digilent Nexys2 Board Reference Manual. Digilent Inc., 2008. URL: http://digilentinc.com/Data/Products/NEXYS2/Nexys2_ rm.pdf. Date of access: 25.03.2011.

96. Spartan–3 Generation FPGA User Guide: Extended Spartan–3A, Spartan–3E, and Spartan–3 FPGA Families. Xilinx Inc., 2011. URL: http://www.xilinx.com/support/documentation/user_guides/ ug331.pdf. Date of access: 05.12.2011.

96. Spartan–3E Libraries Guide for Schematic. Design Xilinx Inc., 2011. URL: http://www.xilinx.com/support/documentation/sw_ manuals/xilinx13_1/spartan3e_scm.pdf. Date of access: 15.12.2011.

97. Spartan–3 Libraries Guide for HDL Designs. Xilinx Inc., 2009. URL: http://www.xilinx.com/support/documentation/sw_ manuals/xilinx13_2/spartan3_hdl.pdf. Date of access: 18.01.2012.

99. DesignWare Memory Compilers and Logic Libraries for TSMC 28HP. Synopsys Inc., 2011. URL: https://www.synopsys.com/ dw/doc.php/ds/es/28HP_memory_logic_DS.pdf. Date of access: 13.02.2012.

100. Xilinx CORE Generator System. Xilinx Inc., 2012. URL: http://www.xilinx.com/tools/coregen.htm. Date of acces: 17.02.2012.

101. ISE In–Depth Tutorial. Xilinx Inc., 2011. URL: http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_ 3/ise_tutorial_ug695.pdf. Date of access: 17.02.2012.

102. Active–HDL. FPGA Design Creation and Simulation. Aldec Inc., 2012. URL: http://www.aldec.com/en/products/ fpga_simulation/active–hdl. Date of access: 05.04.2012.

103. ISE WebPACK Design Software. Xilinx Inc., 2011. URL: http://www.xilinx.com/products/design–tools/ise–design–suite/ise–webpack.htm. Date of access: 14.06.2011.

104. Boundary–Scan Tutorial. Corelis Inc., 2012. URL: http://www.corelis.com/education/Boundary–Scan_Tutorial.htm. Date of access: 25.04.2012.

105. IEEE Std. 1149.1 – Standard Test Access Port and Boundary–Scan Architecture. URL: http://grouper.ieee.org/groups/ 1149/!/. Date of access: 25.04.2012.

106. Synplify Pro: Logic Synthesis for FPGA Implementation. Synopsys Inc., 2012. URL: http://www.synopsys.com/tools/implementation/fpgaimplementation/fpgasynthesis/pages/synplifypro.aspx. Date of access : 30.05.2012.

107. Drime, S. Volatile FPGA design security – a survey. *Computer Laboratory, University of Cambridge*. 2008. URL: http://www.cl.cam.ac.uk/sd410/papers/fpga_security.pdf. Date of access: 04.10.2011.

108. Ruhrmair U., Solter J., Sehn F.On the Foundations of Physical Unclonable Functions. *Cryptology ePrint Archive*. URL: http://eprint.iacr.org/2009/277.pdf. Date of access: 01.09.2011.

109. Gang Q., Chi–En Y.Temperature–aware cooperative ring oscillator PUF. *Hardware–Oriented Security and Trust (HOST'2009)*: Proc. of Int. Workshop, San Francisco, CA, USA, 27 Jul. 2009. P. 36–42.

110. Yin C.–E. D., Gang Q.LISA: Maximizing RO PUF's Secret Extraction. *Hardware–Oriented Security and Trust (HOST'2010)*: Proc. of Int. Workshop, Anaheim, CA, USA, 13–14 Jun. 2010. P. 100–105.

111. Holcomb D. E., Burleson W. P., Fu K.Power–Up SRAM State as an Identifuing Fingerprint and Source of True Random Numbers. *IEEE Transactions on Computers*. 2009. Vol. 58, Num. 9. P. 1198–1210.

112. Maes R., Tuyls P., Verbauwhede I.Intrinsic PUFs from Flip–flops on Reconfigurable Devices. *Information and System Security (WISSec'2008):* Proc. of 3rd Benelux Workshop, Eindhoven, The Netherlands, 13–14 Nov. 2008. URL: http://www.cosic.esat.kuleuven.be/publications/article–1173.pdf. Date of access: 30.09.2011.

113. Ярмолик В. Н., Вашинко Ю. Г. Физически неклонируемые функции. *Информатика*. 2011. № 2. С. 90–100.

114. Creating a unique digital fingerprint using existing combinational logic / H. J. Patel [et al.]. *Circuits and Systems (ISCAS'2009):* Proc. of IEEE Int. Symp., Taipei, PRC, 24–27 May. 2009. P. 2693–2696.

115. FPGA design security with time division multiplexed PUFs / G. Sezer [et al.]. *High Performance Computing and Simulation*

*(HPCS'2010):* Proc. of Int. Conf., Caen, France, 28 Jun. – 2 Jul. 2010. P. 608–614.

116. Trustworthy Hardware: Trojan Detection and Design–for–Trust Challenges / M. Tehranipoor [et al.]. *Computer*. 2011. Vol. 44, Num. 7. P. 66–74.

117. Physical Unclonable Function and True Random Number Generator : a Compact and Scalable Implementation / A. Maiti [et al.]. *ACM Great Lakes Symposium on VLSI (GLSVLSI'2009)*: Proc., Boston, USA, 10–12 May. 2009. P. 425–428.

118. Numonyx Embedded Flash Memory (J3 v D). Micron Technology, 2007. URL: http://www.micron.com/get–document/?documentId=6061. Date of access: 14.10.2011.

119. A Large Scale Characterization of RO–PUF / A. Maiti [et al.]. *Hardware–Oriented Security and Trust (H0ST'10):* Proc. on IEEE Int. Symp., Anaheim, CA, USA, 13–14 June. 2010. P. 66–71.

120. Improving the Quality of Ring Oscillator PUFs on FPGAs / D. Merli [et al.]. *Embedded Systems Security (WESS'10):* Proc. of the 5th Workshop, Scottsdale, AZ, USA, 24 Oct. 2010. P. 190–198.

121. Fine–Grained Characterization of Process Variation in FPGAs / H. Yu [et al.]. *Field Programmable Technology (FPT'10):* IEEE Int. Conf., Beijing, China, 8–10 Dec. 2010. P. 138–145.

122. An FPGA Chip Identification Generator Using Configurable Ring Oscillator / H. Yu [et al.]. *Field Programmable Technology (FPT'10):* IEEE Int. Conf., Beijing, China, 8–10 Dec. 2010. P. 312–315.

123. Drutarovsky M., Varchola M.Analysis of Randomness Sources in Transition Effect Ring Oscillator based TRNG. *Cryptographic Architectures Embedded in Reconfigurable Devices (CryptArchi'10)*: Proc. of Int. Workshop, Gif sur Yvette, France, 27-30 June. 2010. P. 102–107.

# Додаток 1

## Лістинги до розділу 2

Лістинг 2.1. VHDL-опис цифрового пристрою PORTS

```
1   Library IEEE;
2   Use IEEE.STD_LOGIC_1164.ALL;
3
4   Entity PORTS is
5       Port ( IP     : in     STD_LOGIC;
6              Z      : in     STD_LOGIC;
7              OP     : out    STD_LOGIC;
8              ZOP    : out    STD_LOGIC;
9              NOP    : out    STD_LOGIC;
10             BOP    : buffer STD_LOGIC );
11  End PORTS;
12
13  Architecture Behavioral of PORTS is
14  Begin
15
16     OP   <=  IP;
17     BOP  <=  IP ;
18     NOP  <=  BOP ;
19     ZOP  <=  IP  when  Z='0'  else  'Z';
20     End Behavioral ;
```

Лістинг 2.2. VHDL-опис цифрового пристрою IOPORT

```
1   Entity IOPORT is
2       Port ( MODE : in    std_logic;
3              Dio  : inout std_logic;
4              Di   : in    std_logic;
5              Do   : out   std_logic );
6   End IOPORT;
7
8   Architecture Beh of IOPORT is
9
10  Begin
11
12     -- Порт Dio в режимі вихідного порту
13     Dio <= Di when MODE='0' else 'Z';
14
15     -- Порт Dio в режимі вхідного порту
16     Do <= Dio when MODE= '1 ' else 'Z';
17
18     End Beh;
```

Лістинг 2.3. VHDL-опис компоненти CLOGIC_4x1

```
1   Library IEEE;
2   Use IEEE.STD_LOGIC_1164.ALL;
3   Use IEEE.STD_LOGIC_ARITH.ALL;
4   Use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6   Entity CLOGIC_4x1 is
7
8       Generic ( -- Кількість вхідних портів компоненти
9                 Ni     :     integer range 1 to 3                    := 1;
10                -- Значення функції у векторному шістнадцятковому форматі
11                INIT   :     std_logic_vector (15 downto 0 )         := x" 0001");
12
```

```
13        Port    ( — Вхідна Ni-розрядна шина
14                  IP      :       in      std_logic_vector ( Ni–1 downto 0 );
15                  — вихідний порт
16                  F       :       out     std_logic );
17    End CLOGIC_4x1 ;
18
19    Architecture Behavioral of CLOGIC_4x1 is
20    Begin
21
22        — З молодшої частини таблиці INIT,
23        — розмірність якої обмежується параметром Ni,
24        — вибирається двійкове значення, що зберігається за адресою IP.
25
26        — Вибране значення є значенням функції, що транслюється
27        — на вихідний порт F.
28      F <= INIT ( 2 ** Ni–1 downto 0 )( CONV_INTEGER ( IP ) );
29
30    End Behavioral ;
```

Приклади використання компоненти CLOGIC_4x1 і їхні еквівалентні логічні вирази наведено нижче.

```
—           F <= not A;
        U0: CLOGIC_4        generic map     ( 1, x"0001 " )
                            port map        (A, F );
—           F <= A and B;
        U1: CLOGIC_4        generic map     ( 2, x"0004" )
                            port map        ( A & B , F ) ;
—           F <= A xor B;
        U1: CLOGIC_4        generic map     ( 2, x"0006" )
                            port map        ( A & B , F ) ;
—           F <= A xor B xor C;
        U1: CLOGIC_4        generic map     ( 3, x"0096" )      — див. табл. 2.1
                            port map        ( A & B & C, F );
```

Наприклад, у бібліотеці UNISIM є пакет vcomponents, що містить опис технологічної компоненти lut4, яку можна використовувати безпосередньо в призначеному для користувача проекті:

```
        Library  UNISIM;
        use   UNISIM.vcomponents.all;
        —
—               LUT4: 4— input Look—Up Table with general output
—                       Spartan —3E
—               Xilinx HDL Language Template, version 12.4

            LUT4_inst : LUT4
            generic map (
            NIT => X"0000")
        port map (
                O     =>  O, — LUT general output
                I0    =>  I0 , — LUT input
I                   =>  11 , — LUT input
12                  =>  I2 , — LUT input
13                  =>  13 — LUT input
        );
            — End of LUT4 inst instantiation
```

Лістинг 2.4. VHDL-опис D-засувки

```
1   Entity DLATCH is
2       Port  ( D, E   :  in    STD_LOGIC;
3                Q,nQ  :  out   STD_LOGIC );
4   End DLATCH;
5
6   Architecture Beh of DLATCH is
7   Begin
8
9   Main: process ( D, E )
10   begin
11     if  E= '1'   then
12       Q    <=  D ;
13       nQ   <=  not D;
14     end if ;
15   end process;
16
17   End Beh ;
```

Лістинг 2.5. Поведінковий VHDL-опис D-тригера

```
1   Entity DFF is
2       Port ( — вхідний порт даних
3              D      :   in    STD_LOGIC;
4              — вхідний порт сигналу синхронізації
5              CLK    :   in    STD_LOGIC;
6              — вихідний порт даних
7              Q      :   out   STD_LOGIC );
8   End DFF;
9
10   Architecture Beh of DFF is
11  — об'єкт зберігання значення сигналу вхідного порту даних
12   signal s : std_logic ;
13   Begin
14
15   Main:   process ( D, CLK )
16   begin
17     — умова настання переднього фронту сигналу CLK
18     If  (  CLK'event  and  CLK =  '1'  )  then
19       — захоплення значення сигналу порту D
20        s  <=  D;
21     — при всіх інших умовах s здійснює зберігання
22     — захопленого значення сигналу порту D
23     end if;
24   end process;
25
26   — передача значення сигналу s на вихідний порт Q
27   Q <= s ;
28
29 End Beh;
```

Лістинг 2.6. Поведінковий VHDL-опис тригера *fdcpe*

```
1  Entity DFF2 is
2        Port (     — вхідний порт сигналу асинхронного скидання
3                   CLR   :   in      STD_LOGIC;
4                   — вхідний порт сигналу асинхронного установлення
5                   PRE   :   in      STD_LOGIC;
6                   — вхідний порт сигналу дозволу синхронізації
7                   CE    :   in      STD_LOGIC;
8                   — вхідний порт сигналу синхронізації
9                   C     :   in      STD_LOGIC;
10                  — вхідний порт даних
11                  D     :   in      STD_LOGIC;
12                  — вихідний порт даних
13                  Q     :   out     STD_LOGIC );
14 End DFF2;
15
16 Architecture Beh of DFF2 is
17 — об'єкт зберігання значення сигналу вхідного порту даних
18 signal  s : std_logic;
19 Begin
20
21  Main: process( CLR, PRE, CE, C, D )
22  begin
23     — умова асинхронного скидання
24      if CLR = '1' then
25         s <= '0 ';
26     — умова асинхронного установлення
27      elsif PRE = '1' then
28         s <= '1 ';
29     — умова асинхронного дозволу
30      elsif CE = '1' then
31         — умова настання переднього фронту синхросигналу
32          if rising_edge( C ) then
33             s <= D;
34         — при всіх інших умовах s здійснює зберігання
35         — захопленого значення сигналу порту D
36          end if ;
37      end if ;
38 end process;
39
40 — передача значення сигналу s на вихідний порт Q
41 Q <= s ;
42
43  End Beh;
```

Лістинг 2.7. Поведінковий VHDL-опис n-розрядного регістра зберігання

```
1  Entity REGn is
2       Generic
3                — розрядність збережених слів
4              (  n       :   integer := 4 );
5        Port   (— вхідна шина даних
6                  Din    :   in    std_logic_vector      n — 1  downto  0   );
7                — вхідний порт сигналу завантаження даних
8                EN     :   in    std_logic;
9                — вихідна шина даних
```

```
10              Dout  :   out  std_logic_vector      ( n — 1 downto 0)  );
11 End REGn ;
12
13 Architecture Beh of REGn is
14 — множина елементів пам'яті потужністю n
15 signal  reg : std_logic_vector ( n —1 downto 0 );
16
17 Begin
18
19    Main: process ( Din, EN )
20    begin
21         — умова захоплення даних з вхідної шини Din
22         if EN = '1' then
23            reg <= Din;
24         — при всіх інших умовах здійснюється
25         — зберігання n-розрядного слова
26         end if;
27    end process;
28
29    — передача значення слова, що зберігається
30    — на вихідну шину Dout
31    Dout <= reg;
32
33 End Beh;
```

Лістинг 2.8. Поведінковий VHDL-опис синхронного n-розрядного регістра зберігання

```
1   Entity REGn is
2   Generic
3            ( — константне слово для ініціалізації регістра,
4              — також визначає розмірність регістра,
5              — за замовчуванням константне слово дорівнює "1001",
6              — розмірність n=4
7              INITREG     :   std_logic_vector := "1001" );
8
9    Port     ( — вхідна шина даних
10             Din          :   in      std_logic_vector ( INITREG' range );
11             — вхідний порт сигналу завантаження даних
12             EN           :   in      std_logic;
13             — вхідний порт сигналу ініціалізації
14             INIT         :   in      std_logic;
15             — вхідний порт сигналу синхронізації,
16             — управління переднім фронтом
17             CLK          :   in      std_logic;
18             — вхідний порт сигналу управління вихідними
19             — тристабільними буферами
20             OE           :   in      std_logic;
21             — вихідна шина даних
22             Dout         :   out     std_logic_vector ( INITREG' range ) );
23 End REGn;
24
25 Architecture Beh of REGn is
26 — безпосередньо сам регістр (синхронні D-тригери)
27 signal reg        : std_logic_vector ( INITREG' range );
28 — константний вектор "ZZZ .. .Z" розмірністю INITREG ' range
29 constant ALLZ : std_logic_vector ( INITREG' range ) := ( others => 'Z' ):
30
31 Begin
32
```

```
33    Main: process ( Din, EN, INIT, CLK )
34    begin
35        —— умова ініціалізації
36        if INIT = '1 ' then
37          reg <= INITREG;
38        —— умова дозволу завантаження даних
39        elsif EN = '1 ' then
40          —— умова настання переднього фронту
41          —— сигналу синхронізації
42          if rising_edge ( CLK ) then
43              —— захоплення і завантаження даних
44              reg <= Din ;
45          end if ;
46        end if ;
47    end process;
48
49      —— вихідні тристабільні буфери-підсилювачі
50      Dout <= reg when OE='0' else ALLZ;
51
52 End Beh;
```

Лістинг 2.9. Змішаний VHDL-опис двопортового реєстрового файла

```
1    Entity REGFile is
2        Generic
3                (    — константне слово для ініціалізації регістрів,
4                     — також визначає розмірність регістрів,
5                     — за замовчуванням константне слово дорівнює "0000",
6                     — розмірність n=4
7                     INITREG     :     std_logic_vector := "0000";
8                     — розмірності адресних шин запису і читання
9                     — для адресації 2 ** a регістрів
10                    a           :     integer := 2 );
11           Port (   — вхідний порт сигналу ініціалізації регістрів
12                    INIT        :     in      std_logic;
13                    — вхідний порт шини даних для запису
14                    WDP         :     in      std_logic_vector ( INITREG' range );
15                    — вхідний порт шини адреси для запису
16                    WA          :     in      std_logic_vector ( a —1 downto 0 );
17                    — вхідний порт шини адреси для читання
18                    RA          :     in      std_logic_vector ( a —1 downto 0 );
19                    — вхідний порт сигналу дозволу запису
20                    WE          :     in      std_logic;
21                    — вихідний порт шини прочитаних даних
22                    RDP         :     out     std_logic_vector ( INITREG' range ) );
23    End REGFile;
24
25    Architecture Beh of REGFile is
26    — декларація компоненти синхронного регістра зберігання
27    component REGn is
28        Generic
29                ( INITREG     :   std_logic_vector := "1001" );
30
31          Port  ( Din         :   in      std_logic_vector ( INITREG' range );
32                  EN          :   in      std_logic;
33                  INIT        :   in      std_logic;
34                  CLK         :   in      std_logic;
35                  OE          :   in      std_logic;
36                  Dout        :   out     std_logic_vector ( INITREG' range ) );
37    end component;
38    — вихідна шина дешифратора адреси записи
```

```
39    signal  wen     :     td_logic_vector ( 2**a —1 downto 0 );
40    — вихідна шина дешифратора адреси читання
41    signal  ren     :     std_logic_vector ( 2**a —1 downto 0 );
42    — загальна шина прочитаних даних
43    signal  read    :     td_logic_vector ( INITREG' range );
44
45     Begin
46
47    — дешифратор адреси запису
48    WAD: process ( WA )
49    begin
50      for i in 0 to 2**a—1 loop
51         if i = CONV_INTEGER( WA ) then
52           wen( i ) <= '1 ';
53         else
54           wen( i ) <= '0';
55         end if ;
56      end loop;
57    end process;
58
59   — дешифратор адреси читання
60   RAD: process ( RA )
61   begin
62      for i in 0 to 2**a—1 loop
63         if i = CONV_INTEGER( RA ) then
64           ren ( i ) <= '1 ';
65         else
66           ren ( i ) <= '0 ';
67         end if ;
68      end loop;
69   end process;
70
71   — множина з 2 ** а регістрів REGn
72   REGi: for i in 2**a —1 downto 0 generate
73      REGi  :  REGn  generic map  ( INITREG )
74                          port map       ( WDP, wen ( i ) , I NIT, WE, ren ( i ) , readd );
75   end generate;
76
77   — трансляція загальної шини прочитаних даних на вихідний порт RDP
78   RDP <= readd;
79
80   End Beh;
```

Лістинг 2.10. Структурний опис n-розрядного перетворювача послідовного коду в паралельний

```
1     Library IEEE;
2     Use IEEE.STD_LOGIC_1164.ALL;
3     Library  UNISIM;
4     Use UNISIM.vcomponents.ALL;
5
6     Entity SREGn is
7        Generic
8              (   — кількість розрядів зсувного регістра
9                  n          :   integer   := 4 );
10       Port  (   — вхідний порт сигналу синхронізації
11                 CLK        :   in        std_logic ;
12                 — вхідний порт сигналу ініціалізації
13                 RST        :   in        std_logic ;
14                 — вхідний порт сигналу дозволу зсуву
15                 SE         :   in        std_logic ;
16                 — вхідний порт послідовного коду
```

```
17                  Sin       :   in       std_logic;
18                  — вихідний порт паралельного коду
19                  Pout      :   out      std_logic_vector ( 0 to n — 1)
20          );
21  End SREGn;
22
23  Architecture Beh of SREGn is
24  — множина з'єднувальних ліній
25  signal  sreg  :  std_logic_vector  ( 0 to n — 1 );
26
27  Begin
28
29     — тригер з індексом 0
30      FDFF: FDCE port map ( sreg  ( 0 ), CLK, SE, RST, Sin );
31
32     — множина всіх інших n - 1 тригерів
33     DFFs: for i in 1 to n—1 generate
34         DFFi: FDCE port map ( sreg ( i ), CLK, SE, RST, sreg ( i — 1 ) );
35      end generate;
36
37     — передача значень сигналів з виходів всіх тригерів
38     — на вихідний порт Pout
39      Pout <= sreg ;
40
41  End Beh ;
```

Лістинг 2.11. Поведінковий опис n-розрядного перетворювача паралельного коду в послідовний

```
1   Entity PISOn is
2       Generic
3               ( — кількість розрядів зсувного регістра
4                 n       :     integer:=4);
5       Port    ( — вхідний порт сигналу синхронізації
6                 CLK     :     in       std_logic;
7                 — вхідний порт сигналу ініціалізації
8                 RST     :     in       std_logic;
9                 — вхідний порт сигналу управління
10                — LS='0' — завантаження паралельного коду в регістр
11                — LS='1' — послідовний зсув
12                LS      :     in       std_logic;
13                — вхідний порт паралельного коду
14                Pin     :     in       td_logic_vector ( 0 to n —1 );
15                — вихідний порт послідовного коду
16                Sout    :     out      std_logic
17          );
18  End PISOn ;
19
20  Architecture Beh of PISOn is
21  — n— розрядний зсувний регістр
22  signal     sreg      :     std_logic_vector ( 0 to n —1 );
23  — внутрішня шина даних
24  signal     sdat      :     std_logic_vector ( 0 to n —1 );
25  Begin
26
27     — послідовна схема пристрою
28      Main: process ( CLK, RST, sdat )
29      begin
30        — умова асинхронної ініціалізації регістра
31         if RST = '1 ' then
32            sreg <=  (  others  => '0'  );
33        — умова настання переднього фронту
```

```
34          — сигналу синхронізації
35          elsif rising_edge ( CLK ) then
36            — синхронне управління регістром
37            sreg <= sdat;
38          end if;
39        end process;
40
41        — комбінаційна схема пристрою
42        Data:    process ( LS, Pin, sreg )
43        begin
44          — умова завантаження паралельних даних
45          if LS = '0' then
46            sdat <= Pin;
47          — умова зсуву регістра на один розряд вправо
48          else
49            sdat <= Pin ( 0 ) & sreg ( 0 to n—2 );
50          end if;
51      end process;
52
53      — передача на вихідний порт послідовного коду
54      — зі старшого розряду зсувного регістра
55      Sout <= sreg ( n — 1 );
56
57    End Beh ;
```

Лістинг 2.12. Поведінковий опис n-розрядного генератора one-hot послідовності

```
1     Entity OneHot_n is
2       Generic
3               ( — кількість розрядів генератора
4                 n       :  integer:=4);
5       Port    ( — вхідний порт сигналу синхронізації
6                 CLK     :  in      std_logic;
7                 — вхідний порт сигналу ініціалізації
8                 RST     :  in      std_logic;
9                 — вихідний порт one-hot коду
10                Pout    :  out     std_logic_vector ( 0 to n —1 )
11            );
12    End OneHot_n;
13
14    Architecture Beh of OneHot_n is
15    — n— розрядний зсувний регістр
16    signal  sreg  :    std_logic_vector ( 0 to n —1 );
17    — внутрішня шина даних
18    signal  sdat  :    std_logic_vector ( 0 to n —1 );
19    Begin
20
21      — послідовна схема пристрою
22      Main: process ( CLK, RST, sdat )
23      begin
24        — умова асинхронної ініціалізації генератора
25        if RST = '1' then
26          — початковий стан генератора (10...0)
27          sreg ( 0 ) <= '1 ';
28          sreg ( 1 to n — 1 ) <= ( others => '0' );
29        — умова настання переднього фронту
30        — сигналу синхронізації
31        elsif rising_edge( CLK ) then
32          — синхронне управління генератором
33          sreg <= sdat;
34        end if ;
```

```
35      end process;
36
37      — комбінаційна схема генератора
38      sdat <= sreg ( n —1 ) & sreg( 0 to n—2 );
39
40      — передача на вихідний порт
41      — символу one-hot послідовності
42      Pout <= sdat;
43
44  End Beh;
```

Лістинг 2.13. Поведінковий опис n-розрядного лічильника Джонсона

```
1   Entity JC_n is
2       Generic
3               (  — кількість розрядів генератора
4                  — n  =  2**i
5                  I        :     integer:= 2 );
6         Port  (  — вхідний порт сигналу синхронізації
7                  CLK    :     in      std_logic;
8                  — вхідний порт сигналу ініціалізації
9                  RST    :     in      std_logic;
10                 — вхідний порт сигналу управління
11                 — LS='0' — завантаження початкового стану
12                 — LS='1' — дозвіл генерування
13                 LS       :     in      std_logic;
14                 — вхідний порт початкового стану
15                 Pin      :     in      std_logic_vector ( 0 to 2 ** i —1 );
16                 — вихідний порт символів послідовності Джонсона
17                 Pout     :     out     std_logic_vector ( 0 to 2** i —1 )
18            );
19  End JC_n;
20
21  Architecture Beh of JC_n is
22  — n— розрядний зсувний регістр
23  signal     sreg     :      std_logic_vector ( 0  to  2 ** i —1 );
24  — внутрішня шина даних
25  signal     sdat     :      std_logic_vector ( 0  to  2 ** i —1 );
26  Begin
27
28     — послідовна схема пристрою
29     Main: process ( CLK, RST, sdat )
30     begin
31        — умова асинхронної ініціалізації генератора
32        if RST  =  '1'  then
33           — початковий стан генератора (00... 0)
34           sreg <= ( others => '0' );
35        — умова настання переднього фронту
36        — сигналу синхронізації
37        elsif   rising_edge( CLK ) then
38           — синхронне управління генератором
39           sreg <= sdat;
40        end if ;
41     end process;
42
43     — комбінаційна схема генератора
44     Data: process ( LS, Pin, sreg )
45     begin
46     — умова завантаження початкового стану
47        if LS = '0' then
48           sdat <= Pin;
49      — умова генерування символів послідовності Джонсона
```

```
50          else
51              sdat <= not (sreg( 2 ** i —1 )) & sreg( 0 to 2 ** i —2 );
52          end if;
53       end process;
54
55       — передача на вихідний порт
56       — символу послідовності Джонсона
57        Pout <= sdat;
58
59    End Beh;
```

Лістинг 2.14. VHDL-опис генератора М-послідовності довільної розрядності

```
1     Entity LFSR_n is
2          Generic   (  — масив коефіцієнтів характеристичного полінома,
3                       — визначає розрядність зсувного регістра,
4                       — значення за замовчуванням “11001”
5                       — відповідає поліному 4-го ступеня виду
6                       f (x) = 1+ x+ x^4
7                       Alpha     :         std_logic_vector := ”11001” );
8          Port     (     — вхідний порт сигналу синхронізації
9                       CLK       :         in        std_logic;
10                      — вхідний порт сигналу ініціалізації
11                      RST       :         in        std_logic;
12                      — вихідний порт символів ММ-послідовності
13                      Pout      :         out       std_logic_vector ( 0 to alpha ' high —1 ) );
14    End LFSR_n;
15    Architecture Beh of LFSR_n is
16   — n— розрядний зсувний регістр
17    signal       sreg     :         std_logic_vector ( 0 to alpha ' high —1 );
18   — внутрішня шина даних
19    signal       sdat     :         std_logic_vector ( 0 to alpha ' high —1 );
20
21    Begin
22
23       — послідовна схема пристрою
24        Main: process ( CLK, RST, sdat )
25        begin
26           — умова асинхронної ініціалізації генератора
27           if   RST = '1 ' then
28              — початковий стан генератора (00. . . 0)
29              sreg <= ( others => '0' );
30           — умова настання переднього фронту
31           — сигналу синхронізації
32           elsif   rising_edge  (  CLK  )  then
33              — синхронне управління генератором
34              sreg <= sdat;
35           end if ;
36        end process;
37
38        — комбінаційна схема генератора
39        Data:     process ( sreg )
40        — змінна для обчислення нового значення
41        — молодшого розряду генератора
42        variable newbit    :         std_logic ;
43        — змінна для коригування значення
44        — молодшого розряду з метою генерування
```

```
45      — нульового стану
46      variable  zerostate :  std_logic;
47      begin
48      — ініціалізація змінних
49      newbit          : =  ' 0 ' ;
50      zerostate       : =  ' 0 ' ;
51
52      — обчислення значень змінних
53      for i in 0 to alpha'high —2 loop
54          zerostate := zerostate or sreg( i );
55          if alpha( i +1 ) = '1' then
56              newbit := newbit xor sreg( i );
57          end if;
58      end loop ;
59      zerostate := not zerostate;
60      newbit :=  zerostate  xor  newbit  xor  sreg  ( alpha'high —1 );
61      — формування нового значення генератора
62      sdat  <=  newbit & sreg ( 0 to alpha ' high — 2 );
63
64      end process;
65
66      — передача на вихідний порт
67      — символу M - послідовності
68    Pout <= sreg ;
69
70    End Beh;
```

Лістинг 2.15. VHDL-опис генератора M-послідовності з ввімкненими суматорами за модулем два в міжрозрядні зв'язки

```
1     Entity LFSR2_n is
2         Generic (   — масив коефіцієнтів характеристичного полінома
3                     — значення за замовчуванням "11001"
4                     — відповідає поліному 4-го ступеня виду
5                     — f(x) = 1 + x + x^4
6                     alpha     :     std_logic_vector   := "11001" );
7           Port (    — вхідний порт сигналу синхронізації
8                     CLK       :     in       std_logic;
9                     — вхідний порт сигналу ініціалізації
10                    RST       :     in       std_logic;
11                    — вихідний порт символів M-послідовності
12                    Pout      :     out      std_logic_vector ( 0 to alpha'high —1 ) );
13    End LFSR2_n;
14
15    Architecture Beh of LFSR2_n is
16    — n— розрядний зсувний регістр
17    signal   sreg     :     std_logic_vector ( 0 to alpha'high —1 );
18    — внутрішня шина даних
19    signal   sdat     :     std_logic_vector ( 0 to alpha'high —1 );
20
21    Begin
22
23    — послідовна схема пристрою
24        Main: process ( CLK, RST, sdat )
25        begin
26          — умова асинхронної ініціалізації генератора
27          if RST = '1 ' then
28            — початковий стан генератора (00. . . 0)
29            sreg  <=  ( others => '0' );
```

```
30          — умова настання переднього фронту
31          — сигналу синхронізації
32           elsif rising_edge ( CLK ) then
33             — синхронне управління генератором
34              sreg <= sdat;
35           end if ;
36        end process;
37
38        — комбінаційна схема генератора
39        Data: process( sreg )
40        begin
41        — формування нових значень розрядів
42        — з урахуванням коефіцієнтів характеристичного полінома
43        for i in alpha'high —1 downto 1 loop
44          if alpha( i ) = '1' then
45             sdat(i) <= sreg( alpha'high —1 ) xor sreg( i —1 );
46          else
47             sdat(i) <= sreg( i —1 );
48          end if;
49      end loop ;
50      sdat ( 0 )  <=  sreg ( alpha ' high —1 );
51      end process;
52
53      — передача на вихідний порт
54      — символу М-послідовності
55   Pout  <=  sreg ;
56
57   End Beh ;
```

Лістинг 2.16. Поведінковий VHDL-опис кінцевого автомата

```
1    Entity FSM is
2          Port (
3                  — Вхідні порти сигналів синхронізації і ініціалізації
4                  CLK, RST    :   in        std_logic;
5                  — Вхідна шина даних
6                  IP          :   in        std_logic_vector ( 3 downto 0 );
7                  — Вихідна шина даних
8                  OP          :   out       std_logic_vector ( 1 downto 0 ) );
9    End FSM;
10
11   Architecture Beh of FSM is
12  — перераховується тип станів автомата
13   type  states  is  ( S0, S1, S2, S3, S4 );
14  — сигнали поточного і наступного стану
15   signal  current_state , next_state : states;
16  — сигнали для вихідних буферів-підсилювачів
17   signal  fop  : std_logic_vector ( 1 downto 0 );
18
19   Begin
20      — Опис синхронної пам'яті автомата
21       FSM_dff:   process ( CLK, RST, next_state )
22       begin
23          — умова ініціалізації автомата
24           if RST = '1 ' then
25             — стартовий стан автомата
26             current_state  <=  S0;
27          — умова зміни поточного стану
28          — на новий
29           elsif   rising_edge ( CLK )  then
30              current_state <= next_state;
31          end if;
```

```
32      end process;
33
34      -- Опис комбінаційної логіки,
35      -- що виробляє значення нового стану
36      FSM_gamma:   process( current_state , IP )
37      begin
38        -- аналіз поточного стану
39        case current_state is
40          when S0 => next_state <= S1 ;
41          when S1 => if IP = "1101" then
42                                next_state <= S2;
43                             else
44                                next_state <= S1 ;
45                             end if ;
46          when S2 => if IP = "0001" then
47                                next_state <= S4;
48                             elsif IP = "1111 " then
49                                next_state <= S3;
50                             else
51                                next_state<=S2;
52                             end if ;
53          when S3 => next_state <= S3;
54          when S4 => if IP = "1011" then
55                                next_state <= S0;
56                             else
57                                next_state <= S4;
58                       end if;
59          when others => next_state <= S0;
60        end case;
61      end process;
62
63      -- Опис комбінаційної логіки,
64      -- виробляє значення для вихідної
65      -- шини даних
66      FSM_phi: process ( current_state )
67      begin
68        case current_state is
69          when S0            => fop  <=  "00";
70          when S1 | S4       => fop  <=  "01";
71          when S2            => fop  <=  "10";
72          when S3            => fop  <=  "11";
73          when others        => fop <= "00";
74        end case;
75      end process;
76
77      -- Передача сформованих значень
78      -- на вихідну шину даних
79      OP <= fop;
80    End Beh;
```

Лістинг 2.17. Поведінковий VHDL-опис модуля ОЗП

```
1   Entity RAM is
2     Generic (
3       -- розмірність шини адреси
4       m    :   integer   :=   2;
5       -- розмірність шини даних
6        n    :   integer   :=   2
7     );
8     Port (
9       -- шина управління, що містить
```

```
10          — сигнал синхронізації
11          CLK   :     in          std_logic;
12          — і сигнал управління читанням / записом
13          WR    :     in          std_logic ;
14          — шина адреси
15          AB    :     in          std_logic_vector ( m—1 downto 0 ) ;
16          — двонаправлена шина даних
17          DB    :     inout       std_logic_vector ( n — 1 downto 0 )
18        );
19    End RAM;
20
21    Architecture Beh of RAM is
22    — призначений для користувача підтип, що описує
23    — тип одного збереженого слова
24    subtype     word      is std_logic_vector ( n — 1 downto 0 );
25    — призначений для користувача тип, що описує
26    — двовимірну структуру даних, що складається зі слів типу word
27    type          tram      is array ( 0 to 2**m— 1 ) of word;
28    — синтезується множина запам'ятовуючих елементів
29    signal        sRAM    :   tram;
30    — цілочисельний сигнал для адресації sRAM
31    signal        adrreg    :   integer range 0 to 2**m— 1;
32
33    Begin
34       — переведення двійкового подання значення на шині AB
35       — в цілочисельний десятковий формат
36        adrreg  <=  CONV_INTEGER  ( AB );
37
38       — процес, що описує синхронну операцію запису
39       WRP: process ( WR, CLK, adrreg, DB )
40        begin
41           if WR = '0' then
42              if rising_edge ( CLK ) then
43                 sRAM( adrreg ) <= DB;
44              end if ;
45           end if;
46        end process;
47
48       — процес, що описує асинхронну операцію читання
49       — спільно з набором тристабільних буферів
50       RDP: process ( WR, sRAM, adrreg )
51       begin
52           if WR = '1 ' then
53              DB <= sRAM( adrreg );
54           else
55              DB <= ( others => 'Z' );
56           end if ;
57       end process;
58    End Beh ;
```

**Додаток 2**

# Лістинги до розділу 3

Лістинг 3.1: Мікропрограма для стекової архітектури

```
1          PUSH a  ;  TOS <= RAM[0]=a
2          PUSH x  ;  TOS <= RAM[3]=x, TOS+1=a
3          MUL     ;  TOS <= a *x
4          PUSH b  ;  TOS <= RAM[1]=b, TOS+ 1=a*x
5          ADD     ;  TOS <= a *x+b
6          PUSH x  ;  TOS <= RAM[3]=x
7          MUL     ;  TOS <= x * (a*x+b)
8          PUSH c  ;  TOS <= RAM[2]=c
9          ADD     ;  TOS <= x * (a *x+b)+c
10         POP  f  ;  f=RAM[4] <= TOS
```

```
1          PUSH a  ;  TOS <= RAM[0]=a
2          PUSH x  ;  TOS <= RAM[3]=x, TOS+1=a
3          MUL     ;  TOS <= a*x
4          PUSH b  ;  TOS <= RAM[1]=b, TOS+1=a*x
5          ADD     ;  TOS <= a*x+b
6          PUSH x  ;  TOS <= RAM[3]=x
7          MUL     ;  TOS <= x*(a*x+b)
8          PUSH c  ;  TOS <= RAM[2]=c
9          ADD     ;  TOS <= x*(a*x+b)+c
0          POP  f  ;  f=RAM[4] <= TOS
```

Лістинг 3.2. Приклад кодування мікропрограми

```
1   ;  Інструкція      :  Адреса  :  Бінарне значення
2   ; ──────────────────────────────────────────
3          PUSH  a     ;  0000    :  000–000
4          PUSH  x     ;  0001    :  000–011
5          MUL         ;  0010    :  011–000
6          PUSH  b     ;  0011    :  000–001
7          ADD         ;  0100    :  010–000
8          PUSH  x     ;  0101    :  000–011
9          MUL         ;  0110    :  011–000
10         PUSH  c     ;  0111    :  000–010
11         ADD         ;  1000    :  010–000
12         POP   f     ;  1001    :  001–100
13         HALT        ;  1010    :  100–000
```

Лістинг 3.3. Приклад кодування мікропрограми

| 1 ; | Мнемоніка | | : | Дія | \| | Адреса : Бінарне подання |
|---|---|---|---|---|---|---|
| 2 ; | ——— | | | ——— | \| | ——— |
| 3 | **LOAD** | a | ; | ACCUM <= RAM[0] | \| | 000 : 000–000 |
| 4 | **MUL** | x | ; | ACCUM <= ACCUM * RAM[3] | \| | 001 : 011–011 |
| 5 | **ADD** | b | ; | ACCUM <= ACCUM + RAM[ 1] | \| | 010 : 010–001 |
| 6 | **MUL** | x | ; | ACCUM <= ACCUM * RAM[3] | \| | 011 : 011–011 |
| 7 | **ADD** | c | ; | ACCUM <= ACCUM + RAM[2] | \| | 100 : 010–010 |
| 8 | ***STORE*** | f | ; | RAM[4] <= ACCUM | \| | 101 : 001–100 |
| 9 | **HALT** | | ; | STOP | \| | 110 : 100–000 |

Лістинг 3.4. Приклад кодування мікропрограми для GPR-архітектури

```
1   ;  Мнемоніка          :  Дія              |  Адреса   :  Бінарне подання
2   ;————————————————————————————————————|————————————————————————————————
3       MUL   f , a, x    ;  f <= a * x       |  000      :  01–100–000–011
4       ADD   f , f , b   ;  f <= f + b       |  001      :  00–100–100–001
5       MUL   f , f , x   ;  f <= f * x       |  010      :  01–100–100–011
6       ADD   f , f , c   ;  f <= f + c       |  011      :  00–100–100–010
7       HALT              ;  STOP             |  100      :  10–000–000–000
```

Лістинг 3.5. Поведінковий VHDL-опис компонента MROM

```
1   ————————————————————————————————————————————
2   — Пам'ять мікропрограми ———————————————————————
3   ————————————————————————————————————————————
4   Entity MROM is
5         Port ( — Вхідний порт сигналу дозволу читання
6                  RE    :  in    STD_LOGIC;
7                  — Вхідна шина адреси інструкції
8                  ADR   :  in    STD_LOGIC_VECTOR( 2 downto 0 );
9                  — Вихідна шина інструкції
10                 DOUT  :  out   STD_LOGIC_VECTOR( 5 downto 0 ) );
11  End MROM;
12   ———————————————————————————————————————————
13  Architecture Beh of MROM is
14
15  — призначений для користувача підтип для визначення однієї інструкції
16  subtype  inst  is  std_logic_vector ( 5 downto 0 );
17
18  — призначений для користувача тип двовимірного масиву ПЗП
19  type  tROM  is  array  (0 to 7 )  of  inst;
20
21  — ініціалізація ПЗУ двійковим поданням мікропрограми
22  constant ROM : tROM : = (
23   — Двійковий код      —     Адреса  :  Мнемоніка
24   ———————————————————————————————————————————
25     "000"&"000",       —     000     :  LOAD a
26     "011 "&"011",      —     001     :  MUL x
27     "010 " &" 0 01 ",  —     010     :  ADD b
28     "011 " &" 011 ",   —     011     :  MUL x
29     " 010 " &" 010 ",  —     100     :  ADD c
30     "001"&"100",       —     101     :  STORE f
31     "100 "&"000",      —     110     :  HALT
32     "111"&"111"        —     111     :  ———
33   );
34
35  — сигнал, що описує значення обраної інструкції
36  signal data : inst;
37
38  Begin
39
40  -- вибірка адресованої інструкції з ПЗП
41  data <= ROM( CONV_INTEGER( ADR ) );
42
43  — опис вихідних тристабільних буферів
44  ZBUFS: process( RE, data )
45  begin
46     if ( RE = '1 ' ) then
47        DOUT <= data ;
48     else
```

```
49          DOUT <= ( others => 'Z' );
50        end if ;
51    end process;
52
53    End Beh;
54      ------------------------------------------------------
```

Лістинг 3.6. Поведінковий VHDL-опис компонента MRAM

```
1   ------------------------------------------------------
2   -- Пам'ять даних ------------------------------------
3   ------------------------------------------------------
4   Entity MRAM is
5         Port ( -- Вхідний порт сигналу дозволу читання/запису
6                 RW      :   in    STD_LOGIC;
7                 -- Вхідна шина адреси
8                 ADR     :   in    STD_LOGIC_VECTOR( 2 downto 0 );
9                 -- Вхідна шина даних
10                DIN     :   in    STD_LOGIC_VECTOR( 7 downto 0 );
11                -- Вихідна шина даних
12                DOUT    :   out   STD_LOGIC_VECTOR( 7 downto 0 ) );
13  End MRAM;
14  ------------------------------------------------------
15  Architecture Beh of MRAM is
16
17  -- призначений для користувача підтип для визначення 8-бітного слова
18  subtype byte is std_logic_vector ( 7 downto 0 );
19  -- призначений для користувача тип двовимірного масиву ОЗП
20  type tRAM is array (0 to 7 ) of byte;
21  -- початковий стан пам'яті даних
22  signal RAM : tRAM: = (
23  -- Двійковий код    --    Адреса    :   Змінна
24  ------------------------------------------------------------------------
25      "00000001",     --    000       :   a=1
26      "00000010",     --    001       :   b=2
27      "00000011",     --    010       :   c=3
28      "00000100",     --    011       :   x=4
29      "00000000",     --    100       :   f
30      "00000000",     --    101       :   ----
31      "00000000",     --    110       :   ----
32      "00000000"      --    111       :   ----
33    );
34  -- сигнали вхідних і вихідних даних
35  signal  data_in     :   byte;
36  signal  data_out    :   byte;
37
38  Begin
39
40  -- Процес запису даних до масиву запам'ятовуючих елементів
41  data_in <= DIN;
42  WRITE: process ( RW, ADR, data_in )
43  begin
44      if ( RW = '0 ' ) then
45          RAM( CONV_INTEGER( ADR ) ) <= data_in;
46      end if ;
47  end process;
48
49  -- Читання даних з масиву запам'ятовуючих елементів
50  data_out <= RAM( CONV_INTEGER( ADR ) );
51  -- Тристабільні буфери вихідної шини даних
```

```
ZBUFS: process( RW, data_out )
 begin
    if ( RW = '1 ' ) then
       DOUT <= data_out;
    else
       DOUT <= ( others => 'Z' );
    end if;
 end process;
 End Beh;
```

Лістинг 3.7. Поведінковий HDL-опис компоненти DPTH

```
 ---------------------------------------------------------------------
 Library  IEEE;
 Use      IEEE.STD_LOGIC_1164 . ALL;
 Use      IEEE. STD_LOGIC_ARITH . ALL;        — необхідні для опису
 Use      IEEE.STD_LOGIC_UNSIGNED. ALL;  — операцій додавання і множення
 ---------------------------------------------------------------------
— Тракт обробки даних ------------------------------------------------
---------------------------------------------------------------------
Entity DPTH is
       Port (—Вхідний порт сигналу дозволу
             EN    :   in      STD_LOGIC;
             — Вхідна шина коду операції
             OT    :   in      STD_LOGIC_VECTOR( 2 downto 0 );
             — Вхідна шина першого операнда
             OP1   :   in      STD_LOGIC_VECTOR( 7 downto 0 );
             — Вихідна шина результату
             RES   :   out     STD_LOGIC_VECTOR( 7 downto 0 ) );
 End DPTH ;
 ---------------------------------------------------------------------
 Architecture Beh of DPTH is
— регістр-акумулятор
 signal ACCUM       :   std_logic_vector ( 7 downto 0 );
-- результат операції додавання
 signal res_add     :   std_logic_vector ( 7 downto 0 );
-- результат операції множення
signal res_mul      :   std_logic_vector ( 7 downto 0 );
— двійкові коди виконуваних операцій
constant LOAD       :   std_logic_vector ( 2 downto 0 ) := "000";
constant ADD        :   std_logic_vector ( 2 downto 0 ) := "010";
constant MUL        :   std_logic_vector ( 2 downto 0 ) := "011";

Begin

— двійковий 8-розрядний суматор
res_add <=  CONV_STD_LOGIC_VECTOR(
            CONV_INTEGER( ACCUM ) + CONV_INTEGER( OP1 ), 8 );
— двійковий 8-розрядний помножувач
res_mul <=  CONV_STD_LOGIC_VECTOR(
            CONV_INTEGER( ACCUM ) * CONV_INTEGER( OP1 ), 8 );

— синхронний регістр-акумулятор
REGA: process ( EN, OT, OP1, res_add , res_mul )
begin
    — якщо передній фронт сигналу дозволу
     if rising_edge( EN ) then
      — декодування типу операції
      case OT is
        — завантаження в акумулятор значення операнда
        when LOAD => ACCUM <= OP1;
```

```
50            — акумулювання результату операції додавання
51            when ADD => ACCUM <= res_add;
52            — акумулювання результату операції множення
53            when MUL => ACCUM <= res_mul;
54            when others => null;
55          end case;
56        end if;
57    end process;
58
59      — передача значення регістра-акумулятора на вихідну шину результату
60      RES <= ACCUM;
61
62      End Beh;
```

Лістинг 3.8. Поведінковий VHDL-опис компоненти CTRL1

```
1     ——————————————————————————————————————————————————————————
2     — Пристрій управління (кінцевий автомат) ————————————————————
3     ——————————————————————————————————————————————————————————
4      Entity CTRL1 is
5           Port (
6           — Вхідні порти сигналів управління
7           CLK, RST, Start      :  in     std_logic;
8           — Вихідний порт сигналу закінчення обчислення
9           Stop                 :  out    std_logic ;
10
11          — Порти для підключення до ЗП мікропрограми
12          ROM_re               :  out    std_logic;
13          ROM_adr              :  out    std_logic_vector ( 2 downto 0 );
14          ROM_dout             :  in     std_logic_vector ( 5 downto 0 );
15
16          — Порти для підключення до ЗП оброблюваних даних
17          RAM_rw               :  out    std_logic ;
18          RAM_adr              :  out    std_logic_vector ( 2 downto 0 );
19          RAM_din              :  out    std_logic_vector ( 7 downto 0 );
20          RAM_dout             :  in     std_logic_vector ( 7 downto 0 );
21
22          — Порти для підключення до тракту обробки даних
23          DP_op1               :  out    std_logic_vector ( 7 downto 0 );
24          DP_ot                :  out    std_logic_vector ( 2 downto 0 );
25          DP_en                :  out    std_logic;
26          DP_res               :  in     std_logic_vector ( 7 downto 0 ) );
27       End CTRL1 ;
28       ——————————————————————————————————————————————————————
29    Architecture Beh of CTRL1 is
30
31    — перечислювальний тип станів автомата
32    type states is ( I , F, D, R, L, S, A, M, H );
33    — I - ініціалізація (Idle)
34    — F - вибір інструкції (Fetch)
35    — D - декодування (Decode)
36    — R - вибір операнда з пам'яті (Read)
37    — L - виконання інструкції Load
38    — S - виконання інструкції Store
39    — A - виконання інструкції Add
40    — M - виконання інструкції Mul
41    — H - виконання інструкції Halt
42    — сигнали поточного і наступного стану автомата
43     signal    nxt_state ,cur_state: states;
44    — регістр обраної інструкції
45     signal    RI     :     std_logic_vector(5 downto 0);
```

```
46  — регістр лічильника інструкцій
47  signal    IC    :    std_logic_vector(2 downto 0);
48  — регістр типу операції
49   signal    RO    :    std_logic_vector(2 downto 0);
50  — регістр адреси пам'яті даних
51   signal    RA    :    std_logic_vector(2 downto 0);
52  — регістр даних
53   signal    RD    :    std_logic_vector (7 downto 0);
54
55  — Двійкові коди виконуваних операцій
56   constant    LOAD    :    std_logic_vector (2 downto 0) := "000";
57   constant    STORE   :    std_logic_vector (2 downto 0) := "001";
58   constant    ADD     :    std_logic_vector (2 downto 0) := "010";
59   constant    MUL     :    std_logic_vector (2 downto 0) := "011";
60   constant    HALT    :    std_logic_vector (2 downto 0) := "100";
61
62   Begin
63  ------------------------------------------------------------------------
64  — Синхронна пам'ять автомата
65  ------------------------------------------------------------------------
66   FSM:   process( CLK, RST, nxt_state )
67   begin
68       if ( RST = ' 1 ' ) then
69           cur_state  <=  I ;
70       elsif rising_edge ( CLK ) then
71           cur_state  <=  nxt_state;
72       end if;
73   end process;
74
75  ------------------------------------------------------------------------
76  — Комбінаційна частина автомата, що відповідає за вироблення значення
77  — наступного стану
78  ------------------------------------------------------------------------
79   COMB: process ( cur_state , Start,RO )
80   begin
81       case cur_state is
82           when I => if ( Start = '1 ' ) then
83                                nxt_state <= F;
84                             else
85                                nxt_state <= I;
86                             end if ;
87           when F => nxt_state <= D;
88           when D => if ( RO = HALT ) then
89                                nxt_state <= H;
90                             elsif ( RO = STORE ) then
91                                nxt_state <= S;
92                             else
93                                nxt_state <= R;
94                             end if;
95           when R => i f ( RO = LOAD ) then
96                                nxt_state <= L;
97                             elsif ( RO = ADD ) then
98                                nxt_state <= A;
99                             elsif ( RO = MUL ) then
100                               nxt_state <= M;
101                            else
102                               nxt_state <= I;
103                            end if ;
104          when L | S | A | M          =>   nxt_state   <= F;
105          when H                      =>   nxt_state   <= H;
106          when others                 =>   xt_state    <= I;
107       end case;
```

```
108  end process;
109
110  ------------------------------------------------------------------------
111  -- Схема вироблення значення сигналу Stop
112  ------------------------------------------------------------------------
113  PSTOP: process( cur_state )
114  begin
115      if ( cur_state = H ) then
116          Stop <= '1 ';
117      else
118          Stop <= '0 ';
119      end if ;
120  end process;
121  ------------------------------------------------------------------------
122
123  ------------------------------------------------------------------------
124  -- Контролер пам'яті ROM
125  ------------------------------------------------------------------------
126  -- Лічильник інструкцій
127  PMC: process( CLK, RST, cur_state )
128  begin
129      if ( RST = '1' ) then
130          IC <= "000";
131      elsif falling_edge ( CLK ) then
132          if ( cur_state = D ) then
133              IC <= IC + 1;
134          end if ;
135      end if ;
136  end process;
137
138  -- Передача значення лічильника IC на адресну шину пам'яті ROM
139  ROM_adr <= IC;
140
141  -- Формування значення сигналу читання з пам'яті ROM
142  PROMREAD: process( nxt_state , cur_state )
143  begin
144      if ( nxt_state = F or cur_state = F ) then
145          ROM_re <= '1 ';
146      else
147          ROM_re <= '0 ';
148      end if ;
149  end process;
150
151  -- Читання цього значення інструкції та
152  -- запис його в регістр RI
153  PROMDAT: process ( RST, cur_state , ROM_dout )
154  begin
155      if ( RST = '1' ) then
156          RI <= "000000";
157      elsif ( cur_state = F ) then
158          RI <= ROM_dout;
159      end if ;
160  end process;
161  ------------------------------------------------------------------------
162
163  ------------------------------------------------------------------------
164  -- Схема управління регістрами RO і RA
165  ------------------------------------------------------------------------
166  PRORA: process ( RST, nxt_state , RI )
167  begin
168      if ( RST= '1' ) then
169          RO <= "000";
```

```
170         RA <= "000";
171       elsif ( nxt_state = D ) then
172         RO <= RI ( 5 downto 3 );
173         RA <= RI ( 2 downto 0 );
174       end if ;
175   end process;
176   ------------------------------------------------------------------------

177
178   ------------------------------------------------------------------------

179   -- Контролер пам'яті RAM
180   ------------------------------------------------------------------------
181   -- Передача значення регістра RA на адресну шину пам'яті RAM
182   RAM_adr <= RA;
183
184   -- Формування керуючого сигналу читанням/записом для пам'яті RAM
185   PRAMREAD: process( cur_state )
186   begin
187       if ( cur_state = S ) then
188         RAM_rw <= '0';
189       else
190         RAM_rw <= '1';
191       end if;
192   end process;
193
194   -- Запис цього значення з пам'яті RAM в регістр RD
195   PRAMDAT: process( cur_state )
196   begin
197         if ( cur_state = R ) then
198           RD <= RAM_dout;
199         end if ;
200     end process;
201
202     --  Передача результуючого значення тракту обробки даних на
203     -- вхідну шину даних пам'яті RAM
204     RAM_din <= DP_res;
205
206     ------------------------------------------------------------------------
207     -- Схема управління трактом обробки даних
208     ------------------------------------------------------------------------
209     -- Передача значення регістра RD на вхід першого операнда
210     DP_op1 <= RD;
211     -- Передача значення регістра RO на вхідну шину типу операції
212     DP_ot <= RO;
213
214     -- Формування значення сигналу, що дозволяє
215     -- виконання відповідної операції
216     PADDMULEN: process( cur_state )
217     begin
218         if ( cur_state=A or cur_State=M or cur_State=L ) then
219           DP_en <= '1 ';
220         else
221           DP_en <= '0 ';
222         end if;
223     end process;
224     ------------------------------------------------------------------------
225     End Beh;
226     ------------------------------------------------------------------------
```

Лістинг 3.9. Поведінковий VHDL-опис компонента CTRL2

```
1  ------------------------------------------------------------------------------
2  -- Пристрій управління (конвеєрна обробка)-----------------------------------
3  ------------------------------------------------------------------------------
4   Entity CTRL2 is
5       Port (
6       -- Вхідні порти сигналів управління
7       CLK, RST, Start  : in      std_logic;
8       -- Вихідний порт сигналу закінчення обчислення
9       Stop             : out     std_logic ;
10
11      -- Порти для підключення до ЗП мікропрограми
12      ROM_re           : out     std_logic;
13      ROM_adr          : out     std_logic_vector ( 2 downto 0 );
14      ROM_dout         : in      std_logic_vector ( 5 downto 0 );
15
16      -- Порти для підключення до ЗП оброблюваних даних
17      RAM_rw           : out     std_logic ;
18      RAM_adr          : out     std_logic_vector ( 2 downto 0 );
19      RAM_din          : out     std_logic_vector ( 7 downto 0 );
20      RAM_dout         : in      std_logic_vector ( 7 downto 0 );
21
22      -- Порти для підключення до тракту обробки даних
23      DP_op1           : out     std_logic_vector ( 7 downto 0 );
24      DP_ot            : out     std_logic_vector ( 2 downto 0 );
25      DP_en            : out     std_logic;
26      DP_res           : in      std_logic_vector ( 7 downto 0 ) );
27  End CTRL2;
28   ------------------------------------------------------------------------------------------------------
29   Architecture Beh of CTRL2 is
30  -- Регістри лічильника інструкцій
31   signal IC_w, IC_r                : std_logic_vector ( 2 downto 0 );
32  -- Регістри інструкції
33   signal RI_w,RI_r                 : std_logic_vector ( 5 downto 0 );
34  -- Регістри типу операції
35   signal RO_w1, RO_w2              : std_logic_vector ( 2 downto 0 );
36   signal RO_r1 , RO_r2             : std_logic_vector ( 2 downto 0 );
37  -- Регістри адреси пам'яті даних
38   signal RA_w1, RA_w2              : std_logic_vector ( 2 downto 0 );
39   signal RA_r1 , RA_r2             : std_logic_vector ( 2 downto 0 );
40  -- Регістри даних
41   signal RD_w, RD_r                : std_logic_vector ( 7 downto 0 );
42  -- Регістр лічильника Джонсона
43   signal JC                        : std_logic_vector ( 3 downto 0 );
44  -- Біт дозволу обчислення
45   signal GO                        : std_logic ;
46  -- Прапорець закінчення обчислення
47   signal STP                       : std_logic;
48  -- Сигнали синхронізації конвеєра
49   signal F_en , D_en , R_en , W_en : std_logic;
50
51  -- Двійкові коди виконуваних операцій
52   constant LOAD    :   std_logic_vector ( 2 downto 0 ) := "000";
53   constant STORE   :   std_logic_vector ( 2 downto 0 ) := "001";
54   constant ADD     :   std_logic_vector ( 2 downto 0 ) := "010";
55   constant MUL     :   std_logic_vector ( 2 downto 0 ) := "011";
56   constant HALT    :   std_logic_vector ( 2 downto 0 ) := "100";
57
58   Begin
59  ------------------------------------------------------------------------------
```

```
60  — Генератор сигналів синхронізації стадій конвеєра
61  ------------------------------------------------------------------------
62  — Генератор послідовності Джонсона
63   PJC: process( CLK, RST, JC, GO, STP )
64   begin
65      if ( RST = '1 ' ) then
66         JC <= "0000";
67      elsif rising_edge ( CLK ) then
68         — Якщо процес обчислення дозволений
69          if ( GO = '1 ' ) then
70            — Якщо процес обчислення починається або закінчується
71             if ( JC /= " 1 1 11 " or STP = '1 ' ) then
72               — Виробляється черговий символ послідовності
73                JC <= JC( 2 downto 0 ) & not ( JC ( 3 ) );
74               — 0001—0011—0111 : початок процесу
75               — 1111 : процес виконується
76               — 1110 —1100 —1000 : закінчення процесу
77             end if ;
78          end if;
79      end if;
80   end process;
81  — Вироблення значень сигналів синхронізації для кожної стадії
82   F_en <= CLK and JC( 0 );
83   D_en <= CLK and JC( 1 );
84   R_en <= CLK and JC( 2 );
85  W_en <= CLK and JC( 3 );
86  ------------------------------------------------------------------------
87  — Генератор значень сигналу GO
88  ------------------------------------------------------------------------
89   PGO: process( RST, Start, STP, JC )
90   begin
91      if ( RST = '1 ' ) then
92         GO <= ' 0 ' ;
93      — Початок обчислювального процесу
94      elsif ( Start = '1 ' ) then
95         GO <= ' 1 ';
96      — Закінчення обчислювального процесу
97      elsif ( STP = '1' and JC = "0000" ) then
98            GO <= '0 ';
99      end if;
100  end process;
101 ------------------------------------------------------------------------
102 — Генератор значень сигналу STP
103 ------------------------------------------------------------------------
104  PSTP: process ( RST, RI_w )
105  begin
106     if ( RST = '1' ) then
107        STP <= '0';
108     — Якщо була обрана інструкція зупинки HALT
109     elsif ( RI_w = HALT & "000" ) then
110        STP <= '1 ';
111     end if ;
112  end process;
113 ------------------------------------------------------------------------
114 — Генератор вихідного сигналу Stop
115 ------------------------------------------------------------------------
116  PStop: process( RST, RO_r2 )
117  begin
118     if ( RST = '1' ) then
119        Stop <= '0 ';
120     — Якщо на останній стадії WRITE виконується інструкція HALT
121     elsif ( RO_r2 = HALT ) then
```

```
122         Stop <= '1 ';
123     end if ;
124  end process;
125  ---------------------------------------------------------------------------
126  -- Перша стадія конвеєра (FETCH)
127  ---------------------------------------------------------------------------
128  -- Сигнал управління читанням з ROM
129  ROM_re <= not STP;
130  -- Передача значення лічильника IC на адресну шину ROM
131  ROM_adr <= IC _r;
132  -- Формування значення регістра IC_r
133  -- в першій половині стадії FETCH
134  PIC_r: process( RST, F_en, IC_w )
135  begin
136     if ( RST = '1' ) then
137        I C_r <= "000";
138     elsif rising_edge( F_en ) then
139        I C_r <= IC_w;
140     end if ;
141  end process ;
142
143  -- Формування значення регістра IC_w
144  -- і запис обраної інструкції в регістр RI_w
145  -- у другій половині стадії FETCH
146  PIC_w: process ( RST, F_en, STP, IC_w )
147  begin
148     if ( RST = '1 ' ) then
149        IC_w <= "000";
150        RI_w <= "000000";
151     elsif falling_edge ( F_en ) then
152        if ( STP = '0' ) then
153           IC_w <= IC_w + 1;
154           RI_w <= ROM_dout;
155        end if ;
156     end if ;
157  end process;
158  ---------------------------------------------------------------------------
159  -- Друга стадія конвеєра (DECODE)
160  ---------------------------------------------------------------------------
161  -- Формування значення регістра RI_r
162  -- у першій половині стадії DECODE
163  PRIr: process ( RST, D_en, RI_w )
164  begin
165     if ( RST = '1' ) then
166        RI_r <= "000000";
167     elsif rising_edge( D_en ) then
168        RI_r <= RI_w;
169     end if ;
170  end process;
171
172  -- Формування значень регістрів RO_w1 та RA_w1
173  -- у другій половині стадії DECODE
174  PRORAw1: process ( RST, D_en, RI_r )
175  begin
176     if ( RST = '1' ) then
177        RO_w1 <= "000";
178        RA_w1 <= "000";
179     elsif f a11i ng_edge ( D_en ) then
180        RO_w1 <= RI _ r ( 5 downto 3 );
181        RA_w1 <= RI _ r ( 2 downto 0 );
182     end if ;
183  end process;
```

```
184 ------------------------------------------------------------------------
185 -- Третя стадія конвеєра (READ)
186 ------------------------------------------------------------------------
187 -- Формування значень регістрів RO_r1 та RA_r1
188 -- у першій половині стадії READ
189 PRORAr1: process ( RST, R_en, RO_w1, RA_w1 )
190 begin
191     if ( RST = '1 ' ) then
192         RO_r1 <= "000";
193         RA_r1 <= "000";
194     elsif rising_edge( R_en ) then
195         RO_r1 <= RO_w1;
196         RA_r1 <= RA_w1;
197     end if ;
198 end process;
199
200 -- Формування значень регістрів RO_w1 та RA_w1,
201 -- запис прочитаних даних з RAM в регістр RD_w
202 -- для інструкцій LOAD, ADD та MUL
203 -- у другій половині стадії READ
204 PRORAw2: process ( RST, R_en, RO_r1 , RA_r1 , RO_r1 , RAM_dout )
205 begin
206     if ( RST = '1 ') then
207         RO_w2 <= "000";
208         RA_w2 <= "000";
209     elsif falling_edge ( R_en ) then
210         RO_w2 <= RO_r1;
211         RA_w2 <= RA_r1;
212         if ( RO_r1 = LOAD or RO_r1 = ADD or RO_r1 = MUL ) then
213             RD_w <= RAM_dout;
214         end if ;
215     end if ;
216 end process;
217 ------------------------------------------------------------------------
218 -- Четверта стадія конвеєра (WRITE)
219 ------------------------------------------------------------------------
220 -- Передача результуючого значення тракту обробки даних
221 -- на вхідну шину пам'яті RAM
222 RAM_din <= DP_res;
223 -- Формування значення шини адреси пам'яті RAM
224 -- при виконанні різних операцій
225 PRAMADR: process ( RO_r2, RO_r1 , RO_r2, RA_r2 )
226 begin
227     if ( RST = '1 ' ) then
228         RAM_adr <= "000";
229     elsif ( RO_r1 = LOAD or RO_r1 = ADD or RO_r1 = MUL ) then
230         RAM_adr <= RA_r1;
231     elsif ( RO_r2 = STORE ) then
232         RAM_adr <= RA_r2;
233     end if;
234 end process;
235
236 -- Формування значення керуючого сигналу читання/запису
237 -- для пам'яті RAM
238 PSTORE: process ( RO_r2 )
239 begin
240     -- запис на стадії WRITE для інструкції STORE
241     if ( RO_r2 = STORE ) then
242         RAM_rw <= ' 0 ';
243     -- інакше - режим читання за замовчуванням
244     else
245         RAM_rw <= '1 ';
```

```
246       end if;
247     end process;
248
249     -- Формування значень регістрів RO_r2, RA_r2 та RD_r
250     -- у першій половині стадії WRITE
251     PRORARDr2: process ( RST, W_en, RO_w2, RA_w2, RD_w )
252     begin
253       if ( RST = '1 ' ) then
254         RO_r2 <= "000";
255         RA_r2 <= "000";
256         RD_r <= "00000000";
257       elsif rising_edge( W_en ) then
258         RO_r2 <= RO_w2;
259         RA_r2 <= RA_w2;
260         RD_r <= RD_w;
261       end if ;
262     end process;
263
264     -- Формування значень сигналів для тракту обробки даних
265     DP_op1 <= RD_r;
266     DP_ot <= RO_r2;
267     DP_en <= not W_en;
268     ------------------------------------------------------------------------
269     End Beh;
270     ------------------------------------------------------------------------
```

Лістинг 3.10. Поведінковий VHDL-опис компонента TOP

```
1   ------------------------------------------------------------------------
2   --Цифровий пристрій з мікропрограмним управлінням----------------------
3   ------------------------------------------------------------------------
4   entity TOP is
5         Port (--Вхідний порт сигналу синхронізації
6               CLK       : in      STD_LOGIC;
7               -- Вхідний порт сигналу ініціалізації
8               RST       : in      STD_LOGIC;
9               -- Вхідний порт керуючого сигналу Start
10              Start     : in      STD_LOGIC;
11              -- Вихідний порт сигналу закінчення Stop
12              Stop      : out     STD_LOGIC );
13  end TOP;
14
15  ------------------------------------------------------------------------
16  architecture Behavioral of TOP is
17  -- Декларація компоненти ЗП мікропрограми
18  component MROM is
19        Port ( RE       :   in      STD_LOGIC;
20               ADR      :   in      STD_LOGIC_VECTOR( 2 downto 0 );
21               DOUT     :   out     STD_LOGIC_VECTOR( 5 downto 0 ) );
22  end component;
23
24  -- Декларація компоненти ЗП даних
25  component MRAM is
26        Port ( RW       :   in      STD_LOGIC;
27               ADR      :   in      STD_LOGIC_VECTOR( 2 downto 0 );
28               DIN      :   in      STD_LOGIC_VECTOR( 7 downto 0 );
29               DOUT     :   out     STD_LOGIC_VECTOR( 7 downto 0 ) );
30  end component;
31
32  -- Декларація компоненти тракту обробки даних
33  component DPTH is
```

```
34        Port (   EN    :  in     STD_LOGIC;
35                 OT    :  in     STD_LOGIC_VECTOR( 2 downto 0 );
36                 OP1   :  in     STD_LOGIC_VECTOR( 7 downto 0 );
37                 RES   :  out    STD_LOGIC_VECTOR( 7 downto 0 ) );
38  end component;
39
40  — Декларація компоненти пристрою управління
41  — (компонента CTRL1 або CTRL2)
42  component CTRL2 is
43        Port (
44        CLK, RST, Start   :  in     std_logic;
45        Stop              :  out    std_logic ;
46
47        ROM_re            :  out    std_logic ;
48        ROM_adr           :  out    std_logic_vector ( 2 downto 0 );
49        ROM_dout          :  in     std_logic_vector ( 5 downto 0 );
50
51        RAM_rw            :  out    std_logic ;
52        RAM_adr           :  out    std_logic_vector ( 2 downto 0 );
53        RAM_din           :  out    std_logic_vector ( 7 downto 0 );
54        RAM_dout          :  in     std_logic_vector ( 7 downto 0 );
55
56        DP_op1            :  out    std_logic_vector ( 7 downto 0 );
57        DP_ot             :  out    std_logic_vector ( 2 downto 0 );
58        DP_en             :  out    std_logic;
59        DP_res            :  in     std_logic_vector ( 7 downto 0 ) );
60  end component
61
62
63  — Сигнали міжз'єднань з компонентою MROM
64   signal  ROM_re      :   std_logic;
65   signal  ROM_adr     :   std_logic_vector ( 2 downto 0 );
66   signal  ROM_dout    :   std_logic_vector ( 5 downto 0 );
67
68
69    — Сигнали міжз'єднань з компонентою MRAM
70    signal  RAM_rw      :   std_logic ;
71    signal  RAM_adr     :   std_logic_vector ( 2 downto 0 );
72    signal  RAM_din     :   std_logic_vector ( 7 downto 0 );
73    signal  RAM_dout    :   std_logic_vector ( 7 downto 0 );
74
75    — Сигнали міжз'єднань з компонентою DPTH
76    signal  DP_op1      :   std_logic_vector ( 7 downto 0 );
77    signal  DP_ot       :   std_logic_vector ( 2 downto 0 );
78    signal  DP_en       :   std_logic ;
79    signal  DP_res      :   std_logic_vector ( 7 downto 0 );
80    ————————————————————————————————————————————————————————————————
81    Begin
82
83    — Організація міжз'єднань усіх компонент
84
85    — Пристрій управління (CTRL1 або CTRL2 )
86    CU: CTRL2 port map (
87        CLK, RST, Start, Stop,
88        ROM_re, ROM_adr, ROM_dout,
89        RAM_rw, RAM_adr, RAM_din, RAM_dout,
90        DP_op1, DP_ot, DP_en, DP_res
91        );
92
93    — Пам`ять мікропрограми
94    ROMC: MROM port map ( ROM_re, ROM_adr, ROM_dout );
95    — Пам`ять даних
```

```
96    RAMC: MRAM port map ( RAM_rw, RAM_adr, RAM_din, RAM_dout );
97    — Тракт обробки даних
98    DPHC: DPTH port map ( DP_en, DP_ot, DP_op1, DP_res );
99
100   ——————————————————————————————————————————————————————
101    End Behavioral;
102   ——————————————————————————————————————————————————————
```

## Додаток 3

# Лістинги до розділу 4

Лістинг 4.1. Поведінковий VHDL-опис BSC-комірки

```
1   Entity BSC is
2       Port  (  PI              :    in     bit;
3                 SI              :    in     bit;
4                 Clock_DR        :    in     bit;
5                 Update_DR       :    in     bit;
6                 Shift_DR        :    in     bit;
7                 MODE            :    in     bit;
8                 SO              :    out    bit;
9                 PO              :    out    bit);
10  End BSC;
11
12  Architecture Behavioral of BSC is
13  signal s1 , s2 : bit ;
14  Begin
15
16  —
17  CAPTURE : process ( Clock_DR , Shift_DR , PI, SI )
18  begin
19      if Clock_DR = '1 ' then
20          if Shift_DR = '0' then
21            s1 <= PI;
22          else
23            s1 <= SI ;
24          end if ;
25      end if ;
26  end process;
27
28  —
29  UPDATE : process ( Update_DR, s1 )
30  begin
31      if Update_DR = '1 ' then
32          s2 <= s1;
33      end if ;
34  end process;
35
36  —
37  PO <= PI when MODE = '0' else s2;
38  SO <= s1 ;
39
40  End Behavioral;
```

Лістинг 4.2. Поведінковий VHDL-опис IRC-комірки

```
1    Entity IRC is
2          Port (PI              :     in     bit;
3                SI              :     in     bit;
4                Clock_IR        :     in     bit;
5                Update_IR       :     in     bit;
6                Shift_IR        :     in     bit;
7                Reset_bar       :     in     bit;
8                nTRST           :     in     bit;
9                — ініціалізоване значення
10               — тіньового регістра
11               reset_value     :     in     bit;
12               SO              :     out    bit;
13               PO              :     out    bit );
14   End IRC;
15
16   Architecture Behavioral of IRC is
17   signal s1 , s2 : bit ;
18   Begin
```

```
19
20      s2 <= Reset_bar and nTRST;
21
22      —
23      CAPTURE : process ( Clock_IR, Shift_IR , PI, SI )
24      begin
25          if ( Clock_IR = '1' ) then
26              if ( Shift _____ IR = '0' ) then
27                  s1 <= PI;
28              else
29                  s1 <= SI;
30              end if ;
31          end if ;
32      end process;
33
34      —
35      UPDATE : process ( s2, reset_valu, Update_IR,
s1 )
36      begin
37          if ( s2 = '0' ) then
38              PO <= reset_value;
39          elsif ( Update_IR = '1' ) then
40              PO <= s 1 ;
41          end if ;
42      end process;
43
44      —
45      SO <= s 1 ;
46
47      End Behavioral ;
```

Лістинг 4.3. Структурний опис регістра граничного сканування BSR

```
1       Entity BSR is
2           — Кількість елементів пам'яті регістра
3           Generic ( length : integer := 4 );
4
5           Port (
6                   — керуючі сигнали
7                   Clock_DR, Update_Dr, Shift_Dr , MODE : in bit;
8                   — Вхід послідовних даних
9                   SI    :   in      bit ;
10                  — Вихід послідовних даних
11                  SO    :   out     bit ;
12                  — Входи паралельних даних
13                  PI    :   in      bit_vector ( length -1 downto 0 );
14                  — Виходи паралельних даних
15                  PO    :   out     bit_vector ( length -1 downto 0 ) );
16      End BSR ;
17
18      Architecture Struct of BSR is
19      — Декларація компоненти комірки регістра граничного сканування
20      component BSC port (
21                  PI              :   in      bit ;
22                  SI              :   in      bit ;
23                  Clock_DR        :   in      bit;
24                  Update_DR       :   in      bit ;
25                  Shift_DR        :   in      bit ;
26                  MODE            :   in      bit ;
27                  SO              :   out     bit ;
28                  PO              :   out     bit );
29      end component ;
30
31      — Наскрізна лінія перенесення
32      signal internal : bit_vector ( length downto 0 );
33
34      Begin
35
```

```
36          — Послідовне розміщення компонент BSC_i
37          — <- BSC_0 <- BSC_1 <- ... <- BSC_length-1
38          BSR : for i in 1 to length generate
39              BSCi : BSC port map
40                                  ( PI ( i -1 ) , internal ( i ), Clock_DR, Update_DR,
41                                    S h if t_ D R , MODE, internal ( i -1 ), PO( i -1 ) );
42          end generate;
43
44          — Підключення компоненти BSC_0 до вихідного порту SO
45          SO <= internal ( 0 ) ;
46
47          — Підключення вхідного порту SI до входу компоненти BSC_length-1
48           internal ( length ) <= SI;
49
50       End Struct ;
```

Лістинг 4.4. Структурний опис регістра інструкцій IR

```
1   Entity IR is
2       — Кількість елементів пам'яті регістра
3       Generic ( length : integer := 4 );
4       Port (
5           — керуючі сигнали
6           Shift_IR  ,  Clock_IR  ,  Update_IR,
7           Reset_bar, nTRST,
8           — Вхід послідовних даних
9           SI      : in    bit;
10          — Вихід послідовних даних
11          SO      : out   bit ;
12          — Входи паралельних даних
13          PI      : in    bit_vector ( length -1 downto 0 );
14          — Виходи паралельних даних
15          PO      : out   bit_vector ( length -1 downto 0 )
16                  );
17  End IR;
18
19  Architecture Struct of IR is
20  — Декларація компоненти комірки регістра інструкцій
21  component IRC
22          Port ( PI               :   in    bit ;
23                 SI               :   in    bit ;
24                 Clock_IR         :   in    bit;
25                 Update_IR        :   in    bit ;
26                 Shift_IR         :   in    bit ;
27                 Reset_bar        :   in    bit ;
28                 nTRST            :   in    bit ;
29                 reset_value      :   in    bit ;
30                 SO               :   out   bit ;
31                 PO               :   out   bit );
32  end component ;
33
34
35  — Наскрізна лінія перенесення
36  signal internal : bit_vector ( length downto 0 );
37
38  — Вектор ініціалізуючих значень тіньового регістра
39  signal RESET_VALUE_VECTOR: BIT_VECTOR( length -1 downto 0 );
40
41  — Вектор вхідних значень
42  signal DATA_INPUT_VECTOR : BIT_VECTOR( length -1 downto 0 );
43
44  Begin
45
46      RESET_VALUE_VECTOR <= ( others = >'1 ' );
47      DATA_INPUT_VECTOR <= PI ( length -1 downto 2 ) & '0' & '1';
48
49      — Послідовне розміщення компонент IRC_i
```

```
50    — <- IRC_0 <- IRC_1 <- ... <- IRC_length-1
51     IREG : for i in 1 to length generate
52         I RCi : IRC port map
53                         ( DATA_INPUT_VECTOR( i -1 ) , internal ( i ), Clock_IR ,
54                            Update_IR , Shif t_I R , Reset_bar ,
55                            nTRST, RESET_VALUE_VECTOR( i -1 ), internal( i -1 ),
56                            PO( i -1) );
57     end generate ;
58
59    — Підключення компоненти IRC_0 до вихідного порту SO
60    SO <= internal ( 0 ) ;
61
62    — Підключення вхідного порту SI до входу компоненти IRC_length-1
63     internal (length ) <= SI ;
64
65  End Struct ;
```

Лістинг 4.5. Поведінковий опис декодера JTAG-інструкцій

```
1   Entity IRDecoder is
2         — Розрядність однієї інструкції
3         Generic ( length : integer := 4 );
4         Port (
5                  — керуючі сигнали
6                  Shift_DR , Clock_DR, Update_DR : in bit;
7                  — Вхідна шина інструкції
8                  IR : bit_vector ( length -1 downto 0 );
9                  — Вихідні сигнали управління
10                 MODE, Select_BR , Shift_BR , Clock_BR ,
11                 Shift_BSR , Clock_BSR, Update_BSR : out bit
12               );
13  End IRDecoder;
14
15  Architecture Beh of IRDecoder is
16
17  — Обов'язкові JTAG-інструкції
18  type INSTRUCTIONS is
19   ( EXTEST, SAMPLE_PRELOAD, IDCODE, BYPASS );
20
21   signal I : INSTRUCTIONS;
22
23   Begin
24
25     — Процес декодування інструкції IR
26     DEC: process ( IR )
27     begin
28        case bit_vector'( IR ( 1 ) , IR ( 0 ) ) is
29           when   "00"   =>   I   <=   EXTEST;
30           when   "01"   =>   I   <=   SAMPLE_PRELOAD;
31           when   "10"   =>   I   <=   IDCODE;
32           when   "11"   =>   I   <=   BYPASS;
33        end case;
34     end process ;
35
36     — Формування керуючих і синхронізуючих сигналів
37     MODE          <=   '1 '          when ( I = EXTEST )
38                                       else '0 ';
39
40     Select_BR     <=   '1'           when ( I = BYPASS or I = IDCODE )
41                                       else '0 ';
42
43     Shift_BR      <=   Shift_DR ;
44     Shift_BSR     <=   Shift_DR ;
45
46     Update_BSR    <=   Update_DR     when ( I = EXTEST or I = SAMPLE_PRELOAD )
```

```
47                                  else '0 ';
48
49    Clock_BR      <=   Clock_DR   when ( I = BYPASS or I = IDCODE )
50                                  else '1 ';
51
52    Clock_BSR     <=   Clock_DR   when ( I = EXTEST or I = SAMPLE_PRELOAD )
53                                  else '1 ';
54
55  End Beh;
```

Лістинг 4.6. Поведінковий опис компоненти TAP_FSM

```
1     Library ieee;
2     Use ieee . std_logic_11 64 . all ;
3     Use ieee . std_logic_unsigned . all ;
4
5     Package JP is
6        — Усі можливі стани TAP-автомата
7        Type TAP_STATE is
8        (
9           RESET_s, RUN_IDLE_s, SELECT_DR_s, CAPTURE_DR_s,
10          SHIFT_DR_s , EXIT1_DR_s , PAUSE_DR_s, EXIT2_DR_s ,
11          UPDATE_DR_s, SELECT_IR_s , CAPTURE_IR_s, SHIFT_IR_s ,
12          EXIT1 _IR_s , PAUSE_IR_s , EXIT2_IR_s , UPDATE_IR_s
13       );
14    End JP;
15
16    Package Body JP is
17    End JP;
18     ————
19
20    Library ieee;
21    Use ieee . std_logic_1164 . all ;
22    Use ieee . std_logic_unsigned . all ;
23
24    Library Work;
25    Use Work. JP. all ;
26
27    Entity TAP_FSM is
28       Port (
29                — Керуючі сигнали TAP-автомата
30                TMS, TCK, nTRST     : in    BIT;
31                — Вихідна шина поточного стану
32                — TAP-автомата
33                S                   : out   AP_STATE );
34    End TAP_FSM;
35
36    Architecture Beh of TAP_FSM is
37
38    type STATE_ARRAY is array (TAP_STATE, 0 to 1) of TAP_STATE;
39
40    — Таблиця переходів станів TAP-автомата
41    constant T : STATE_ARRAY := (
42     (   RUN_IDLE_s,     RESET_s       ),
43     (   RUN_IDLE_s,     SELECT_DR_s   ),  (   CAPTURE_DR_s,   SELECT_IR_s   ),
44     (   SHIFT_DR_s,     EXIT1_DR_s    ),  (   SHIFT_DR_s,     EXIT1_DR_s    ),
45     (   PAUSE_DR_s,     UPDATE_DR_s   ),  (   PAUSE_DR_s,     EXIT2_DR_s    ),
```

```
46      (   SHIFT_DR_s,      UPDATE_DR_s  ),  (   RUN_IDLE_s,       SELECT_DR_s  ),
47      (   CAPTURE_IR_s,    RESET_s      ),  (   SHIFT_IR_s,       EXIT1_IR_s   ),
48      (   SHIFT_IR_s,      EXIT1_IR_s   ),  (   PAUSE_IR_s,      UPDATE_IR_s  ),
49      (   PAUSE_IR_s,      EXIT2_IR_s   ),  (   SHIFT_IR_s,      UPDATE_IR_s  ),
50      (   RUN_IDLE_s,      SELECT_DR_s  )
51     );
52
53   Begin
54
55      process ( TCK, nTRST )
56      variable Si: TAP_STATE;
57      begin
58          if ( nTRST = '0' ) then
59              Si := RESET_s;
60          elsif rising_edge( TCK ) then
61              if ( TMS = '1' ) then
62                  Si := T( Si , 1 );
63              else
64                  Si := T( Si , 0 );
65              end if ;
66          end if ;
67          S <= Si;
68      end process ;
69
70   End Beh;
```

Лістинг 4.7. Поведінковий опис компоненти TAP_OUT

```
1    Library ieee;
2    Use ieee . std_logic_1164.all ;
3    Use ieee . std_logic_unsigned.all ;
4
5    — Підключення VHDL-пакета JP
6     Library Work;
7    Use Work. JP. all ;
8
9     Entity TAP_OUT is
10        Port
11            (
12              — сигнал синхронізації TAP-автомата
13              TCK    :    in     bit;
14              — Вхідна шина стану TAP-автомата
15              S      :    in     TAP_STATE;
16              — Вихідні керуючі сигнали
17              Reset_bar,     Select_IR,       Enable_TDO,
18              Shift_IR,      Clock_IR,        Update_IR,
19              Shift_DR,      Clock_DR ,       Update_DR    :    out   bit
20               );
21   End TAP_OUT;
22
23   Architecture Beh of TAP_OUT is
24             Begin
25        process ( TCK, S )
26        begin
27            if rising_edge ( TCK ) then
28              — Формування сигналу ініціалізації
29               if ( S = RESET_s ) then
30                   Reset_bar <= '0 ';
31               else
```

```
32              Reset_bar <= '1 ';
33            end if ;
34            — Формування сигналу дозволу виходу TDO
35            if ( S = SHIFT_IR_s or S = SHIFT_DR_s ) then
36              Enable_TDO <= '1 ';
37            else
38              Enable_TDO <= '0 ';
39            end if ;
40            — Формування сигналу зсуву регістра IR
41            if ( S = SHIFT_IR_s ) then
42              Shift __ I R <= '1 ';
43            else
44              Shift __ I R <= '0';
45            end if ;
46            — Формування сигналу зсуву регістра DR
47            if ( S = SHIFT_DR_s ) then
48              Shift_DR <= '1 ';
49            else
50              Shift_DR <= '0 ';
51            end if ;
52          end if;
53        end process;
54
55        process ( TCK )
56        begin
57          — Формування сигналу синхронізації для регістра IR
58          if ( TCK = '0' and ( S = CAPTURE_IR_s or S = SHIFT_IR_s ) ) then
59            Clock_IR <= '0 ';
60          else
61            Clock_IR <= '1 ';
62          end if;
63          — Формування сигналу синхронізації для регістра BSR
64          if ( TCK = '0' and ( S = CAPTURE_DR_s or S = SHIFT_DR_s ) ) then
65            Clock_DR <= '0 ';
66          else
67            Clock_DR <= '1 ';
68          end if;
69          — Формування сигналу поновлення регістра IR
70          if ( TCK = '0' and S = UPDATE_IR_s ) then
71            Update_IR <= '1 ';
72          else
73            Update_IR <= '0 ';
74          end if;
75          — Формування сигналу поновлення регістра BSR
76          if ( TCK = '0' and S = UPDATE_DR_s ) then
77            Update_DR <= '1 ';
78          else
79            Update_DR <= '0 ';
80          end if;
81        end process;
82
83          — Формування сигналу вибору регістра IR
84          Select_I R <= '1 '
85              when ( S = RESET_s          or S = RUN_IDLE_s     or
86                     S = CAPTURE_IR_s     or S = SHIFT_IR_s     or S = EXIT 1 _IR_s    or
87                     S = PAUSE_IR_s       or S = EXIT2_IR_s     or S = UPDATE_IR_s )
88            else '0 ';
89      End Beh;
```

Лістинг 4.8. Змішаний VHDL-опис TAP-контролера

```
1   Library IEEE;
2   Use IEEE . std_logic_11 64 . all ;
3
4   — Підключення VHDL-пакета JP
5    Library Work ;
```

```
6   Use Work. JP. all ;
7
8   Entity TAP_CTRL is
9     — Розмірність регістра граничного сканування
10      Generic ( length : integer := 4 );
11      Port (
12              — Входи TAP-контролера
13              TMS, TCK, TDI, nTRST      :   in    bit;
14              — Вихід послідовних тестових даних
15              TDO                       :   out   std_logic ;
16              — Вхід даних від регістра BSR
17              BSR_SO                    :   in    bit;
18              — Виходи керуючих сигналів
19              Shift_BSR , Clock_BSR,
20              Update_BSR, MODE          :   out   bit
21              );
22  End TAP_CTRL;
23
24  Architecture Mix of TAP_CTRL is
25
26  — Декларація компоненти TAP-автомата
27  component TAP_FSM
28  port ( TMS, TCK, nTRST      : in    bit;
29         S                    : out   TAP_STATE );
30  end component;
31
32   — Декларація компоненти, що виробляє керуючі сигнали
33   component TAP_OUT
34   port ( TCK                 :   in    bit;
35          S                   :   in    TAP_STATE;
36           Reset_bar ,
37           Select_IR,
38           Enable_TDO,
39           Shift __I R ,
40           Clock_IR ,
41           Update_IR ,
42           Shift_DR ,
43           Clock_DR ,
44           Update_DR         :   out   bit );
45   end component;
46
47   — Декларація компоненти регістра інструкцій
48   component IR
49   generic ( length : INTEGER := 4 );
50   port      (Shift_IR ,
51              Clock_IR ,
52              Update_IR,
53              Reset_bar ,
54              nTRST, SI  : in      bit;
55              SO         : out     bit;
56              PI         : in      bit_vector ( length -1 downto 0 );
57              PO         : out     bit_vector ( length -1 downto 0 ) );
58   end component;
59
60   — Декларація компоненти декодера JTAG-інструкцій
61   component IRDecoder
62   generic (  length : INTEGER := 4 );
63   port      (  S h if t_ D R , Clock_DR , Update_DR : in bit;
64                IR                :   in    bit_vector (length -1 downto 0);
65                MODE,
66                Select_BR ,
67                Shift_BR ,
68                Clock_BR ,
69                Shift_BSR ,
70                Clock_BSR ,
71                Update_BSR    :   out   bit );
72   end component;
73
74   — Декларація компоненти регістра обходу
```

```
  component BR
  port ( SI          :   in    bit ;
         Shift_DR    :   in    bit ;
         Clock_DR    :   in    bit;
         SO          :   out   bit );
  end component;

  — Внутрішні сигнали TAP-контролера
  signal
  Reset_bar,      — Сигнал ініціалізації
  Select_IR,      — Сигнал вибору регістра IR
  Enable_TDO,     — Сигнал дозволу вихідної лінії TDO
  Shift_IR,       — Керуючий сигнал зсуву регістра IR
  Clock_IR,       — Сигнал синхронізації регістра IR
  Update_IR,      — Сигнал поновлення регістра IR
  Shift_DR,       — Керуючий сигнал зсуву регістра DR
  Clock_DR,       — Сигнал синхронізації регістра DR
  Update_DR,      — Сигнал поновлення регістра DR
  IR_SO,          — Вихід послідовних даних регістра IR
  BR_SO,          — Вихід послідовних даних регістра BR
  TDO_reg,        — Елемент зберігання сигналу TDO
  TDO_data,       — Загальний вихідний сигнал послідовних даних
  TDR_SO,         — Вихідний сигнал регістрів BSR та BR
  Select_BR,      — Сигнал вибору регістра BR
  Clock_BR,       — Сигнал синхронізації регістра BR
  Shift BR        — Керуючий сигнал зсуву регістра BR
  : bit;

 signal
 IR_PI ,          — Входи паралельних даних регістра IR
 IR_PO            — Виходи паралельних даних регістра IR
 : bit_vector ( length -1 downto 0 );
— Поточний стан TAP-автомата
signal S : TAP_STATE;

Begin

   — Ініціалізуюче значення регістра інструкцій
    IR_PI    <=    "1001";

   — Підключення буфера з трьома станами до вихідного порту TDO
   TDO      <=    TO_STDULOGIC( TDO_reg ) when Enable_TDO = '1' else 'Z';

   — Опис асинхронного елемента пам'яті для зберігання значення
   — вихідного сигналу TDO
    R1 : process ( TCK )
    begin
        if ( TCK = '0' ) then
            TDO_reg <= TDO_data;
        end if ;
    end process ;

   — мультиплексор сигналу TDO
   TDO_data   <=  IR_SO    when Select_IR = '1' else TDR_SO;

   — мультиплексор сигналу TDR_SO
   TDR_SO     <=  BR_SO    when Select_BR = '1' else BSR_SO;

   — підключення компоненти TAP_FSM
   TC1    : TAP_FSM        port map (  TMS, TCK, nTRST, S );

   — підключення компоненти TAP_OUT
   TC2    : TAP_OUT        port map (  TCK, S, Reset_bar,
                                       Select_IR , Enable_TDO,
                                       Shift_IR , Clock_IR ,
                                       Update_IR , Shift_DR ,
                                       Clock_DR, Update_DR );

    — Підключення регістра інструкцій
```

```
144       IR1   : IR                  generic map ( length => 4 )
145                                   port map (  Shift_IR , Clock_IR,
146                                               Update_IR , Reset_bar,
147                                               nTRST, TDI, IR_SO,
148                                               IR_PI , IR_PO );
149
150       — Дешифратор інструкцій TAP-контролера
151       DEC1 : IRDecoder            generic map ( length => 4 )
152                                   port map (  Shift_DR , Clock_DR,
153                                               Update_DR , IR_PO,
154                                               MODE, Select_BR ,
155                                               Shift_BR , Clock_BR ,
156                                               Shift_BSR , Clock_BSR ,
157                                               Update_BSR );
158
159       — регістр обходу
160       BR1   : BR                  port map ( TDI, Shift_BR , Clock_BR, BR_SO );
161
162     End MIX;
```

Лістинг 4.9. Початковий опис компоненти ASIC

```
1     Library IEEE;
2     Use IEEE . std_logic_11 64 . all ;
3     Entity ASIC is
4        port (a, b, c   :   in    bit ;
5              z         :   out   bit);
6     End ASIC;
7     Architecture Beh of ASIC is
8     Begin
9          z <= a xor b xor c;
10    End Beh;
```

Лістинг 4.10. Структурний опис JTAG-сумісного пристрою

```
1      Library IEEE;
2      Use IEEE . std_logic_1164.all ;
3
4      Entity ASIC_JTAG is
5         Port (
6               TMS, TCK, TDI, nTRST    :   in      bit;
7               TDO                     :   out     std_logic;
8               a_PAD, b_PAD, c_PAD     :   in      bit;
9               z_PAD                   :   out     bit
10                 );
11     End ASIC_JTAG;
12
13     Architecture Struct of ASIC_JTAG is
14     — Декларація вихідної цифрової компоненти
15     component ASIC
16     port ( a, b, c      :   in    bit ;
17            z            :   out   bit );
18     end component;
19     — Декларація TAP-контролера
20     component TAP_CTRL
21     generic  ( length : INTEGER := 4 );
22     port       ( TMS,
23                  TCK,
24                  TDI,
25                  nTRST         :  in     bit;
26                  TDO           : out     std_logic ;
27                  BSR_SO        :  in     bit ;
28                  Shift_BSR ,
```

```
29                Clock_BSR ,
30                Update_BSR,
31                MODE          :  out      bit );
32    end component;
33    — Декларація регістра граничного сканування
34    component BSR
35    generic ( length : INTEGER := 4 );
36    port          ( Clock_DR ,
37                Update_Dr ,
38                Shift_Dr ,
39                MODE          : in        bit ;
40                SI            : in        bit ;
41                SO            : out       bit ;
42                PI            : in        bit_vector ( length -1 downto 0 );
43                PO            : out       bit_vector ( length -1 downto 0 ) );
44    end component;
45
46    — Розрядність регістра граничного сканування
47    constant BSR_length : INTEGER := 4;
48
49    — Внутрішні з'єднувальні сигнали
50    signal BSR_SO, MODE, Shift_BSR,
51                Clock_BSR, Update_BSR          : bit;
52    signal BSR_PI, BSR_PO                      : bit_vector ( BSR_length-1 downto 0 );
53    signal a, b, c, z                          : bit ;
54
55    Begin
56
57       — Підключення паралельних входів регістра BSR
58        BSR_PI <= a_PAD & b_PAD & c_PAD & z;
59
60       — Підключення паралельних виходів регістра BSR
61          a           <=    BSR_PO( 3 );
62          b           <=    BSR_PO( 2 );
63          c           <=    BSR_PO( 1 );
64          z_PAD       <=    BSR_PO( 0 );
65
66       — Підключення вихідної компоненти
67        ASIC1  :  ASIC          port map ( a, b, c, z );
68
69       — Підключення TAP-контролера
70        C1       :  TAP_CTRL   generic map ( length => BSR_length )
71                               port map       ( TMS, TCK, TDI, nTRST,
72                                                TDO, BSR_SO, Shift_BSR,
73                                                Clock_BSR , Update_BSR ,
74                                                MODE );
75
76       -- Підключення регістра граничного сканування
77        BSR1   :  BSR           generic map ( length => BSR_length )
78                               port map       ( Shift_BSR , Clock_BSR ,
79                                                Update_BSR, MODE, TDI,
80                                                BSR_SO, BSR_PI, BSR_PO );
81
82    End Struct ;
```

**Додаток 4**

# Лістинги до розділу 5

## Лістинг 5.1. Поведінковий опис повного суматора

```
1   -----------------------------------------------------------------------
2   -— Компонента повного суматора
3   -----------------------------------------------------------------------
4   Entity FADDER is
5       Port (
6           — Вхідний порт розряду першого доданка
7           A0   :   in    std_logic;
8           — Вхідний порт розряду другого доданка
9           B0   :   in    std_logic;
10          — Вхідний порт перенесення з молодшого розряду
11          P0   :   in    std_logic;
12          — Вихідний порт сигналу суми
13          S1   :   out   std_logic ;
14          — Вихідний порт сигналу перенесення
15          P1   :   out   std_logic ;
16           );
17  End FADDER;
18  ------------------------------------------------------------------------
19  Architecture Behavioral of FADDER is
20  — Поведінковий опис суматора
21  Begin
22
23      — функція суми
24      S1 <= P0 xor A0 xor B0;
25      — функція передачі
26      P1 <= ( P0 and ( A0 xor B0 ) ) or ( A0 and B0 );
27
28  End Behavioral ;
29  ------------------------------------------------------------------------
```

## Лістинг 5.2. Функціональна обфускація повного суматора

```
1   -----------------------------------------------------------------------
2    -—…
3   -----------------------------------------------------------------------
4   Architecture Behavioral of FADDER is
5   — Опис таблиці істинності
6   subtype     b3    is std_logic_vector ( 2 downto 0 );
7   type        ttt   is array (0 to 7 ) of b3;
8   constant    tt:   ttt := ( "000","100","100","010","100","010" ,"010","110" );
9   signal      arg   :   std_logic_vector ( 2 downto 0 );
10  signal      res   :   std_logic_vector ( 2 downto 0 );
11
12  — Поведінковий опис суматора
13  Begin
14
15  — Формування рядка аргументів для таблиці істинності
16  arg <= P0 & AO & B0;
17
18  — Вибірка значень з таблиці істинності
19  Main: process ( arg )
20  begin
21      res <= tt( conv_integer( arg ) );
22  end process;
23
24  — функція суми
25  S1  <=  res( 2 ) or res ( 0 );
26  — функція передачі
27  P1  <=  res( 1 ) or res ( 0 );
28
```

```
29  End Behavioral ;
30  ----------------------------------------------------------------------------
```

Лістинг 5.3. Поведінковий опис мультиплексора і арбітра

```
1  ----------------------------------------------------------------------------
2        -- Library IEEE;
3        Use       IEEE.STD_LOGIC_1164.ALL;
4        Use       IEEE.STD_LOGIC_ARITH.ALL;
5        Use       IEEE.STD_LOGIC_UNSIGNED.ALL;
6        ----------------------------------------------------------------------
7  -- мультиплексор 2x1
8  ----------------------------------------------------------------------------
9  Entity MUX2x1 is
10           Port ( IN0     :  in    STD_LOGIC; -- Нульовий вхідний порт
11                  IN1     :  in    STD_LOGIC; -- Одиничний вхідний порт
12                  SEL     :  in    STD_LOGIC; -- Вхідний порт сигналу вибору
13                  F       :  out   STD_LOGIC ); -- вихідний порт
14 End MUX2x1;
15 ----------------------------------------------------------------------------
16
17 Architecture Beh of MUX2x1 is
18 Begin
19
20           F <= I N0 when ( SEL = '0' ) else IN1 ;
21
22 End Beh;
23 ----------------------------------------------------------------------------
24 -- Синхронний D-тригер
25 ----------------------------------------------------------------------------
26 Entity DFF is
27           Port (D   :  in   STD_LOGIC;      -- Вхідний порт даних
28                 CLK :  in   STD_LOGIC;      -- Вхідний порт синхросигналу
29                 R   :  in   STD_LOGIC;      -- Вхідний порт асинхронного скидання
30                 Q   :  out  STD_LOGIC );    -- вихідний порт
31 End DFF;
32 ----------------------------------------------------------------------------
33
34 Architecture Beh of DFF is
35
36       signal state : std_logic;                    -- Поточний стан тригера
37
38 Begin
39
40           MAIN: process ( D, CLK, R )
41           begin
42               -- Перевірка умови асинхронного скидання тригера
43               if ( R = ' 1' ) then
44                   state <= '0 ';
45               -- Перевірка умови настання переднього фронту синхросигналу
46               elsif rising_edge( CLK ) then
47                   state <= D;
48               end if ;
49           end process;
50
51           -- Передача значення поточного стану тригера на вихід
52           Q <= state;
53
54 End Beh;
55 ----------------------------------------------------------------------------
```

Лістинг 5.4. Структурний опис компоненти LINK

```
1  ----------------------------------------------------------------
2  -- Ланка конфігурованого шляху
3  ----------------------------------------------------------------
4  Entity LINK is
5          Port ( IN_A     : in   STD_LOGIC;      -- Вхідний порт сигналу A
6                 IN_B     : in   STD_LOGIC;      -- Вхідний порт сигналу B
7                 SEL      : in   STD_LOGIC;      -- Вхідний порт сигналу вибору
8                 OUT_A    : out  STD_LOGIC;      -- Вихідний порт сигналу A
9                 OUT_B    : out  STD_LOGIC );    -- Вихідний порт сигналу B
10 End LINK;
11 ----------------------------------------------------------------
12 Architecture Behavioral of LINK is
13 -- Декларація компоненти мультиплексора
14 component MUX2x1 is
15             Port ( IN0  :    in   STD_LOGIC;
16                    IN1  :    in   STD_LOGIC;
17                    SEL  :    in   STD_LOGIC;
18                    F    :    out  STD_LOGIC );
19 end component;
20
21
22 -- Структурний опис компоненти LINK
23 Begin
24
25         -- "Верхній" мультиплексор ланки MUX(0,j)
26         MUX_0j: MUX2x1 port map ( IN_A, IN_B, SEL, OUT_A );
27
28         -- "Нижній" мультиплексор ланки MUX(1,j)
29         MUX_1j: MUX2x1 port map ( IN_B, IN_A, SEL, OUT_B );
30
31 End Behavioral ;
32 ----------------------------------------------------------------
```

Лістинг 5.5. Налагоджуваний VHDL-опис компоненти CHAIN

```
1  ----------------------------------------------------------------
2  -- Конфігуровані пари шляхів
3  ----------------------------------------------------------------
4  Entity CHAIN is
5          Generic  (  N          :  integer := 4 );          -- Довжина шляхів (кількість ланок LINK)
6
7             Port  (  IN_A       :  in     STD_LOGIC;     -- Вхідний порт сигналу A
8                      IN_B       :  in     STD_LOGIC;     -- Вхідний порт сигналу B
9                      OUT_A      :  out    STD_LOGIC;     -- Вихідний порт сигналу A
10                     OUT_B      :  out    STD_LOGIC;     -- Вихідний порт сигналу B
11
12                     -- Вхідна шина сигналів управління ланками
13                     C          :  in     STD_LOGIC_VECTOR ( N_1 downto 0 ) );
14 End CHAIN;
15 ----------------------------------------------------------------
16 Architecture Beh of CHAIN is
17
18 -- Декларація компоненти ланки LINK
19 component LINK is
20             Port ( IN_A     : in   STD_LOGIC;
21                    IN_B     : in   STD_LOGIC;
22                    SEL      : in   STD_LOGIC;
23                    OUT_A    : out  STD_LOGIC;
24                    OUT_B    : out  STD_LOGIC );
25 end component;
26
27 -- Декларація множини внутрішніх сигналів міжз'єднань ланок
28 signal a,b: std_logic_vector ( N downto 0 );
29
```

```
30 — Структурний опис компоненти CHAIN
31 Begin
32
33          — Вхідні сигналі першої ланки LINK_0
34          a ( 0 ) <= IN_A;
35          b ( 0 ) <= IN_B;
36
37          — Формування шляхів, що конфігуруються з N ланок
38          GCHAIN_N: for i in 0 to N_1 generate
39          LINKi: LINK port map ( a ( i ) , b ( i ), C( i ), a( i+1 ), b( i+1 ) );
40          end generate;
41
42          — Вихідні сигнали останньої ланки LINK_N_1
43          OUT_A <= a( N );
44          OUT_B <= b( N );
45
46 End Beh ;
47 ————————————————————————————————————————————————————
```

Лістинг 5.6. Поведінковий VHDL-опис генератора PG

```
1 ————————————————————————————————————————————————————
2 — Генератор одиночного імпульсу
3 ————————————————————————————————————————————————————
4 Entity PG is
5           Port ( RST      :  in     STD_LOGIC;      — Вхідний порт асинхронного
6                                                     — сигналу скидання
7                  GO       :  in     STD_LOGIC;      — Вхідний порт асинхронного
8                                                     — сигналу дозволу
9                  CLK      :  in     STD_LOGIC;      — вхідний порт
10                                                    — сигналу синхронізації
11                 OUT_A    :  out    STD_LOGIC;      — Вихідні порти імпульсу
12                 OUT_B    :  out    STD_LOGIC );
13      End PG ;
14      ————————————————————————————————————————————————————————————————————
15      Architecture Behavioral of PG is
16      — Сигнал генерованого імпульсу
17      signal  state  :   std_logic;
18      — Дворозрядний лічильник синхроімпульсів CLK
19      signal  cnt    :   std_logic_vector ( 1 downto 0 );
20
21      — Поведінковий опис генератора
22      Begin
23
24          MAIN: process ( RST, GO, CLK, cnt )
25          begin
26              —  Перевірка умови асинхронного скидання генератора і лічильника
27              if ( RST = '1 ' ) then
28                  state <= '0 ';
29                  cnt <= "00";
30              — Перевірка умови настання переднього фронту синхросигналу
31              elsif rising_edge ( CLK ) then
32                  — Перевірка умови дозволу генерування імпульсу
33                  if ( GO = '1 ' ) then
34                      — Якщо значення лічильника досягло значення 2,
35                      if ( cnt = "10" ) then
36                        — закінчуємо генерування імпульсу
37                        state <= '0';
38                      — Інакше
39                      else
40                        — продовжуємо рахунок
41                        cnt <= cnt + 1;
42                        — і формуємо імпульс
43                        state <= not state;
44                      end if ;
45                  end if ;
46              end if ;
```

```
47          end process;
48
49      — Передаємо дві копії сформованого імпульсу на вихідні порти
50      OUT_A <= state;
51      OUT_B <= state;
52
53      End Behavioral ;
54      ——————————————————————————————————————————————————————
```

Лістинг 5.7. Структурний VHDL-опис компоненти APUF

```
1 ——————————————————————————————————————————————————————
2 — Компонента PUF типу "арбітр "
3 ——————————————————————————————————————————————————————
4 Entity APUF is
5               Generic ( N : integer: = 4 );
6               Port  ( RST   :   in      STD_LOGIC;
7                       GO    :   in      STD_LOGIC;
8                       CLK   :   in      STD_LOGIC;
9                       C     :   in      STD_LOGIC_VECTOR ( N-1 downto 0 );
10                      R     :   out     STD_LOGIC );
11      End APUF;
12      ——————————————————————————————————————————————————————
13      Architecture Struct of APUF is
14
15      — Декларація компоненти генератора імпульсу
16      component PG is
17            Port (  RST     :   in      STD_LOGIC;
18                    GO      :   in      STD_LOGIC;
19                    CLK     :   in      STD_LOGIC;
20                    OUT_A   :   out     STD_LOGIC;
21                    OUT_B   :   out     STD_LOGIC );
22      end component;
23
24      — Декларація компоненти конфігурованих шляхів
25      component CHAIN is
26          Generic ( N : integer );
27            Port (  IN_A    :   in      STD_LOGIC;
28                    IN_B    :   in      TD_LOGIC;
29                    OUT_A   :   out     STD_LOGIC;
30                    OUT_B   :   out     STD_LOGIC;
31                    C       :   in      STD_LOGIC_VECTOR ( N-1 downto 0 ) );
32      end component;
33
34      — Декларація компоненти арбітра
35      component DFF is
36            Port (  D       :   in      STD_LOGIC;
37                    CLK     :   in      STD_LOGIC;
38                    R       :   in      STD_LOGIC;
39                    Q       :   out     STD_LOGIC );
40      end component;
41
42      — Внутрішні сигнали міжз'єднань
43      signal  a_in , b_in         : std_logic;
44      signal  a_out , b_out       : std_logic;
45
46      — Структурний опис компоненти APUF
47      Begin
48
49      PG_unit:            PG          port map        ( RST, GO, CLK, a_in , b_in );
50      CHAIN_unit:         CHAIN       generic map     ( N )
51      port map ( a_in , b_in , a_out , b_out , C );
52      ARBITER_unit: DFF port map ( a_out, b_out , RST, R );
53      End Struct ;
```

## Лістинг 5.8. Структурний VHDL-опис кільцевого генератора

```
1.  ------------------------------------------------------------------------
2.  -- Компонента RO кільцевого генератора
3.  ------------------------------------------------------------------------
4.  Entity RO is
5.      -- Коефіцієнт масштабування ланцюга зворотного зв'язку
6.      Generic ( N         :       integer := 3 );
7.      Port    ( -- Вхідний порт сигналу дозволу
8.                EN        :   in  STD_LOGIC;
9.                -- Вихідний порт генератора
10.               ROOUT     :   out STD_LOGIC );
11. End RO;
12. ------------------------------------------------------------------------
13. Architecture Struct of RO is
14.
15. -- Декларація компоненти інвертора
16. component INV1 is
17.       Port ( IN0      :   in  STD_LOGIC;
18.              OUT0     :   out STD_LOGIC );
19. end component;
20.
21. -- Декларація компоненти логічного елемента І
22. component AND21 is
23.       Port ( IN0      :   in  STD_LOGIC;
24.              IN1      :   in  STD_LOGIC;
25.              OUT0     :   out STD_LOGIC );
26. end component;
27.
28. -- Внутрішні сигнали міжз'єднань інверторів у ланцюзі зворотного зв'язку
29. signal  s:  std_logic_vector ( N downto 0 );
30.
31. -- Структурний опис компоненти RO
32. Begin
33.
34.     -- Формування негативного зворотного зв'язку з інверторів
35.     NFEEDBACK: for i in 0 to N_1 generate
36.         IN Vi: INV1 port map ( s( i ), s( i+1 ) );
37.     end generate;
38.
39.     -- Підключення елемента І
40.     ANDGATE: AND21 port map ( EN, s ( N ) , s ( 0 ) );
41.
42.     -- Передача сформованих імпульсів
43.     -- на вихідний порт генератора
44.     ROOUT <= s ( 0 ) ;
45.
46. End Struct ;
47. ------------------------------------------------------------------------
```

## Лістинг 5.9. VHDL-опис мультиплексорів

```
1.  ------------------------------------------------------------
2.  -- Компонента мультиплексора "1 з N=2**m"
3.  ------------------------------------------------------------
4.  Entity MUXNx1 is
5.      -- коефіцієнт масштабування
6.      Generic     (m      :       integer := 3 );
7.      Port        (--Вхідна шина інформаційних сигналів
8.                   INn     :   in  STD_LOGIC_VECTOR ( 2**m-1 downto 0 );
9.                   --  Вхідна шина селективних сигналів
10.                  SEL     :   in  STD_LOGIC_VECTOR ( m-1 downto 0 );
11.                  -- вихідний порт
12.                  OUT1    :   out STD_LOGIC );
13. End MUXNx1;
14. ------------------------------------------------------------
15. Architecture Beh of MUXNx1 is
```

```
16.
17.     — Поведінковий опис мультиплексора MUXNx1
18.     Begin
19.
20.         OUT1 <= INn ( CONV_INTEGER( SEL ) );
21.
22.     End Beh;
23.     ——————————————————————————————————————————
24.     ——————————————————————————————————————————
25.     — Компонента мультиплексора "2 з N=2**m"
26.     ——————————————————————————————————————————
27.     Entity MUXNx2 is
28.         — коефіцієнт масштабування
29.         Generic    (m      :          integer := 3 );
30.         Port       (—Вхідна шина інформаційних сигналів
31.                     INn     :   in     STD_LOGIC_VECTOR ( 2**m-1 downto 0 );
32.                     — Вхідна шина селективних сигналів
33.                     SEL     :   in     STD_LOGIC_VECTOR ( m-1 downto 0 );
34.                     — Вихідні порти пари вибраних сигналів
35.                     OUT0    :   out    STD_LOGIC;
36.                     OUT1    :   out    STD_LOGIC );
37.     End MUXNx2;
38.     ——————————————————————————————————————————
39.     Architecture Behavioral of MUXNx2 is
40.
41.     — Декларація компоненти мультиплексора "1 з N=2**m"
42.     component MUXNx1 is
43.         Generic    (m      :          integer := 3 );
44.         Port       (INn    : in       STD_LOGIC_VECTOR ( 2**m-1 downto 0 );
45.                     SEL     : in       STD_LOGIC_VECTOR ( m-1 downto 0 );
46.                     OUT1    : out      STD_LOGIC );
47.     end component;
48.
49.     — Шина інформаційних сигналів з реверсивною адресацією
50.     signal reverse: std_logic_vector ( 2 **m-1 downto 0 );
51.
52.     — Структурний опис компоненти MUXNx2
53.     Begin
54.
55.       — Формування інформаційної шини для другого мультиплексора
56.       CONNECT: for i in 0 to  2**m_1 generate
57.           reverse( i ) <= IN n(   2**m_1_i );
58.       end generate;
59.
60.       — Підключення першого мультиплексора з прямою адресацією
61.       MUX0: MUXNx1 generic map ( m ) port map ( INn, SEL, OUT0 );
62.
63.       — Підключення другого мультиплексора зі зворотною адресацією
64.       MUX1: MUXNx1 generic map ( m ) port map ( reverse, SEL, OUT1 );
65.
66.     End Behavioral ;
67.     ——————————————————————————————————————————
```

Лістинг 5.10. VHDL-опис компонентів схеми PUF типу метелик

```
1.      ——————————————————————————————————————————
2.      — Компонента D-засувки (D-Latch)
3.      ——————————————————————————————————————————
4.      Entity DLAT is
5.          Port (
6.              — Вхідний порт даних
7.              D       :   in     STD_LOGIC;
8.              — Вхідний порт сигналу дозволу (активний рівень 1)
9.              E       :   in     STD_LOGIC;
10.             — Вхідний порт сигналу скидання в 0 (активний рівень 1)
11.             CLR     :   in     STD_LOGIC;
12.             — Вхідний порт сигналу установлення в 1 (активний рівень 1)
```

```
13.                 PRE   :   in    STD_LOGIC;
14.                 — Вихідний порт даних
15.                 Q     :   out   STD_LOGIC );
16.     End DLAT;
17.     ---------------------------------------------------------------------------
18.     Architecture Beh of DLAT is
19.
20.     — Внутрішній сигнал стану елемента пам'яті
21.     signal  s:   std_logic;
22.
23.     — Поведінковий опис D- засувки
24.     Begin
25.
26.         MAIN: process ( D, E, CLR, PRE )
27.         begin
28.             — Перевірка активності сигналу скидання
29.             if ( CLR = '1 ' ) then
30.                 s <= ' 0 ';
31.             — Перевірка активності сигналу установлення
32.             elsif ( PRE = '1 ' ) then
33.                 s <= ' 1 ' ;
34.             — Перевірка активності сигналу дозволу
35.             elsif ( E = '1 ' ) then
36.                 s <= D;
37.             end if ;
38.         end process ;
39.
40.         — Передаємо значення стану елемента пам'яті на вихідний порт
41.         Q <= s ;
42.
43.     End Beh ;
44.     ---------------------------------------------------------------------------
45.     — компонента схеми Butterfly PUF (BPUF)
46.     ---------------------------------------------------------------------------
47.     Entity BPUF is
48.           Port (
49.                 — Вхідний порт керуючого сигналу
50.                 FLY :   in    STD_LOGIC;
51.                 — Вхідний порт сигналу дозволу
52.                 E   :   in    STD_LOGIC;
53.                 — Вихідний порт даних
54.                 Q   :   out   STD_LOGIC );
55.     End BPUF;
56.     ---------------------------------------------------------------------------
57.     Architecture Struct of BPUF is
58.
59.     — Декларація компоненти D-засувки
60.     component DLAT is
61.             Port ( D      :   in    STD_LOGIC;
62.                    E      :   in    STD_LOGIC;
63.                    CLR    :   in    STD_LOGIC;
64.                    PRE    :   in    STD_LOGIC;
65.                    Q      :   out   STD_LOGIC );
66.     end component;
67.
68.     — Внутрішні сигнали міжз'єднань
69.     signal s0,s1: std_logic;
70.
71.     — Структурний опис компоненти BPUF
72.     Begin
73.
74.         — Перехресне підключення двох D-засувок
75.         DLAT0: DLAT port map ( s1, E, FLY, '0', s0 );
76.         DLAT1: DLAT port map ( s0, E, '0', FLY, s1 );
77.
78.         — Передача сигналу з виходу засувки DLAT0 на вихідний порт Q
79.         Q <= s0;
80.
81.     End Struct;
```

Лістинг 5.11. Параметрична модель компоненти RSPUF

```
1.  ------------------------------------------------------------------------
2.  -- Параметрична модель компоненти RSPUF
3.  ------------------------------------------------------------------------
4.  Entity RSPUF is
5.      Port ( RS     : in    STD_LOGIC;
6.             Q,nQ   : out   STD_LOGIC );
7.  End RSPUF;
8.  ------------------------------------------------------------------------
9.  Architecture Struct of RSPUF is
10. -- декларація компоненти 2I-НЕ
11. component NAND2 is
12.     Port ( IN0     :   in    STD_LOGIC;
13.            IN1     :   in    STD_LOGIC;
14.            OUT0    :   out   STD_LOGIC );
15. end component;
16. -- Декларація міжз'єднувальних ліній
17. signal s00,s01: std_logic;
18. signal s 10 , s 11 : std_logic;
19. -- Структурний опис компоненти RSPUF
20. Begin
21.
22.     -- Підключення двох компонент 2I-НЕ
23.     NAND20: NAND2 port map ( RS, s11, s00 );
24.     NAND21: NAND2 port map ( RS, s01 , s10 );
25.
26.
27.     -- Визначення затримок поширення сигналів
28.     -- за двома з'єднувальними лініями
29.     s01 <= s00 after 1 ns ;
30.     s11 <= s10 after 2 ns ;
31.
32.     -- Трансляція сигналів станів на вихідні порти
33.     Q <= s00;
34.     nQ <= s10;
35.
36. End Struct ;
37. ------------------------------------------------------------------------
```

**Навчальний посібник**

**Мірошник** Марина Анатоліївна,
**Курцев** Максим Сергійович

# АВТОМАТИЗАЦІЯ ПРОЕКТУВАННЯ ВБУДОВАНИХ СИСТЕМ І ПРОГРАМНИХ ЗАСОБІВ НА ПЛІС МОВОЮ ОПИСУ АПАРАТУРИ

Підписано до друку 07.07.20 р.

Формат паперу 60х84 1/16. Папір писальний.

Умовн.-друк. арк. 19,0. Тираж 50. 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.

332